vmware by Broadcom

VMware vSphere SDKs and Tools 8.0

Table of Contents

Release Notes **66**

vSphere Management SDK **66**

vSphere Management SDK 8.0 Update 3 Release Notes 66

Introduction 66

Overview 66

Distribution Kit 67

Compatibility Notices 67

New in this Release 67

Deprecated Features 68

Resolved Issues 69

Known Issues 69

Known Issues in VSLM 70

vSphere Management SDK 8.0 Update 2 Release Notes 70

Introduction 71

Overview 71

Distribution Kit 71

Compatibility Notices 72

New in this Release 72

Deprecated Features 73

Resolved Issues 73

Known Issues 73

Known Issues in VSLM 74

vSphere Management SDK 8.0 Update 1 Release Notes 75

Introduction 75

Overview 75

Distribution Kit 76

Compatibility Notices 76

New in this Release 76

Deprecated Features 77

Known Issues 77

Known Issues in VSLM 78

Known Issues in Documentation 79

Resolved Issues 79

vSphere Management SDK 8.0 Release Notes 80

Introduction 80

Overview 80

Distribution Kit 81

Compatibility Notices ... 81
New in this Release ... 81
Deprecated Features ... 82
Known Issues ... 83
Known Issues in VSLM ... 84
Known Issues in Documentation ... 84
Resolved Issues ... 85
vSphere Solutions Manager, vServices, and ESX Agent Manager 8.0 Update 2 Release Notes ... 85
Introduction ... 85
About the vSphere Solutions Manager, vServices, and ESX Agent Manager SDK ... 85
Requirements ... 85
What's New ... 86
vSphere Solutions Manager, vServices, and ESX Agent Manager 8.0 Update 1 Release Notes ... 86
Introduction ... 86
About the vSphere Solutions Manager, vServices, and ESX Agent Manager SDK ... 86
Requirements ... 86
What's New ... 87
Known Issues ... 87
vSphere Solutions Manager, vServices, and ESX Agent Manager 8.0 Release Notes ... 87
Introduction ... 87
About the vSphere Solutions Manager, vServices, and ESX Agent Manager SDK ... 87
Requirements ... 87
What's New ... 88
Resolved Issues ... 88
Known Issues ... 88
pyVmomi SDK 8.0 Release Notes ... 88
Introduction ... 88
Maintenance Patch 1 for 8.0U3 ... 88
vSphere 8.0 Update 3 ... 89
Maintenance Patch 1 for 8.0U2 ... 90
vSphere 8.0U2 Release ... 90
Maintenance Patch 2 for 8.0U1 ... 90
Maintenance Patch 1 for 8.0U1 ... 91
vSphere 8.0U1 Release ... 91
Maintenance Patch 2 for 8.0 ... 91
Maintenance Patch 1 for 8.0 ... 91
vSphere 8.0 Release ... 92
vSphere Client SDK ... 93
vSphere Client SDK 8.0 Update 3 Release Notes ... 93
Introduction ... 93

About the vSphere Client SDK ... 93
Changes and New Features ... 93
Compatibility Notices ... 96
Recently Resolved Issues ... 96
Known Issues and Workarounds ... 97
Use of /sdkTunnel Endpoint ... 97
vSphere Client SDK 8.0 Update 2 Release Notes ... 97
Introduction ... 97
About the vSphere Client SDK ... 97
Changes and New Features ... 97
Compatibility Notices ... 98
Recently Resolved Issues ... 98
Known Issues and Workarounds ... 98
Avoid Use of /sdkTunnel Endpoint ... 98
vSphere Client SDK 8.0 Update 1 Release Notes ... 98
Introduction ... 99
About the vSphere Client SDK ... 99
Changes and New Features ... 99
Compatibility Notices ... 101
Recently Resolved Issues ... 101
Known Issues and Workarounds ... 102
vSphere Client SDK 8.0 Release Notes ... 102
Introduction ... 102
Notice of Local Plug-in Deprecation ... 102
What's New ... 102
Resolved Issues ... 104
Known Issues ... 104
Virtual Disk Development Kit ... 104
Virtual Disk Development Kit 8.0 Update 3 Release Notes ... 104
Introduction ... 104
About the Virtual Disk Development Kit ... 105
Changes and New Features ... 105
Compatibility Notices ... 106
Recently Resolved Issues ... 106
Known Issues and Workarounds ... 107
Virtual Disk Development Kit 8.0 Update 2 Patch 1 Release Notes ... 107
Introduction ... 108
About the Virtual Disk Development Kit ... 108
Changes and New Features ... 108
Compatibility Notices ... 109

How to Change NFC Logging Level 109
Recently Resolved Issues 110
Known Issues and Workarounds 111
Virtual Disk Development Kit 8.0 Update 2 Release Notes 111
Introduction 111
About the Virtual Disk Development Kit 111
Changes and New Features 112
Compatibility Notices 113
How to Change NFC Logging Level 113
Recently Resolved Issues 114
Known Issues and Workarounds 115
Virtual Disk Development Kit 8.0 Update 1 Release Notes 115
Introduction 115
About the Virtual Disk Development Kit 115
Changes and New Features 116
Compatibility Notices 117
Recently Resolved Issues 118
Known Issues and Workarounds 118
Virtual Disk Development Kit 8.0 Release Notes 119
Introduction 120
About the Virtual Disk Development Kit 120
Changes and New Features 120
Compatibility Notices 121
Recently Resolved Issues 122
Known Issues and Workarounds 122
vSphere Guest SDK 122
Guest SDK 12.3.0 Release Notes 123
Introduction 123
About the Guest SDK 123
Compatibility Notices 123
Changes and New Features 123
Known Issues and Workarounds 124
Guest SDK 12.1.0 Release Notes 124
Introduction 124
About the Guest SDK 124
Compatibility Notices 124
Changes and New Features 125
Known Issues and Workarounds 125
VMware HTML Console SDK 2.2.0 Release Notes 125
Introduction 125

About the HTML Console SDK ... 125
What's New in this Release ... 126
Quick Start ... 126
Compatibility Notices ... 127
Recently Resolved Issues ... 128
Known Issues and Workarounds ... 128
ESXCLI 8.0 Release Notes ... 128
Introduction ... 128
ESXCLI 8.0 Release Notes ... 128
About ESXCLI ... 128
What's New ... 129
Supported Platforms ... 129
VMware OVF Tool ... 129
VMware OVF Tool 4.6.3 Release Notes ... 129
Introduction ... 129
About the OVF Tool ... 129
Before You Begin ... 129
What's New ... 130
Resolved Issues ... 130
Compatibility Notices ... 131
Known Issues and Workarounds ... 131
VMware OVF Tool 4.6.2 Release Notes ... 131
Introduction ... 132
About the OVF Tool ... 132
Before You Begin ... 132
What's New ... 132
Compatibility Notices ... 132
Resolved Issues ... 133
Known Issues and Workarounds ... 133
VMware OVF Tool 4.6.0 Release Notes ... 133
Introduction ... 134
About the OVF Tool ... 134
Before You Begin ... 134
What's New ... 134
Compatibility Notices ... 134
Resolved Issues ... 135
Known Issues and Workarounds ... 135
VMware OVF Tool 4.5.4 Release Notes ... 135
Introduction ... 136
About the OVF Tool ... 136

Before You Begin ... 136
What's New ... 136
Compatibility Notices ... 136
Resolved Issues ... 136
Known Issues and Workarounds ... 137
VMware OVF Tool 4.5.0 Release Notes ... 137
Introduction ... 137
About the OVF Tool ... 138
Before You Begin ... 138
What's New ... 138
Compatibility Notices ... 139
Resolved Issues ... 139
Known Issues and Workarounds ... 139
An Introduction: Getting Started with vSphere APIs and SDKs ... 141
Intended Audience ... 141
Overview of vSphere APIs and SDKs ... 141
Core vSphere APIs ... 143
vSphere Automation API ... 144
vSphere Web Services (VIM) API ... 144
vSphere Functional Extension APIs ... 145
VMware vSAN ... 145
Virtual Storage Lifecycle Management ... 146
Storage Monitoring Service ... 147
Storage Policy API ... 147
ESX Agent Manager ... 148
vCenter Server Management API ... 148
vCenter Single Sign-On ... 149
Virtual Disk API ... 150
vSphere Add-On SDKs ... 150
pyVmomi ... 150
govmomi ... 150
Getting Started with vSphere APIs and SDKs ... 151
Preliminary Resources for vSphere Programming ... 151
Authentication with vSphere APIs ... 152
Choosing Your Identity Source ... 152
Obtaining an Identifier and Unified Session ... 152
Choosing Your Authentication Method ... 153
Basic Authentication ... 153
Token-Based Authentication ... 153
Unified Management and Automation Session ... 154

Python Access to vSphere APIs 156
Installing the Python SDKs for vSphere 156
Basic Authentication Using pyvmomi 157
Creating a Hello Folder with the VIM API Using pyvmomi 158
Tagging the Folder with Automation API for Python 158
What To Do Next 160
Java Access to vSphere APIs 161
Installing the vSphere Automation SDK and Web Services SDK for Java 161
Creating and Tagging a Folder Using Java 161
Golang Access to vSphere APIs 167
Installing the govmomi SDK for vSphere 168
Basic Authentication with the VIM API Using govmomi 168
Creating a Hello Folder with the VIM API Using govmomi 169
Basic Authentication with the Automation API Using govmomi 169
Tagging the Hello World Folder with the Automation API Using govmomi 170
REST Access to vSphere APIs 171
Installing Cygwin for Windows 171
Basic Authentication to the VIM API 172
Creating a Hello Folder with the VIM API Through REST 172
Basic Authentication to the vSphere Automation API 173
Tagging the Hello World Folder with the Automation API Through REST 173
PowerCLI Access to vSphere APIs 174
Using Get-View 175
Using the vSphere Automation SDK for PowerShell 175
Installing Microsoft PowerShell and VMware PowerCLI 175
Basic Authentication to the VIM API and the vSphere Automation API 175
Creating a Hello Folder with the VIM API Using PowerCLI 175
Tagging the Hello World Folder with the Automation API Through PowerCLI 176
Use Cases for vSphere APIs 176
Virtual Disk Development Kit 177
Safekeeping – a VDDK Use Case 177
vSphere Web Services SDK Developer's Setup Guide 179
Revision History 179
Intended Audience 180
What is the vSphere Web Services SDK 180
What you need to know for the vSphere Web Services SDK 180
Programming Language Support in the vSphere Web Services SDK 180
Types of Applications That You Can Build Using This SDK 181
How You Can Download the vSphere Web Services SDK 181
Sample Authentication with Single Sign-On Configurations 181

What files are in the vSphere Web Services SDK package ... 182
SDK Version Support and Limitations of the ESXi API ... 182
Set Up to Develop in Java with the vSphere Web Services SDK ... 183
Java Development Requirements for the Web Services SDK ... 183
Set Up for Java Development for the Web Services SDK ... 183
Software Downloads for the Web Services SDK ... 183
Set Up for Development Using JAX-WS with the Web Services SDK ... 183
Batch Files and Shell Scripts for Building and Running Samples in the Web Services SDK ... 184
Import Server Certificates into the Java Keystore for the Web Services SDK ... 185
Generating Stubs and Compiling Classes for the Web Services SDK ... 186
Precompiled JAX-WS Samples in the Web Services SDK ... 186
Use the Included Build Scripts in the Web Services SDK ... 186
Running the Web Services SDK SimpleClient Sample Application to Validate Setup ... 187
Run a Sample Application Using the Provided Scripts in the Web Services SDK ... 187
Endpoint Configuration for HTTP ... 189
Modifying Service Endpoint Configurations to Support HTTP ... 190
HTTP Configuration for Web Services API Endpoint ... 190
HTTP Configuration for Web Services API Endpoint on ESXi or vCenter Server for Windows ... 190
An `endpoints.conf` File Modified To Support HTTP connections to the SDK and the MOB ... 191
HTTP Configuration for Web Services API Endpoint on vCenter Server Appliance ... 192
HTTP Configuration for vSphere Automation API Endpoint ... 193
A vapi-endpoint.conf File Modified To Support HTTP connections for API and REST ... 194
vSphere Web Services SDK Programming Guide ... 196
Revision History ... 196
Intended Audience ... 198
The vSphere Web Services SDK ... 198
vSphere Web Services SDK ... 198
SDK Developer Setup for the Web Services SDK ... 199
SDK Samples for the Web Services SDK ... 199
UML Diagrams Used in the Web Services SDK Programming Guide ... 199
vSphere Web Services Programming Model ... 199
vSphere Client-Server Architecture ... 199
Web Services API as a Web Service ... 200
WSDL Files and the Client-Side Proxy Interface ... 200
Network Access to the vSphere Web Service ... 201
Language-Specific Classes and Methods ... 201
Mapping XML Data Types to Java or JSON Data Types ... 202
Access to Managed Objects ... 203
Access to vSphere Server Data ... 204
Obtaining Information from a Server ... 204

Working with Data Structures ... 205
Accessing Property Values ... 205
Nested Properties and Property Paths in Composite Data Structures ... 205
xsd:anyType Arrays ... 206
Indexed Array and Key-Based Array Properties ... 208
Unset Optional Properties ... 208
Escape Character in Name and Path Properties ... 208
Client Applications for the Web Services API ... 209
vCenter Server Connections ... 209
Establishing a Single Sign-On Session with a vCenter Server ... 209
LoginByToken to vCenter Server By Using Java ... 210
vCenter Server Single Sign-On Session Using Java ... 210
HTTP and SOAP Header Handlers in Java ... 210
Creating the HTTP Connection in Java ... 211
Saving the vCenter Server Session Cookie ... 212
Using LoginByToken in Java ... 213
Using LoginByToken ... 214
Establishing a Session with Username and Password Credentials ... 217
Overview of a Java Sample Application for the Web Services SDK ... 217
Build a Simple vSphere Client Application for the Web Services SDK ... 217
Java Client Example for the Web Services SDK ... 218
Java Test Client Application ... 218
Web Server Session Token ... 223
Accessing the vSphere Web Services HTTP Endpoint with JAX-WS ... 223
Accessing the vSphere Server from a Web Services Client ... 226
Closing the Connection from a Web Services Client ... 227
Using the Java Samples as Reference ... 227
Obtaining a Session Token - Code Fragments from VMPromoteDisks.java ... 227
Multiple Versions of the vSphere API ... 229
Service-Versions File (vimServiceVersions.xml) ... 229
Java Sample Applications in the Web Services SDK ... 230
Java Samples in the Web Services SDK ... 230
Client Applications Using the JSON Protocol ... 230
Client Requirements for the JSON Protocol ... 230
Outlining a Client for the JSON Protocol ... 231
Building JSON Request URLs ... 231
Building JSON Request Bodies ... 232
Retrieving the Service Instance with the JSON Protocol ... 233
Excerpt from Service Content ... 234
Authenticating a JSON Client with the Session Manager ... 235

Basic Authentication for the JSON Protocol ... 235
... 236
SAML Token Authentication for a JSON Client ... 236
Mapping Methods for the JSON Protocol ... 237
Accessing Managed Object Properties with the JSON Protocol ... 237
Retrieving Nested Properties ... 237
Retrieving Managed Object Reference Properties ... 238
Using the Property Collector to Retrieve Object Properties ... 238
Using Polymorphic Types with the JSON Protocol ... 241
Python Client Example for the JSON Protocol ... 242
Datacenter Inventory ... 246
Inventory Overview for the Web Services SDK ... 246
Inventory Hierarchies and ServiceInstance ... 246
Folders in the Hierarchy ... 247
ESXi Inventory Hierarchy ... 247
Accessing Inventory Objects ... 248
Creating Inventory Objects ... 249
Privileges Required for Inventory Management ... 249
Privileges ... 250
Permissions ... 250
Managed and Standalone ESXi Hosts ... 251
Property Collector ... 252
Introduction to the PropertyCollector ... 252
Data Retrieval ... 252
Inventory Traversal and Property Selection ... 252
Gathering Data with a ViewManager Object ... 253
vSphere Data Objects for Property Collection ... 253
vSphere Methods for Property Collection ... 254
Using the PropertyCollector with RetrievePropertiesEx ... 255
Retrieve a Property from a Specified Managed Object ... 255
Retrieve Properties from a Set of Managed Objects with a View ... 257
Property Collector Program in Java for Web Services SDK ... 260
Simple PropertyCollector Example (Java) ... 261
Inventory Traversal ... 268
TraversalSpec Traversal ... 268
Traverse the Inventory By Using the Property Collector ... 269
Inventory Traversal Example in Java ... 270
Inventory Traversal in Java ... 270
SelectionSpec Traversal ... 278
Client Data Synchronization with WaitForUpdatesEx ... 287

Property Filtering ... 287
WaitForUpdatesEx ... 288
Server Data Transmission ... 289
PropertyCollector Performance ... 290
SearchIndex ... 290
Authentication and Authorization ... 290
Objects for Authentication and Authorization Management ... 290
Authentication and Authorization for ESXi and vCenter Server ... 291
ESXi User Model ... 292
vCenter Server User Model ... 292
vSphere Security Model ... 292
Setting Up Users, Groups, and Permissions ... 294
Obtaining User and Group Information from UserDirectory ... 294
`RetrieveUserGroups` Method ... 295
Managing ESXi Users with HostLocalAccountManager ... 295
Methods Available for Local Account Management ... 296
Create a Local User Account on an ESXi System ... 296
Managing Roles and Permissions with AuthorizationManager ... 296
Using Roles to Consolidate Sets of Privileges ... 297
Modifying Sample Roles to Create New Roles ... 298
Granting Privileges Through Permissions ... 298
Authenticating Users Through SessionManager ... 301
Using VMware Single Sign On for vCenter Server Sessions ... 301
Using the Credential Store for Automated Login ... 302
Credential Store Libraries ... 302
Credential Store Methods ... 302
Credential Store Backing File ... 303
Credential Store File Format ... 303
Credential Store Samples ... 303
Specifying Roles and Users with the Credential Store ... 304
Managing Licenses with LicenseManager ... 304
Retrieve Information ... 305
Add Licenses ... 305
Remove Licenses ... 305
Hosts ... 305
Host Management Objects ... 305
Retrieving Host Information ... 305
Configuring and Reconfiguring Hosts ... 306
Managing the Host Lifecycle ... 307
Reboot and Shutdown ... 307

Using Standby Mode ... 307
Disconnecting and Reconnecting Hosts ... 307
Querying and Changing the Host Time ... 308
Querying Virtual Machine Memory Overhead ... 308
Storage ... 308
Storage Management Objects ... 309
Introduction to Storage ... 309
How Virtual Machines Access Storage ... 310
Datastores ... 311
Choosing the Storage API to Use ... 312
Managed Objects for Working with Storage ... 312
Configuring Disk Partitions ... 313
Multipath Management ... 314
Configuring iSCSI Storage ... 314
Configure the VMkernel To Support Software iSCSI ... 315
Configure iSCSI Initiators ... 315
Creating and Managing Datastores ... 316
Accessing Datastores ... 316
Creating and Modifying a VMFS Datastore ... 317
Removing and Updating Datastores ... 318
Managing VMFS Datastores with HostStorageSystem ... 318
Managing VMFS Volume Copies (Resignaturing) ... 319
Resignaturing Volumes with **`ResignatureUnresolvedVmfsVolume_Task`** ... 319
Managing Diagnostic Partitions ... 320
Retrieving Diagnostic Partition Information ... 320
Create a Diagnostic Partition ... 320
Sample Code for Storage ... 321
vSphere Networks ... 321
Virtual Switches ... 321
Port Groups ... 321
Virtual Machine Network Interfaces ... 322
VMkernel Network Interfaces ... 322
Physical Network Adapter (pnic) ... 322
Using a Distributed Virtual Switch ... 322
Distributed Virtual Switch Configuration ... 323
Backup, Rollback, and Query Operations ... 324
VMware Standard Virtual Switch ... 324
Configuring a Standard Virtual Switch ... 325
vNetwork Standard Switch Environment ... 325
Setting Up Networking with vSS ... 326

Defining the Host Network Policies 328
NIC Teaming 329
Setting Up IPv6 Networking 329
IPv6 Setup 330
Adding Networking Services 330
Sample Code for Networks 331
Virtual Machine Configuration 331
VirtualMachine Management Objects and Methods 332
Creating Virtual Machines and Virtual Machine Templates 332
Creating a Virtual Machine Using VirtualMachineConfigSpec 332
Cloning a Virtual Machine 334
Cloning a Virtual Machine 334
Continuous Virtual Machine Clone 334
Creating Virtual Machine Templates 335
Converting a Template to a Virtual Machine 335
Accessing Information About a Virtual Machine 335
Configuring a Virtual Machine 337
Name and Location 338
Hardware Version 338
Boot Options 338
Operating System 339
CPU and Memory Information 339
Networks 342
Fibre Channel NPIV Settings 342
File Locations 343
Adding Devices to Virtual Machines 343
Performing Virtual Machine Power Operations 345
Registering and Unregistering Virtual Machines 346
Customizing the Guest Operating System 346
Installing VMware Tools 347
Upgrading a Virtual Machine 347
Virtual Machine Management 347
Virtual Machine Migration 347
Cold Migration 348
Migration with vMotion 348
Using Storage vMotion 348
Snapshots 349
Creating a Snapshot 349
Reverting to a Snapshot 350
Deleting a Snapshot 350

Linked Virtual Machines ... 350
Linked Virtual Machines and Disk Backings ... 350
Creating a Linked Virtual Machine ... 351
Instant Clone Virtual Machines ... 353
Removing Snapshots and Deleting Linked Virtual Machines ... 357
Relocating a Virtual Machine in a Linked Virtual Machine Group ... 357
Relocating a Linked Virtual Machine ... 358
Relocating Multiple Linked Virtual Machines ... 358
Promoting a Virtual Machine's Disk ... 358
Performing Advanced Manipulation of Delta Disks ... 359
Adding a Delta Disk Backing ... 359
Virtual Machine Guest Operations ... 360
Authenticating with the Guest Operating System ... 360
Running Guest OS Operations ... 360
Java Source Code Samples ... 362
Guest Operating System Customization ... 362
Guest Customization for Stopped Virtual Machines ... 364
Guest Customization Using cloud-init ... 364
Prepare a Virtual Machine Template for Raw Data cloud-init Customization ... 365
Prepare a cloud-init Metadata File ... 366
Prepare a cloud-init Userdata File ... 367
Initiate Raw Data cloud-init Customization with the vSphere Web Services API ... 368
Guest Network Customization for Instant Clone Virtual Machines ... 369
Installing the Guest Customization Engine ... 369
Disconnecting Virtual NICs ... 370
Customizing Guest Network Settings for Running Virtual Machines ... 370
Reconnecting Virtual NICs in a Running Virtual Machine ... 371
Restarting the Guest Network After Customization ... 371
Recovering from Guest Network Customization Errors ... 372
Application-Dependent Customization ... 372
Resetting the Network Stack in a Running Virtual Machine ... 372
Resetting the Network Stack in a Linux Virtual Machine ... 373
Resetting the Network Stack in a Windows Virtual Machine ... 373
Virtual Machine Security ... 374
How Virtual Machine Encryption Protects a Datacenter ... 374
What Keys are Used ... 374
What Is Encrypted ... 374
What Is Not Encrypted ... 375
Who Can Perform Cryptographic Operations ... 376
How Can I Perform Cryptographic Operations ... 376

vSphere Virtual Machine Encryption Components ... 376
Key Management Server ... 377
vCenter Server ... 377
ESXi Hosts ... 377
Encryption Process Flow ... 378
Prerequisites and Required Privileges for Encryption Tasks ... 378
Cryptography Privileges and Roles ... 378
Host Encryption Mode ... 379
Encrypted vSphere vMotion ... 379
API Methods for vSphere Virtual Machine Encryption ... 379
vSphere API Methods for KMS Management ... 379
API Methods to Prepare an ESXi Host ... 380
vSphere API Methods for Cryptographic Operations ... 381
SPBM API Methods for Encryption ... 381
Workflows for vSphere Virtual Machine Encryption ... 381
Set Up the Key Management Server Cluster ... 382
Create an Encryption Storage Policy ... 386
Java program to set storage policy for encryption ... 386
Create an Encrypted Virtual Machine ... 390
Clone an Encrypted Virtual Machine ... 392
Clone an encrypted virtual machine ... 392
Encrypt an Existing Virtual Machine or Disk ... 392
Encrypt a virtual machine based on storage profile ... 392
Decrypt an Encrypted Virtual Machine or Disk ... 393
Decrypt a virtual machine based on storage profile ... 394
Encrypt with Different Keys for VM and Disk ... 394
Different encryption keys for VM home and virtual disk ... 395
Recrypt (Rekey) Encrypted Virtual Machines ... 396
Query Crypto Key In-Use Status ... 397
Encrypted vSphere vMotion ... 398
Virtual Disk Manager ... 398
Best Practices for Virtual Machine Encryption ... 398
Key Lifecycle and Removal ... 398
Be Careful with Force Remove ... 398
Remove Keys on ESXi host ... 398
Carefully Manage Differential Keys ... 398
Rename During Registration ... 399
Encryption on vSAN Datastores ... 399
Unlocking Encrypted Virtual Machines ... 399
Configuring Advanced Options ... 399

Querying Advanced Options 399
Setting Advanced Options 399
Physical Platform Resources 400
USB 400
Storage 400
Physical CD-ROM or DVD 400
Virtual GPU (vGPU) 401
PCI Passthrough 401
Physical Network 401
Virtual Applications: the vApp 401
About Virtual Applications 401
Management Overview 402
Direct and Linked Children 402
OVF Packages 402
Creating a VirtualApp 403
Managing VirtualApp Children 403
Exporting a Virtual Application 404
VirtualApp and OvfManager Methods 405
VirtualApp Data Structures 405
OvfManager Data Structures 406
Example of Generating an OVF Package 407
Importing an OVF Package 410
Virtual Application Life Cycle 410
Powering a Virtual Application On or Off 410
Unregistering a Virtual Application 411
Suspending a Virtual Application 411
Destroying a Virtual Application 411
Resource Management 411
Resource Management Objects 411
Introduction to Resource Management 412
Resource Allocation 412
Resource Pool Hierarchies 412
Cluster Overview 412
Creating and Configuring Resource Pools 413
Configuring Reservation and Limit for Resource Pools 413
Configuring Priority Shares for Resource Pools 416
Deleting Child Resource Pools 426
Moving Resource Pools or Virtual Machines Into a Resource Pool 426
Introduction to vSphere Clusters 426
VMware DRS 427

VMware HA 427
VMware HCI 427
Creating and Configuring Clusters 428
Creating a Cluster 428
Adding a Host to a Cluster 429
Reconfiguring a Cluster 429
Managing DRS Clusters 429
Managing HA Clusters 430
Primary and Secondary Hosts 430
Failure Detection and Host Network Isolation 430
Using VMware HA and DRS Together 431
Tasks and Scheduled Tasks 431
Creating Tasks 431
Session Persistence 431
Cancelling a Task 432
Using TaskInfo to Determine Task Status 432
Monitoring TaskInfo Properties 433
Displaying TaskInfoState Values for Tasks in recentTask Array 433
Sample Run of the TaskList Java Application 434
Accessing and Manipulating Multiple Tasks 435
Task Monitoring Example Using the Listview Object 435
Gathering Data with a TaskManager Interface 448
Understanding the ScheduledTaskManager Interface 449
Scheduling Tasks 450
Cancelling a Scheduled Task 454
Using a TaskHistoryCollector 454
Creating a TaskHistoryCollector Filter 455
Managing the HistoryCollector 455
Sample Code for Tasks 455
Events and Alarms 456
Event and Alarm Management Objects 456
Understanding Events 456
Managing Events with EventManager 457
Event Data Objects 457
Formatting Event Message Content 458
Creating Custom Events 458
Using an EventHistoryCollector 460
Creating an EventHistoryCollector Filter 460
Managing the HistoryCollector 455
Using Alarms 461

Obtaining a List of Alarms ... 461
Creating an Alarm ... 462
Defining Alarms Using the AlarmSpec Data Object ... 462
Specifying Alarm Trigger Conditions with AlarmExpression ... 463
Specifying Alarm Actions ... 464
Deleting or Disabling an Alarm ... 465
Sample Code for Events and Alarms ... 465
vSphere Performance ... 466
vSphere Performance Data Collection ... 466
PerformanceManager Objects and Methods ... 468
Retrieving vSphere Performance Data ... 469
Performance Counter Example (QueryPerf) ... 470
Large-Scale Performance Data Retrieval ... 480
Using the QueryPerf Method as a Raw Data Feed ... 480
Comparison of Query Methods ... 481
Retrieving Summary Performance Data ... 481
Performance Counter Metadata ... 482
PerfCounterInfo ... 482
Performance Intervals ... 482
ESXi Server Performance Intervals ... 483
vCenter Server Performance Intervals ... 483
vSphere Performance and Data Storage ... 484
Modifying Historical Intervals ... 484
Modifying Performance Counter Collection Levels ... 484
Sample Code for Performance ... 485
Diagnostics and Troubleshooting ... 486
Troubleshooting Best Practices ... 486
Overview of Configuration Files and Log Files ... 486
ESXi Log File ... 487
Sample ESXi Log (`hostd.log`) Data ... 487
Virtual Machine Log Files ... 488
VMkernel Availability Report ... 488
vCenter Server Log Files ... 490
Modifying the Log Level to Obtain Detailed Information ... 490
Setting the Log Level on ESXi Systems ... 490
Generating Logs ... 491
Setting the Log Level on vCenter Server Systems ... 491
Using DiagnosticManager ... 491
Using the MOB to Explore the DiagnosticManager ... 493
Generating Diagnostic Bundles ... 493

Export Diagnostic Data By Using the vSphere Client ... 494
Managed Object Browser ... 494
Using the MOB to Explore the Object Model ... 494
Accessing the MOB ... 494
Using the MOB to Navigate the vSphere Object Model ... 495
Using the MOB to Invoke Methods ... 495
Passing Primitive Datatypes to Method ... 496
Passing Arrays of Primitives to Methods ... 496
Passing Complex Structures to Methods ... 496
Using the MOB Along With the API Reference ... 500
HTTP Access to vSphere Server Files ... 500
Introduction to HTTP Access ... 500
URL Syntax for HTTP Access ... 501
Datastore Access (/folder) ... 501
Host File Access to vSphere Server Files (/host) ... 503
Update Package Access (/tmp) ... 503
Privilege Requirements for HTTP Access ... 504
Sample Program Overview ... 504
Java Sample Programs (JAXWS Bindings) ... 504
VMware Storage Policy SDK Programming Guide ... 509
Revision History ... 509
Intended Audience ... 509
Sample Code ... 509
API Reference ... 509
VMware Storage Policies ... 509
Storage Capabilities ... 509
Virtual Machine Storage ... 510
Storage Capability Profiles ... 510
Storage Policy Operations ... 510
SPBM Managed Objects ... 510
Storage Policy API and vSphere API ... 511
Access to the Storage Policy Server ... 511
Access to Storage Policy API Methods ... 511
Storage Profile Queries ... 512
VMware Storage Policy SDK ... 513
VMware Storage Policy SDK Examples ... 514
Storage Policy Server Connection ... 516
About Storage Policy Server Sessions ... 516
Establish a Connection with the VMware Storage Policy Server ... 516
Server URLs for Basic Connection ... 517

Create the Storage Policy Server Connection 517
Storage Policy Server Connection 517
Establish the vCenter Session Connection for the Local Instance 518
vCenter Server Connection 518
vVol Based Storage Profiles 519
vVol Storage Policy Architecture 519
SPBM Calling Sequence for vVols 519
SPBM Data Objects 520
vSAN Based Storage Profiles 520
Create a vSAN Requirements Profile 521
Create an Individual Storage Requirement 521
Create a Storage Profile 522
vSAN Storage Profile Creation 523
Virtual Machine Storage Profiles 524
Retrieve an Existing Storage Profile from the Server 524
Retrieving a Storage Profile 525
Apply the Storage Profile to a Virtual Machine 526
Associating a Storage Profile with a Virtual Machine 526
Tag-Based Storage Profiles 527
Creating a Tag-Based Storage Profile 527
Creating a Storage Profile 528
Tag-Based Storage Profile Creation 528
Retrieving Tag Metadata 530
getTagCategoryMeta (PbmUtil Package) 530
Policy Rules 531
Create a Stand-Alone Virtual Disk with an Attached Storage Encryption Policy 531
Legacy Storage Profiles 533
VASA 1.0 Storage Capability Upgrade 533
vSphere Web Client User Label Conversion 534
vCenter Single Sign-On Client Example 536
vCenter Single Sign-On Token Request Overview 536
Using Handler Methods for SOAP Headers 536
Sending a Request for a Security Token 539
Acquiring a vCenter Single Sign-On Token – Sending the Request 540
vCenter LoginByToken Example 542
vCenter Server Single Sign-On Session 543
HTTP and SOAP Header Handlers 543
LoginByToken Sample Code 544
Saving the vCenter Server Session Cookie 544
Saving the vCenter Server Session Cookie 545

Using LoginByToken 547
Using LoginByToken 548
Restoring the vCenter Server Session Cookie 549
Restoring the vCenter Server Session 549
vCenter Single Sign-On Programming Guide 551
Revision History 551
Intended Audience 551
Single Sign-On in the vSphere Environment 551
vCenter Single Sign-On Overview 551
vCenter Single Sign-On Client API 553
Acquiring a SAML Token from a vCenter Single Sign-On Server 553
vCenter Single Sign-On Security Policies 554
Connecting to a vCenter Single Sign-On Server 556
vCenter Single Sign-On Token Delegation 556
vCenter Single Sign-On Token Lifetime - Clock Tolerance 556
vCenter Single Sign-On Challenge (SSPI) 556
vCenter Single Sign-On SOAP Message Structure 557
vCenter Single Sign-On SDK 557
vCenter Single Sign-On SDK Examples 558
vCenter Single Sign-On API Reference 559
vCenter Single Sign-On Client API Methods 560
Issue 560
Renew 560
Validate 561
Challenge 562
vCenter Single Sign-On API Data Structures 562
RequestSecurityTokenType 562
RequestSecurityTokenResponseCollectionType 564
RequestSecurityTokenResponseType 564
LifetimeType 565
RenewingType 565
KeyTypeOpenEnum 566
UseKeyType 566
ParticipantsType 566
ParticipantType 566
EndpointReference 567
BinaryExchangeType 567
AdviceType 567
AttributeType 567
vCenter Single Sign-On Client Example (JAX-WS) 568

vCenter Single Sign-On Token Request Overview ... 568
Using Handler Methods for SOAP Headers in Java ... 569
Acquiring a vCenter Single Sign-On Token – Soap Handlers ... 570
Sending a Request for a Security Token in Java ... 572
Acquiring a vCenter Single Sign-On Token – Sending the Request ... 572
LoginByToken Example (JAX-WS) ... 575
vCenter Server Single Sign-On Session ... 575
HTTP and SOAP Header Handlers in Java ... 575
LoginByToken Sample Code in Java ... 576
Steps to Connect with a vSphere Server ... 577
Import the Necessary Java Packages for LoginByToken ... 577
Import the Packages that Are Needed ... 577
Get the VimPort for LoginByToken in Java ... 578
Getting the VimPort ... 578
Retrieve the Service Content in Java ... 579
Call `retrieveServiceContent` ... 579
Invoke the `loginByToken` method in Java ... 579
Call the `loginByToken` Method ... 579
Extract the Session Cookie in Java ... 580
Extract the Session Cookie ... 580
Inject the Session Cookie Back Into the Request in Java ... 580
Inject the Session Cookie Back Into the Request ... 580
Additional Information ... 581
Using a Proxy with vSphere Virtual Serial Ports ... 582
Revision History ... 582
Intended Audience ... 582
Remote Serial Port Access with a Virtual Serial Port Proxy ... 582
Virtual Serial Port Proxy Capabilities ... 583
Proxy as a Virtual Serial Port Concentrator ... 583
Network Serial Ports ... 584
Backing Options and Backing Information for Network Serial Ports ... 584
Network Connections Between a Virtual Machine, a Proxy, and a Remote System ... 585
Determining Backing Options ... 586
Create a Virtual Machine with a Network Serial Port ... 586
VMware Telnet Session ... 587
VMware Telnet Commands ... 587
Establish a VMware Telnet Extension Session ... 588
Proxy Failure Detection (TCP keepalive) ... 589
vMotion and Persistent Serial Port Connections ... 589
VMware Telnet Extension Commands for Proxy Communication ... 591

Command Syntax ... 591
VMware Telnet Extension Option ... 592
Option Subnegotiation ... 593
Unknown Command Response ... 594
Proxy Operation ... 594
vMotion Notification ... 595
Virtual Machine Identification ... 597
Developing and Deploying vSphere Solutions, vServices, and ESX Agents ... 599
Intended audience ... 599
Related documentation ... 599
How to develop vSphere solution ... 599
Download the vSphere ESX Agent Manager SDK ... 600
What is in the vSphere Extension APIs ... 600
How to use the vCenter Server Extensions ... 601
Standard Tabs in the vCenter Server Extensions ... 601
What are the vSphere ESX agents and agencies ... 601
vSphere ESX Agent Manager ... 602
What are vServices ... 602
vService Manager ... 602
vCenter Extension vService ... 603
How do I create vSphere solutions ... 603
Contents and structure of an extension ... 603
Key objects in the vSphere API for extension and solution development ... 604
How do I register an extension with vCenter Server ... 605
Create the program that manages the extension ... 605
Connect the extension to vCenter Server ... 606
Set the extension key ... 609
Set the extension product information ... 609
Set the extension name and localization resources ... 611
Identify the virtual machines or vApps that an extension manages ... 615
Set the types of the virtual machines or vApps that the extension manages ... 617
Set the description for a type of virtual machine or vApp that a solution manages ... 618
Unregister the extension from vCenter Server ... 618
How do I integrate solutions with vSphere ESX Agent Manager ... 619
Benefits of integrating solutions with ESX Agent Manager ... 619
Monitoring solutions ... 619
User privileges ... 619
Integration with vSphere features ... 620
ESX Agent virtual machines and High Availability ... 620
vSphere ESX Agent Manager architecture ... 621

Configuration components of ESX agencies ... 621
ESX agency scope ... 621
ESX agency goal state ... 621
ESX agency status ... 622
Status of ESX agency scope changes ... 623
Requirements for itegrating a solution with ESX Agent Manager ... 624
How do I authenticate against ESX Agent Manager ... 624
How do I configure the availability of ESX agent virtual machines ... 625
Monitoring and resolving ESX agent issues ... 625
How does the ESX Agent Manager remediate issues ... 625
How do I integrate a solution with ESX Agent Manager ... 626
Connect a solution to ESX Agent Manager ... 626
Configure an ESX agency and ESX agents ... 628
Create an ESX agency ... 631
Update the Agency scope of a solution ... 633
Change the goal state of an agency ... 635
Delete an ESX agency ... 637
Resolve ESX agent issues ... 638
How do I integrate an extension with the vCenter Extension vService ... 640
Integrate a virtual machine with the vCenter Extension vService ... 640
Configure the OVF descriptor file ... 640
vService dependency section in the OVF descriptor file ... 643
Provide a script in the Extension virtual machine to register as a vCenter Extension ... 643
`vServiceEnvironmentSection` in the OVF environment XML file ... 645
Deploy an extension in the vSphere Client by using the vCenter Extension vService ... 645
vCenter Extension vService XML schema ... 646
VMware vSphere Automation REST API Programming Guide ... 649
Intended Audience ... 649
Introduction to the vSphere Automation REST API ... 649
Understanding the vSphere Automation API ... 649
How Does the vSphere Automation API Work ... 649
The vSphere Automation API in Context ... 650
Which REST Clients to Use with the vSphere Automation API ... 652
Why Use the vSphere Automation API Through REST ... 652
vSphere Automation API Base Path ... 652
vSphere Automation API Requests ... 652
HTTP Method ... 653
URL ... 653
HTTP Headers ... 653
Request Body ... 654

vSphere Automation API Responses ... 654
Supplementing the vSphere Automation APIs ... 657
vSphere Client ... 657
vCenter Server Management Interface ... 657
Direct Console User Interface ... 657
vCenter Server Appliance Bash Shell ... 657
Data Center CLI ... 657
PowerCLI ... 657
vSphere Web Services API ... 658
Authentication Mechanisms ... 658
Choosing Your Identity Source ... 658
Default or External Identity Provider ... 658
Basic or Token-Based Authentication ... 658
Basic Authentication ... 658
Token-Based Authentication ... 659
Authentication Terminolgy ... 659
Authenticate to vCenter Server with vCenter Single Sign-On Credentials ... 660
Identity Provider Federation for vCenter Server ... 661
vCenter Server Identity Provider Federation Basics ... 661
Why Is Identity Provider Federation Useful ... 661
vCenter Server External Identity Provider Support ... 661
Federation to Microsoft Active Directory Federation Services (AD FS) ... 662
Federation to Okta and Azure AD Through VMware Identity Broker - vCenter Server ... 662
What is OAuth 2.0 ... 662
OAuth 2.0 Grant Types ... 662
What Is OpenID Connect (OIDC) ... 663
AD FS Federation Workflows ... 663
Federate vCenter Server to Microsoft Active Directory Federation Services (AD FS) ... 663
Authenticate to an AD FS-Federated vCenter Server by Using the Password Grant Type ... 664
Authenticate to an AD FS-Federated vCenter Server by Using the Authorization Code Grant Type ... 665
VMware Identity Broker - vCenter Server Workflows ... 666
What Is VMware Identity Broker - vCenter Server ... 666
Configure vCenter Server Identity Provider Federation to Okta or Azure AD ... 666
OAuth Authentication to Your Federated vCenter Server, App, or Script ... 667
Federate Apps or Scripts to an External Identity Provider by Using Your VMware Identity Broker - vCenter Server Configuration ... 667
Obtain JWT Tokens from VMware Identity Broker - vCenter Server by Using the Password Grant Type ... 668
Obtain JWT Tokens from VMware Identity Broker - vCenter Server by Using the Authorization Code Grant Type ... 670
Obtain JWT Token from VMware Identity Broker - vCenter Server by Using the Client Credentials Grant Type ... 671

Exchange JWT Tokens for a SAML Token and Obtain a Session Identifier ... 672
ESXi Hosts ... 675
How to Retrieve Information About ESXi Hosts ... 675
How to Add a Standalone ESXi Host to vCenter Server ... 675
How to Disconnect and Reconnect ESXi Hosts ... 676
How to Configure ESXi Entropy ... 676
Querying Entropy Data on a Host ... 676
Adding External Entropy Data to a Host ... 677
Managing the Life Cycle of Hosts and Clusters ... 677
vSphere Lifecycle Manager Terms ... 678
vSphere Lifecycle Manager Terminology ... 678
vSphere Lifecycle Manager Overview ... 679
Depot Manager ... 680
Image Manager ... 680
Coordinator and ESXi Updater Managers ... 680
Options for Managing the ESXi Life Cycle ... 681
vSphere Lifecycle Manager Features ... 681
Required Privileges for Using vSphere Lifecycle Manager and vSphere Configuration Profiles ... 681
vSphere Lifecycle Manager Privileges For Using Images ... 681
Required Privileges for Using vSphere Configuration Profiles ... 684
Software Depots ... 685
Types of Software Depots ... 685
Working with Online Depots ... 687
Working with UMDS Depots ... 687
Synchronizing Software Depots ... 688
Working with Offline Depots ... 688
Managing Depot Overrides ... 689
Inspecting Depot Contents ... 689
Enabling a Cluster to Use a Software Specification ... 689
Creating a Cluster with Enabled vSphere Lifecycle Manager ... 689
Enabling an Existing Cluster to Use vSphere Lifecycle Manager ... 690
Enabling a Standalone Host to Use a Software Specification ... 691
Working with Draft Software Specifications ... 691
Creating a Draft Software Specification ... 691
Editing a Draft Software Specification ... 692
Validating the Draft Software Specification ... 692
Committing the Draft Software Specification ... 692
Working with Desired Software States ... 693
Exporting and Importing a Desired State ... 693
Checking the Compliance Against the Desired State ... 694

Hardware Compatibility Data ... 695
Checking the Hardware Compatibility of an ESXi Host ... 695
Configuring Remediation Settings ... 695
How Remediation Policies Overrides Work ... 695
Editing Global or Cluster- and Host-Specific Remediation Policies ... 696
Remediation Policy Options for Clusters ... 696
Remediation Policy Options for Standalone Hosts ... 698
Remediating an ESXi Cluster and a Standalone Host ... 699
Integrate Third-Party Solutions with vSphere Lifecycle Manager ... 699
Prerequisites ... 700
Integrate a Third-Party Solution to Work with vSphere Lifecycle Manager ... 700
Results ... 701
What to do next ... 701
Enable vSphere Lifecycle Manager on a Cluster Managed with Baselines ... 701
vSphere Lifecycle Manager APIs Equivalent to the vSphere Host Patch Manager APIs ... 702
Virtual Machine Configuration and Management ... 702
Creating Virtual Machines ... 703
Creating a Virtual Machine Without a Clone or Template ... 703
Configuring Virtual Machines ... 703
Name and Location ... 703
Hardware Version ... 704
Boot Options ... 704
Guest Operating System ... 704
CPU and Memory ... 704
Networks ... 705
Managing Virtual Machines ... 705
Filtering Virtual Machines ... 705
Installing VMware Tools ... 705
Performing Virtual Machine Power Operations ... 706
Registering and Unregistering Virtual Machines ... 706
Managing Data Sets ... 707
Data Set Operations ... 707
Using Content Libraries ... 708
What Is a Content Library ... 708
Content Library Types ... 708
Content Library Items ... 709
Content Library Storage ... 709
Querying Content Libraries ... 711
Listing All Content Libraries ... 711
Listing Content Libraries of a Specific Type ... 711

Listing Content Libraries by Using Specific Search Criteria 711
Working with Content Libraries 711
Creating a Local Content Library 711
Publishing an Existing Content Library 711
Subscribing to a Content Library 712
Synchronizing a Subscribed Content Library 712
Editing the Settings of a Content Library 712
Removing the Content of a Subscribed Library 712
Delete a Content Library 712
Working with Library Items 712
Creating an Empty Library Item 713
Querying Library Items 713
Editing the Settings of a Library Item 713
Uploading a File from a Local System to a Library Item 713
Upload a File from a URL to a Library Item 713
Downloading Files to a Local System from a Library Item 714
Synchronizing a Library Item in a Subscribed Content Library 714
Removing the Content of a Library Item 714
Deleting a Library Item 714
Content Library Support for OVF and OVA Packages 714
Working with OVF and OVA Packages in a Content Library 715
Creating Virtual Machines and vApps from Templates in a Content Library 716
Create a VM Template in a Content Library from a Virtual Machine 716
Create an OVF Template in a Content Library from a Virtual Machine or vApp 717
Deploy a Virtual Machine from a VM Template in a Content Library 717
Deploy a Virtual Machine or vApp from an OVF Template in a Content Library 718
vSphere Tag Service 719
Creating vSphere Tags 719
Creating a Tag Category 719
Creating a Tag 719
Creating Tag Associations 721
Assign the Tag to a Content Library 721
Assign a Tag to a Cluster 721
Updating a Tag 721
Using Tags to Create and Manage Compute Policies 723
Create a Compute Policy 723
vSphere Security 723
Managing Certificates 723
Certificate Management Operations 723
Add a Root Certificate to vCenter Server 725

Delete a Root Certificate from vCenter Server ... 726
Change the Machine SSL Certificate of vCenter Server ... 727
Refresh the vCenter Server STS Signing Certificate with a VMCA-Issued Certificate ... 728
Set a Custom STS Signing Certificate to vCenter Server ... 729
Managing vSphere Native Key Provider ... 730
Retrieve vSphere Native Key Providers ... 730
Manage vSphere Native Key Providers ... 730
Export and Import vSphere Native Key Provider Configuration ... 731
Retrieve vSphere Native Key Providers on a Host ... 731
vSphere Trust Authority ... 731
Configure a vSphere Trust Authority Cluster ... 731
Configure Key Providers ... 731
Establish Trust Between Key Provider and Key Server ... 732
Configure Trusted TPMs of Attested ESXi Hosts on a Cluster Level ... 732
Configure Trusted ESXi Builds on a Cluster Level ... 733
Retrieve vSphere Trust Authority Components Information ... 733
Configure vSphere Trust Authority Components ... 733
Configure vSphere Trust Authority Components for Trusted Clusters ... 734
Establish Trust Between Hosts in a vSphere Trust Authority Cluster and a Workload vCenter Server ... 735
Check Trusted Cluster Health ... 735
Remediate a Trusted Cluster ... 736
Managing Host Hardware Security ... 736
Retrieve Host Hardware TPM Information ... 736
Manage Host Hardware TPM Endorsement Keys ... 736
Register an SGX Host ... 737
vSphere Supervisor Configuration and Management ... 737
vSphere Supervisor Terminology ... 737
vSphere Supervisor Basic Terms ... 737
vSphere Supervisor Components and Services ... 738
How to Configure and Manage a Supervisor ... 739
Persistent Storage in vSphere Supervisor ... 740
Supervisor Networking ... 741
Enable vSphere Supervisor on a Cluster with NSX as the Networking Stack ... 743
Enable vSphere Supervisor on a Cluster with the vSphere Networking Stack ... 748
Upgrading a Supervisor ... 752
Monitoring the Enable and Upgrade Supervisor Operations ... 752
Reconfiguring a Supervisor ... 752
Deactivating a Supervisor ... 752
Content Libraries in vSphere Supervisor ... 753
Creating, Securing, and Synchronizing Content Libraries for TKG Releases ... 753

Creating and Managing Content Libraries for VM Provisioning in vSphere Supervisor ... 754
Associating a Content Library with a Namespace ... 754
Managing Namespaces on a Supervisor ... 755
Create a vSphere Namespace ... 755
Updating the Namespace Configuration ... 757
Configuring the Access to a Namespace ... 757
Self-Service Namespace Management ... 757
Virtual Machines in vSphere Supervisor ... 758
Create a VM Class in vSphere Supervisor ... 759
Editing or Removing a VM Class from Your Environment ... 760
Associating a VM Class with a vSphere Namespace ... 760
vCenter Server Management ... 761
Authorization Model for Administration of vCenter Server ... 761
Authorization Model Mapping to the vCenter Single Sign-On Domain ... 761
Using the Operator Role ... 761
Using the Admin Role ... 761
Using the SuperAdmin Role ... 762
Performing Privilege Checks Operations ... 762
Filtering Privilege Checks ... 762
Using Advanced Settings ... 763
vCenter Server Installation and Setup ... 763
Install Stage 2 ... 763
Setting Up a Newly Installed vCenter Server Instance ... 763
Workflows for Install Stage 2 ... 765
File-Based Backup and Restore of vCenter Server ... 767
Backing up vCenter Server ... 767
Backup and Restore Protocols for vCenter Server ... 768
Calculate the Size Needed To Store the Backup File ... 768
Back up a vCenter Server Instance by Using the API ... 769
Schedule a Backup Job ... 770
Restoring vCenter Server ... 770
Authentication When Restoring a vCenter Server Instance ... 771
Availability of Services While Restoring a vCenter Server Instance ... 771
Restore a vCenter Server Instance by Using the API ... 771
Reconcile a vCenter Server Instance with Nodes in Embedded Linked Mode ... 772
Troubleshooting for vCenter Server Installation or Deployment ... 772
Managing Support Bundles ... 772
vCenter Server Upgrade ... 774
Upgrade Stage 2 ... 775
Upgrading a vCenter Server Instance ... 775

Workflows for Upgrade Stage 2 ... 777
Historical Data Transfer ... 779
Deferred Import ... 780
Historical Data Import Errors ... 781
vCenter Server Configuration ... 782
Health Monitoring of vCenter Server ... 782
Check Overall System Health of vCenter Server ... 782
Capacity Monitoring ofvCenter Server ... 783
Check Database Usage in vCenter Server ... 785
List Storage Consumption By Data Type in vCenter Server ... 785
Managing the Global FIPS Compliance ... 785
Performing Infrastructure Profile Management Operations ... 786
Patching and Updating vCenter Server Deployments ... 786
Planning vCenter Server Updates ... 786
Performing Discovery and Planning Operations ... 787
List Available Products and Manage Associated Products ... 788
List Available Updates ... 788
Retrieve a Report ... 788
Updating vCenter Server ... 789
ApplyingvCenter Server Software Updates ... 789
vCenter Server Software Update Workflow ... 790

VMware vSphere Automation SDKs Programming Guide ... 792

Intended Audience ... 792
Introduction to the vSphere Automation SDKs ... 792
vSphere Automation SDK Overview ... 792
SDK Developer Setup ... 793
API Explorer ... 793
SDK Samples ... 793
Supported Programming Languages ... 794
What Are the Components and Services of the vSphere Environment ... 794
Authentication Services Installed with vCenter Server ... 794
Components Installed with vCenter Server ... 794
Retrieving Service Endpoints ... 795
Workflow for Retrieving Service Endpoints ... 795
Filtering for Predefined Service Endpoints ... 795
Lookup Service Registration Filters ... 796
Filter Parameters for Predefined Service Endpoints ... 796
Connect to the Lookup Service and Retrieve the Service Registration Object ... 797
Retrieve Service Endpoints on vCenter Server Instances ... 797
Retrieve a vCenter Server ID by Using the Lookup Service ... 798

Retrieve a vSphere Automation API Endpoint on a vCenter Server Instance ... 798
Authentication Mechanisms ... 658
Choosing Your Identity Source ... 658
Default or External Identity Provider ... 658
Basic or Token-Based Authentication ... 658
Basic Authentication ... 658
Token-Based Authentication ... 659
Authentication Terminolgy ... 659
Create a vSphere Automation Session with User Credentials ... 801
Retrieve a SAML Token ... 802
Create a vSphere Automation Session with a SAML Token ... 802
Create a Web Services Session ... 803
Identity Provider Federation for vCenter Server ... 661
vCenter Server Identity Provider Federation Basics ... 661
Why Is Identity Provider Federation Useful ... 661
vCenter Server External Identity Provider Support ... 661
Federation to Microsoft Active Directory Federation Services (AD FS) ... 662
Federation to Okta and Azure AD Through VMware Identity Broker - vCenter Server ... 662
What is OAuth 2.0 ... 662
OAuth 2.0 Grant Types ... 662
What Is OpenID Connect (OIDC) ... 663
AD FS Federation Workflows ... 663
Federate vCenter Server to Microsoft Active Directory Federation Services (AD FS) ... 663
Authenticate to an AD FS-Federated vCenter Server by Using the Password Grant Type ... 664
Authenticate to an AD FS-Federated vCenter Server by Using the Authorization Code Grant Type ... 665
VMware Identity Broker - vCenter Server Workflows ... 666
What Is VMware Identity Broker - vCenter Server ... 666
Configure vCenter Server Identity Provider Federation to Okta or Azure AD ... 666
OAuth Authentication to Your Federated vCenter Server, App, or Script ... 667
Federate Apps or Scripts to an External Identity Provider by Using Your VMware Identity Broker - vCenter Server Configuration ... 667
Obtain JWT Tokens from VMware Identity Broker - vCenter Server by Using the Password Grant Type ... 668
Obtain JWT Tokens from VMware Identity Broker - vCenter Server by Using the Authorization Code Grant Type ... 670
Obtain JWT Token from VMware Identity Broker - vCenter Server by Using the Client Credentials Grant Type ... 671
Exchange JWT Tokens for a SAML Token and Obtain a Session Identifier ... 672
Accessing vSphere Automation Services ... 819
Access a vSphere Automation Service ... 820
Managing the Life Cycle of Hosts and Clusters ... 677
vSphere Lifecycle Manager Terms ... 678

vSphere Lifecycle Manager Terminology 678
vSphere Lifecycle Manager Overview 679
Depot Manager 680
Image Manager 680
Coordinator and ESXi Updater Managers 680
Options for Managing the ESXi Life Cycle 681
vSphere Lifecycle Manager Features 681
Enabling a Cluster to Use a Software Specification 689
Creating a Cluster with Enabled vSphere Lifecycle Manager 689
Enabling an Existing Cluster to Use vSphere Lifecycle Manager 690
Enabling a Standalone Host to Use a Software Specification 691
Hardware Compatibility Data 695
Checking the Hardware Compatibility of an ESXi Host 695
Configuring Remediation Settings 695
How Remediation Policies Overrides Work 695
Editing Global or Cluster- and Host-Specific Remediation Policies 696
Remediation Policy Options for Clusters 696
Remediation Policy Options for Standalone Hosts 698
Remediating an ESXi Cluster and a Standalone Host 699
Integrate Third-Party Solutions with vSphere Lifecycle Manager 699
Prerequisites 700
Integrate a Third-Party Solution to Work with vSphere Lifecycle Manager 700
Results 701
What to do next 701
Enable vSphere Lifecycle Manager on a Cluster Managed with Baselines 701
vSphere Lifecycle Manager APIs Equivalent to the vSphere Host Patch Manager APIs 702
Virtual Machine Configuration and Management 702
Creating Virtual Machines 835
Creating a Virtual Machine Without a Clone or Template 835
Configuring Virtual Machines 836
Name and Location 836
Hardware Version 836
Boot Options 836
Guest Operating System 837
CPU and Memory 837
Networks 837
Managing Virtual Machines 838
Filtering Virtual Machines 838
Installing VMware Tools 838
Performing Virtual Machine Power Operations 839

Registering and Unregistering Virtual Machines ... 839
Virtual Machine Guest Operations ... 840
Upload and Run a Script on a Guest Operating System ... 840
Managing Data Sets ... 707
Data Set Operations ... 707
Using Content Libraries ... 708
How to Use Content Libraries ... 849
Create a Local Content Library ... 849
Publish an Existing Content Library ... 851
Publish a Library at the Time of Creation ... 853
Subscribe to a Content Library ... 853
Synchronize a Subscribed Content Library ... 856
Editing the Settings of a Content Library ... 857
Removing the Content of a Subscribed Library ... 858
Delete a Content Library ... 858
How To Use Library Items ... 858
Create an Empty Library Item ... 858
Querying Library Items ... 860
Edit the Settings of a Library Item ... 860
Upload a File from a Local System to a Library Item ... 863
Upload a File from a URL to a Library Item ... 866
Download Files to a Local System from a Library Item ... 870
Synchronizing a Library Item in a Subscribed Content Library ... 874
Removing the Content of a Library Item ... 874
Deleting a Library Item ... 875
vSphere Tag Service ... 719
Creating vSphere Tags ... 719
Creating a Tag Category ... 719
Creating a Tag ... 719
Creating Tag Associations ... 721
Assign a Tag to a Content Library ... 877
Assign a Tag to a Cluster ... 878
Updating a Tag ... 721
Using Tags to Create and Manage Compute Policies ... 723
Create a Compute Policy ... 723
vSphere Supervisor Configuration and Management ... 737
vSphere Supervisor Terminology ... 737
vSphere Supervisor Basic Terms ... 737
vSphere Supervisor Components and Services ... 738
How to Configure and Manage a Supervisor ... 739

Enable vSphere Supervisor on a Cluster with NSX as the Networking Stack ... 885
Enable vSphere Supervisor on a Cluster with the vSphere Networking Stack ... 891
Upgrading a Supervisor ... 752
Monitoring the Enable and Upgrade Supervisor Operations ... 752
Reconfiguring a Supervisor ... 752
Deactivating a Supervisor ... 752
vSphere Security ... 723
Managing Certificates ... 723
Certificate Management Operations ... 896
Add a Root Certificate to vCenter Server ... 898
Delete a Root Certificate from vCenter Server ... 898
Change the Machine SSL Certificate of vCenter Server ... 899
Refresh the vCenter Server STS Signing Certificate with a VMCA-Issued Certificate ... 905
Set a Custom STS Signing Certificate to vCenter Server ... 906
Managing vSphere Native Key Provider ... 730
Retrieve vSphere Native Key Providers ... 730
Manage vSphere Native Key Providers ... 730
Export and Import vSphere Native Key Provider Configuration ... 731
Retrieve vSphere Native Key Providers on a Host ... 731
vSphere Trust Authority ... 731
Configure a vSphere Trust Authority Cluster ... 731
Configure Key Providers ... 731
Establish Trust Between Key Provider and Key Server ... 732
Configure Trusted TPMs of Attested ESXi Hosts on a Cluster Level ... 732
Configure Trusted ESXi Builds on a Cluster Level ... 733
Retrieve vSphere Trust Authority Components Information ... 733
Configure vSphere Trust Authority Components ... 733
Configure vSphere Trust Authority Components for Trusted Clusters ... 734
Establish Trust Between Hosts in a vSphere Trust Authority Cluster and a Workload vCenter Server ... 735
Check Trusted Cluster Health ... 735
Remediate a Trusted Cluster ... 736
Managing Host Hardware Security ... 736
Retrieve Host Hardware TPM Information ... 736
Manage Host Hardware TPM Endorsement Keys ... 736
Register an SGX Host ... 737
vCenter Server Management ... 761
Authorization Model for Administration of vCenter Server ... 761
Authorization Model Mapping to the vCenter Single Sign-On Domain ... 761
Using the Operator Role ... 761
Using the Admin Role ... 761

Using the SuperAdmin Role ... 762
Performing Privilege Checks Operations ... 762
Filtering Privilege Checks ... 762
Using Advanced Settings ... 763
vCenter Server Installation and Setup ... 763
Install Stage 2 ... 763
Setting Up a Newly Installed vCenter Server Instance ... 763
Workflows for Install Stage 2 ... 765
File-Based Backup and Restore of vCenter Server ... 767
Backing up vCenter Server ... 767
Backup and Restore Protocols for vCenter Server ... 768
Calculate the Size Needed To Store the Backup File ... 768
Back up a vCenter Server Instance by Using the API ... 769
Schedule a Backup Job ... 770
Restoring vCenter Server ... 770
Authentication When Restoring a vCenter Server Instance ... 771
Availability of Services While Restoring a vCenter Server Instance ... 771
Restore a vCenter Server Instance by Using the API ... 771
Reconcile a vCenter Server Instance with Nodes in Embedded Linked Mode ... 772
Troubleshooting for vCenter Server Installation or Deployment ... 772
Managing Support Bundles ... 772
vCenter Server Upgrade ... 774
Upgrade Stage 2 ... 775
Upgrading a vCenter Server Instance ... 775
Workflows for Upgrade Stage 2 ... 777
Historical Data Transfer ... 779
Deferred Import ... 780
Historical Data Import Errors ... 781
vCenter Server Configuration ... 782
Health Monitoring of vCenter Server ... 782
Check Overall System Health of vCenter Server ... 782
Capacity Monitoring ofvCenter Server ... 783
Check Database Usage in vCenter Server ... 785
List Storage Consumption By Data Type in vCenter Server ... 785
Managing the Global FIPS Compliance ... 785
Performing Infrastructure Profile Management Operations ... 786
Patching and Updating vCenter Server Deployments ... 786
Planning vCenter Server Updates ... 786
Performing Discovery and Planning Operations ... 787
List Available Products and Manage Associated Products ... 788

List Available Updates ... 788
Retrieve a Report ... 788
Updating vCenter Server ... 789
ApplyingvCenter Server Software Updates ... 789
vCenter Server Software Update Workflow ... 790
Developing Remote Plug-ins with the vSphere Client SDK ... 945
The vSphere Client SDK ... 945
Intended Audience ... 945
Revision History ... 945
Using the vSphere Client Remote Plug-in Samples ... 948
Try the Remote Plug-in Sample Starter ... 948
What To Do First ... 948
Generate a Plug-in Server Certificate ... 949
Build the Remote Plug-in Sample Starter ... 949
Start the Remote Plug-in Sample Starter ... 950
Save the Certificate or Fingerprint of the Plug-in Server ... 950
Register the Remote Plug-in Sample Starter ... 950
View the Plug-in User Interface ... 952
Remote Plug-in Sample UI and Server ... 952
Components in the Remote Plug-in Sample ... 952
Build the vSphere Client Remote Plug-in Sample ... 953
Find the SSL Thumbprint and GUID of vCenter Server ... 955
Start the Remote Plug-in Server ... 955
... 956
Get Thumbprint or Certificate of Remote Plug-in Server ... 956
Register the vSphere Client Remote Plug-in Sample ... 957
Viewing the vSphere Client Remote Plug-in Sample ... 959
Remote Plug-in Sample Directory Structure ... 972
Top-Level Directory ... 973
Remote Plug-in Architecture in the vSphere Client ... 974
Components of the vSphere Client Architecture ... 974
vCenter Server Configurations ... 974
Communication Paths in Remote Plug-in Architecture ... 975
Communications Among UI Components in the vSphere Client ... 976
Client-Server Communications with Remote Plug-ins ... 978
Security Concepts for Remote Plug-ins ... 978
Creating a Remote Plug-in for the vSphere Client ... 978
Code Components for a vSphere Client Remote Plug-in ... 978
Deployment Requirements for a vSphere Client Remote Plug-in ... 979
Using Auxiliary Plug-in Servers ... 980

vSphere Client Plug-in Registration Tool ... 980
Plug-in Registration Script ... 980
Plug-in Registration Script Syntax ... 981
Sample Manifest File for a Remote Plug-in ... 982
Deploying Remote Plug-ins for the vSphere Client ... 983
Remote Plug-in Life Cycle ... 983
Remote Plug-in Deployment ... 984
Plug-In Caching ... 984
Remote Plugin Uninstallation ... 985
Redeploying Plug-ins During Development ... 985
Specifying Remote Plug-in Compatibility ... 986
Remote Plug-in Topologies ... 987
Remote Plug-in Terminology ... 988
Visibility of Remote Plug-in Views ... 988
Remote Plug-in Multi-Instance Support ... 988
Differentiating Plug-in Instances ... 991
Custom Topologies ... 993
Deploying Auxiliary Plug-in Servers ... 993
Remote Plug-in Deployment Example with Simultaneous Users ... 994
Remote Plug-in Multi-Version Support ... 1003
Remote Plug-in Multi-Manifest Support ... 1004
Choosing Extension Points for vSphere Client Plug-ins ... 1005
Types of Extension Points in the vSphere Client ... 1005
Action and View Extensions ... 1005
Extension Types ... 1006
Remote Plug-in Manifest Example ... 1013
... 1013
Dynamic Extensions for Remote Plug-ins ... 1017
Dynamic Extension Use Cases ... 1017
How the vSphere Client Displays Dynamic Extensions ... 1017
Dynamic Views ... 1017
Dynamic Cards ... 1017
Dynamic Actions ... 1017
Direct Links to Dynamic Views ... 1018
Direct Links to Static Views ... 1018
Time-outs ... 1018
Caching Dynamic Extension Visibility ... 1018
Configure Dynamic Extensions ... 1018
plugin.json With Dynamic Extensions ... 1027
Dynamic Extensions Filter Query ... 1029

Example Code for Filtering Dynamic Extensions ... 1032
vSphere Client Plug-in User Interface Modules ... 1034
Bootstrapping the JavaScript API ... 1034
vSphere Client JavaScript API: htmlClientSdk Interface ... 1034
`htmlClientSdk.getProxiedPluginServerOrigin()` ... 1034
`htmlClientSdk.initialize()` ... 1035
`htmlClientSdk.isInitialized()` ... 1035
vSphere Client JavaScript API: Modal Interface ... 1035
`modal.AlertLevel` ... 1035
`modal.ButtonStyle` ... 1036
`modal.ButtonType` ... 1036
`modal.close()` ... 1036
`modal.ConfirmationModalConfig` ... 1036
`modal.DynamicModalConfig` ... 1037
`modal.getCustomData()` ... 1037
`modal.getSharedModalsMap()` ... 1038
`modal.ModalButton` ... 1038
`modal.ModalConfig` ... 1038
`modal.ModalSize` ... 1039
`modal.open()` ... 1039
`modal.openConfirmationModal()` ... 1039
`modal.openSharedModal()` ... 1040
`modal.SharedModalConfig` ... 1040
`modal.setOptions()` ... 1040
`modal.SharedModalsMap` ... 1040
`modal.VcSharedModalsInfo` ... 1041
vSphere Client JavaScript API: Application Interface ... 1041
`app.ApiEndpoints` ... 1041
`app.ClientInfo` ... 1041
`app.ClientViewNavigationOptions` ... 1042
`app.formatDateTime()` ... 1042
`app.getApiEndpoints()` ... 1042
`app.getClientInfo()` ... 1043
`app.getClientLocale()` ... 1043
`app.getContextObjects()` ... 1043
`app.getNavigationData()` ... 1043
`app.getPluginBackendInfo()` ... 1043
`app.getRemotePluginNavigationMap()` ... 1044
`app.getSessionInfo()` ... 1044

app.getTheme() ... 1044
app.navigateTo() ... 1044
app.navigateToClientView() ... 1045
app.navigateToRemotePluginView() ... 1045
app.NavigationOptions ... 1045
app.PluginBackendInfo ... 1045
app.PluginBackendServerInfo ... 1046
app.PluginDateTimeFormatOptions ... 1046
app.PluginTheme ... 1047
app.QueryParam ... 1047
app.refreshPluginItemsState() ... 1045
app.RemotePluginNavigationMap ... 1047
app.RemotePluginVcNavigationInfo ... 1048
app.RemotePluginViewNavigationOptions ... 1048
app.SessionInfo ... 1048
app.UiApiEndpoint ... 1049
vSphere Client JavaScript API: Event Interface ... 1049
event.onDateTimeFormatChanged() ... 1049
event.onGlobalRefresh() ... 1049
event.onThemeChanged() ... 1049
Example Using the modal API ... 1050
modal.html ... 1050
modal.js ... 1050
Using Themes with vSphere Client Plug-ins ... 1051
Using Style Variables in Plug-In CSS ... 1052
Building Output Style Sheets for vSphere Client Plug-Ins ... 1053
Configuring and Loading Theme Style Sheets in vSphere Client Remote Plug-Ins ... 1055
Configuring Theme-Dependent Icons for vSphere Client Remote Plug-ins ... 1058
Integrated Solution Installer for the vSphere Client ... 1060
Conceptual Overview of the Preparation Process ... 1060
Preparing a Solution OVF to Use the Integrated Solution Installer ... 1061
Deployment with the Integrated Solution Installer ... 1072
Deploy the OVF template in a production environment ... 1072
Injected OVF Properties ... 1074
Solution Install API ... 1075
Integrated Installer Solution Metadata ... 1077
vmw.vsphereui.solutioninstall.solutionId ... 1077
vmw.vsphereui.solutioninstall.pluginId ... 1078
vmw.vsphereui.solutioninstall.vCenterSupport ... 1079

vmw.vsphereui.solutioninstall.requiredVcVersion ... 1079
vmw.vsphereui.solutioninstall.legacyExtensionId ... 1080
vmw.vsphereui.solutioninstall.preventInstallationWhenLegacyExtension ... 1081
Plug-in Properties XML Example ... 1082
Advanced Considerations for Remote Plug-in Servers ... 1083
Registering Auxiliary Plug-in Servers ... 1083
Auxiliary Server Data Paths ... 1085
Communication Paths for Authentication in the Remote Plug-in Server ... 1086
How to Delegate Session Authority to the Plug-in Server ... 1087
Response Codes to session/clone-ticket Request ... 1092
Best Practices for vSphere Client Remote Plug-ins ... 1092
Best Practices for Implementing Plug-in Workflows ... 1092
Best Practices for Localization ... 1093
Best Practices for Deploying and Testing Remote Plug-ins ... 1093
Additional Resources ... 1093
... 1094
Troubleshooting Remote Plug–ins for the vSphere Client ... 1094
Plug-in Does Not Appear in vSphere Client ... 1094
Wrong Plug-in URL ... 1094
Troubleshooting: Plug-in Thumbprint Incorrect ... 1095
Manifest Cannot Be Parsed ... 1096
Wrong Plug-in Type ... 1096
Plug-in Marked As Incompatible ... 1097
Plug-in Registered with Incompatible Version ... 1097
Plug-in View is missing in the vSphere Client ... 1097
Missing Entry in the Instance Selector ... 1098
Unable to Change Plug-in Manifest File ... 1098
Troubleshooting: Problems with Registration Script in SDK ... 1098
Plug-in Already Registered ... 1098
Unable To Unregister Plug-in ... 1099
Developing Local Plug-ins with the vSphere Client SDK ... 1100
Intended Audience ... 1100
VMware Technical Publications Glossary ... 1100
About This Book ... 1100
Intended Audience ... 1100
VMware Technical Publications Glossary ... 1100
Deprecation of Local Plug-in Architecture ... 1101
Revision History ... 1101
About the vSphere Web Client and the vSphere Client ... 1102
Understanding the vSphere Client Architecture ... 1102

Overview of the User Interface Layer Components 1104
Understanding Extensibility in thevSphere Client 1104
Extending the User Interface Layer 1105
Extending the Java Service Layer 1105
About the vSphere Client 1106
Knowledge Requirements for Using the vSphere Client SDK 1106
SDK Versions and Compatibility 1106
vSphere Client SDK Contents 1107
..... 1107
vSphere Client SDK Setup 1108
Software Requirements 1108
Development Environment Requirements Overview 1108
Setting Up for HTML-Based Plug-In Development 1108
Set Up for Java Development 1109
Automate the Plug-In Build Process 1109
Download thevSphere Client SDK 1110
Set Up the Eclipse Integrated Development Environment (optional) 1110
Register Your Local vSphere Client with the vCenter Server Instance 1111
Using the vSphere Client SDK Samples 1113
Location of Sample Plug-in in the vSphere Client SDK 1113
..... 1113
vSphere HTML SDK Sample 1114
Build and Deploy the vSphere HTML SDK Sample Plug-in 1114
Running the vSphere HTML SDK Sample 1115
Creating a vSphere Client SDK Solution 1115
Before Creating an HTML Plug-In 1115
Create an HTML Plug-In Project 1115
Contents of the HTML Plug-In Project Template 1116
Building a Plug-In Package from the Project Template 1117
Plug-In Package Manifest File 1118
Testing the Generated Plug-Ins 1118
Deploy the Plug-In on a Local vSphere Client 1119
Deploying Your Plug-In on a Remote vSphere Client 1119
vCenter Server Plug-In Registration Tool 1119
Extension Points in the vSphere Client 1121
Global Extension Points 1121
Object Navigator Extension Points 1124
Object Workspace Extension Points 1125
Actions Extension Points 1131
Extension Templates 1134

Custom Object Extension Points 1136
Ordering Extensions 1142
Filtering Extensions 1143
Filtering Based on Selected Object Type 1143
Extension Filtered by Entity Type 1143
Filtering Based on the Value of a Property of the Selected Object 1143
Example Property Value Filter 1144
Filtering Based on User Privilege Level 1145
Limitations on Filtering Based on User Privilege Level 1146
Extension Filtered by User Privileges 1146
Using the vSphere Client JavaScript API 1146
vSphere Client JavaScript API: Modal Interface 1035
`modal.AlertLevel` 1035
`modal.ButtonStyle` 1036
`modal.ButtonType` 1036
`modal.close()` 1036
`modal.ConfirmationModalConfig` 1036
`modal.DynamicModalConfig` 1037
`modal.getCustomData()` 1037
`modal.getSharedModalsMap()` 1038
`modal.ModalButton` 1038
`modal.ModalConfig` 1038
`modal.ModalSize` 1039
`modal.open()` 1039
`modal.openConfirmationModal()` 1039
`modal.openSharedModal()` 1040
`modal.SharedModalConfig` 1040
`modal.setOptions()` 1040
`modal.SharedModalsMap` 1040
`modal.VcSharedModalsInfo` 1041
vSphere Client JavaScript API: Application Interface 1041
`app.ApiEndpoints` 1041
`app.ClientInfo` 1041
`app.ClientViewNavigationOptions` 1042
`app.formatDateTime()` 1042
`app.getApiEndpoints()` 1042
`app.getClientInfo()` 1043
`app.getClientLocale()` 1043
`app.getContextObjects()` 1043

app.getNavigationData() 1043
app.getPluginBackendInfo() 1043
app.getRemotePluginNavigationMap() 1044
app.getSessionInfo() 1044
app.getTheme() 1044
app.navigateTo() 1044
app.navigateToClientView() 1045
app.navigateToRemotePluginView() 1045
app.NavigationOptions 1045
app.PluginBackendInfo 1045
app.PluginBackendServerInfo 1046
app.PluginDateTimeFormatOptions 1046
app.PluginTheme 1047
app.QueryParam 1047
app.refreshPluginItemsState() 1045
app.RemotePluginNavigationMap 1047
app.RemotePluginVcNavigationInfo 1048
app.RemotePluginViewNavigationOptions 1048
app.SessionInfo 1048
app.UiApiEndpoint 1049
vSphere Client JavaScript API: Event Interface 1049
event.onDateTimeFormatChanged() 1049
event.onGlobalRefresh() 1049
event.onThemeChanged() 1049
Example Using the modal API 1050
modal.html 1050
modal.js 1050
Using Themes with vSphere Client Plug-ins 1051
Using Style Variables in Plug-In CSS 1052
Building Output Style Sheets for vSphere Client Plug-Ins 1164
Configuring and Loading Theme Style Sheets in vSphere Client Local Plug-Ins 1167
Configuring Theme-Dependent Icons for vSphere Client Local Plug-ins 1169
CSS For dark Theme Icons 1169
Developing HTML-Based User Interface Extensions 1170
Overview 1170
Accessing Data 1170
Global View Extensions 1171
Use Cases 1171
Creating Global View Extensions 1171

Extending the vCenter Object Workspace....1171
Use Cases....1171
Extending an Existing Object Workspace....1172
Adding a Host **Monitor** View....1172
Creating an Object Workspace for a Custom Object....1173
Creating Extensions to the **Summary** Tab....1173
Adding Portlet Views to the **Summary** Tab....1173
Creating Data View Extensions....1174
Common Data Access Pattern....1174
Creating Actions Extensions....1175
Use Cases....1176
Actions Framework Overview....1176
Defining an Action Set....1176
Defining Individual Actions for HTML-Based Action Extensions....1176
HTML-Based Headless Action Extension Definition....1177
HTML-Based UI Action Extension Definition....1179
Handling Actions for HTML-Based Action Extensions....1180
Example Java Actions Controller Class....1180
Handling Locales....1180
Handling Resources in the `plugin.xml` Manifest File....1180
Localizing Strings and Icons in the `plugin.xml` Manifest File....1181
Handling Resources in the HTML and JavaScript Code....1182
Handling Resources at the Service Layer....1182
Guidelines for Creating Plug-Ins Compatible with the vSphere Client....1182
Using the Web Context Path in HTML Plug-Ins....1183
Handling Icons Outside the HTML Views....1183
Defining Menus and Sub-Menus....1183
Developing for the vSphere Client Service Layer....1185
Developing Extensions to the Service Layer....1185
Understanding the vSphere Client Data Service....1185
Overview of Data Service Queries....1188
Extending the Data Service with a Data Service Adapter....1191
Creating a Custom Java Service....1201
Importing a Service in a User Interface Plug-In Module....1203
Creating and Deploying Plug-In Packages....1203
Plug-In Package Overview....1203
XML Elements of the Plug-In Package Manifest File....1204
XML Elements in the Manifest File....1204
`<pluginPackage>` Element....1205
`<dependencies>` Element....1205

<bundlesOrder> Element ... 1206
Deploying a Plug-In Package ... 1206
Plug-In Caching ... 984
Deploying a Plug-In Package From a Remote Server ... 1207
Register a Plug-In Package as a vCenter Server Extension ... 1207
Creating the vCenter ServerExtension Data Object ... 1208
Example vim.Extension XML Definition ... 1209
Verifying Your Plug-In Package Deployment ... 1211
Redeploying Plug-ins During Development ... 1211
Unregister a Plug-In Package ... 1211
Best Practices for Developing Extensions for the vSphere Client ... 1212
Best Practices for Creating Plug-In Packages ... 1212
Best Practices for Plug-In Modules Implementation ... 1214
Best Practices for Developing HTML-Based Extensions ... 1214
Best Practices for Extending the User Interface Layer ... 1215
Best Practices for Extending the Service Layer ... 1215
OSGi-Specific Recommendations ... 1216
DataService-Specific Best Practices ... 1218
Best Practices for SSL Connections ... 1219
Best Practices for Using VimPortType with JAX-WS ... 1220
Best Practices for Deploying and Testing Your vSphere Client Extensions ... 1220
VMware Guest SDK Programming Guide ... 1222
Revision History ... 1222
Intended Audience ... 1222
Installing the Development Kit ... 1222
What the Guest SDK Contains ... 1222
Displaying vSphere Guest Library Statistics ... 1223
Security of Remote RPC ... 1223
Security of the DataSets API ... 1224
Accessing HA Application Monitoring ... 1224
Controlling the Application Monitoring Heartbeat ... 1224
Library Path or Path Environment ... 1224
Compiling the Sample Program on Linux ... 1224
Compiling Sample Programs on Windows ... 1224
Demonstrating the HA Application Monitoring API ... 1225
The VM Guest Library ... 1225
About the VM Guest Library ... 1225
Supported Guest Operating Systems ... 1225
Prerequisite LibC Version ... 1225
Virtual Machine Statistics ... 1226

How to Use the VM Guest API ... 1226
VM Guest API Runtime Components ... 1226
VM Guest API Data Types ... 1227
VM Guest Library Functions ... 1228
VM Guest API Error Codes ... 1232
Tools for Extended Guest Statistics ... 1233
Guest OS and Statistics Fetch ... 1233
Prerequisites ... 1233
Guest Statistics Interfaces ... 1234
Guest SDK Library ... 1234
C code with StatGet and StatFree functions ... 1234
Command Line Interface ... 1235
Raw RPC interface ... 1235
Fetching a List of Statistics ... 1236
Return list of statistics ... 1236
Get I/O statistics for a device ... 1236
Metadata Fields ... 1236
Text ... 1237
YAML ... 1237
XML (formatted here for presentation) ... 1238
JSON (formatted here for presentation) ... 1238
Metrics Examples ... 1238
Fetch Available Statistics ... 1239
Get Session Information ... 1239
Host Hardware ... 1240
CPU and Memory Statistics ... 1240
Storage Statistics ... 1245
Network Statistics ... 1245
DataSets for Guest Management ... 1246
About the DataSets Service ... 1246
Use Cases for DataSets ... 1247
Control Through vCenter Server ... 1247
DataSets Transfer to ESXi Host ... 1248
Access from Guest OS ... 1249
JSON in a File ... 1253
Python for DataSets ... 1253
DataSets Schema for Guest Access ... 1253
Table of Error Codes ... 1266
In-Guest Communication Limits ... 1266
vSphere HA Application Monitoring ... 1266

About vSphere HA ... 1267
Prerequisites for HA Application Monitoring ... 1267
Using the HA Application Monitoring APIs ... 1268
HA Application Monitoring API Functions ... 1268
Code Sample for appmon.cpp ... 1270
Calling the APIs from Your Application ... 1270
HA Application Monitoring API Error Messages ... 1271
Virtual Disk Development Kit (VDDK) Programming Guide ... 1273
Revision History ... 1273
Intended Audience ... 1274
Supported Platform Products ... 1274
VMware Technical Publications ... 1274
The Virtual Disk API and vSphere ... 1274
Virtual Disk Library VixDiskLib ... 1274
VDDK Components ... 1274
Virtual Disk Library ... 1275
Disk Mount Library ... 1275
Virtual Disk Utilities ... 1275
Backup and Restore on vSphere ... 1275
Backup Design for vCloud Director ... 1275
Use Cases for the Virtual Disk Library ... 1275
Developing for VMware Platform Products ... 1276
Managed Disk and Hosted Disk ... 1276
Advanced Transports ... 1277
VDDK and VADP Compared ... 1277
Platform Product Compatibility ... 1277
Redistributing VDDK Components ... 1277
Installing the Development Kit ... 1278
Prerequisites for VDDK and VADP ... 1278
Programming Environments ... 1278
Visual Studio on Windows ... 1278
C++ and C on Linux ... 1278
Java Development for VADP ... 1278
VMware Platform Products ... 1278
Storage Device Support ... 1278
Installing the VDDK Package ... 1279
Repackaging VDDK Libraries ... 1280
How to Find VADP Components ... 1280
Virtual Disk Interfaces ... 1280
VMDK File Location ... 1280

Virtual Disk Types ... 1280
Persistence Disk Modes ... 1281
VMDK File Naming ... 1281
Thin Provisioned Disk ... 1282
Internationalization and Localization ... 1282
Grain Directories and Grain Tables ... 1283
Data Structures in Virtual Disk API ... 1283
Datastore-Level Access to VMDK ... 1284
Credentials and Privileges for VMDK Access ... 1285
Adapter Types ... 1285
Virtual Disk Transport Methods ... 1285
Local File Access ... 1285
SAN Transport ... 1286
HotAdd Transport ... 1287
NBDSSL Transport ... 1288
Virtual Disk API Functions ... 1291
Virtual Disk Library Functions ... 1292
Alphabetic Table of Functions ... 1292
Start Up ... 1293
Initialize the Library ... 1293
Connect to a Workstation or Server ... 1294
VMX Specification ... 1294
Disk Operations ... 1295
Create a New Hosted Disk ... 1295
Open a Local or Remote Disk ... 1295
Read Sectors From a Disk ... 1295
Write Sectors To a Disk ... 1296
Close a Local or Remote Disk ... 1296
Get Information About a Disk ... 1296
Free Memory from Get Information ... 1297
Metadata Handling ... 1297
Read Metadata Key from Disk ... 1297
Get Metadata Table from Disk ... 1297
Write Metadata Table to Disk ... 1297
Check and Repair Sparse Disk Metadata ... 1297
Disk Chaining and Redo Logs ... 1297
Create Child from Parent Disk ... 1298
Attach Child to Parent Disk ... 1298
Opening in a Chain ... 1300
Redo Logs and Linked Clone Backup ... 1300

Cloning a Virtual Disk ... 1300
Compute Space Needed for Clone ... 1300
Clone a Disk by Copying Data ... 1301
Error Handling ... 1301
Return Error Description Text ... 1301
Free Error Description Text ... 1301
Administrative Disk Operations ... 1301
Rename an Existing Disk ... 1301
Grow an Existing Local Disk ... 1301
Defragment an Existing Disk ... 1302
Shrink an Existing Local Disk ... 1302
Unlink Extents to Remove Disk ... 1302
Shut Down ... 1302
Disconnect from Server ... 1302
Clean Up and Exit ... 1302
Advanced Transport APIs ... 1302
Initialize Virtual Disk API ... 1303
Wildcards in SAN Allow and Deny Lists ... 1305
How to Enable FIPS on VDDK Proxy VM ... 1305
Phone Home Support ... 1306
Location of Log Files ... 1307
List Available Transport Methods ... 1307
Connect to VMware vSphere ... 1307
Get Selected Transport Method ... 1308
Prepare For Access and End Access ... 1308
SAN Mode on Linux Uses Direct Mode ... 1309
Clean Up After Disconnect ... 1309
Ordering of Function Calls in Sequence ... 1310
Updating Applications for Advanced Transport ... 1310
Algorithm for vSphere Backup ... 1310
Backup and Recovery Example ... 1311
Multithreading Considerations ... 1312
Multiple Threads and VixDiskLib ... 1312
Capabilities of Library Calls ... 1312
Support for Managed Disk ... 1313
Support for Hosted Disk ... 1313
Virtual Disk API Sample Code ... 1313
Compiling the Sample Program ... 1313
Visual C++ on Windows ... 1313
C++ on Linux Systems ... 1314

Library Files Required 1314
Usage Message 1314
Walk-Through of Sample Program 1316
Include Files 1316
Definitions and Structures 1316
Dynamic Loading 1316
Wrapper Classes 1317
Command Functions 1317
SSL Certificate Thumbprint 1320
Practical Programming Tasks 1320
Scan VMDK for Virus Signatures 1320
Creating Virtual Disks 1322
Create Local Disk 1322
Create Remote Disk 1322
Special Consideration for ESXi Hosts 1323
VMDK File Versions 1323
Working with Virtual Disk Data 1324
Reading and Writing Local Disk 1324
Reading and Writing Remote Disk 1324
Deleting a Disk (Unlink) 1325
Renaming a Disk 1325
Managing Child Disks 1325
Create Redo Logs 1325
Virtual Disk in Snapshots 1326
RDM Disks and Virtual BIOS 1326
Restore RDM Disks 1326
Restore the Virtual BIOS or UEFI 1327
Interfacing With VMware vSphere 1327
The VIX API 1327
Virus Scan all Hosted Disk 1327
The vSphere Web Services API 1328
Virus Scan All Managed Disk 1328
Read and Write VMDK Using vSphere API 1328
First Class Disk (FCD) Backup 1328
FCD Workflows and Operations 1329
Discovering FCDs for Backup 1329
New Connect Parameters and Functions 1329
FCD Managed Objects and Tasks 1331
Behavior of FCD With Changed Block Tracking 1331
Behavior of FCD with vMotion 1331

Code Sample for Connect Parameters ... 1331
Container Storage Snapshot ... 1332
VADP and VM Service Managed VMs ... 1334
Datasets Backup and Restore ... 1336
Backing Up Virtual Disks in vSphere ... 1337
Design and Implementation Overview ... 1337
The VADP Backup Process ... 1337
Communicating With the Server ... 1337
Information Containers as Managed Objects ... 1338
Gathering Status and Configuration Information ... 1339
Doing a Backup Operation ... 1340
Restore a Virtual Machine ... 1342
Doing a Restore Operation ... 1342
Access Files on Virtual Disks ... 1343
More VADP Details ... 1344
Low Level Backup Procedures ... 1344
Communicate with the Server ... 1344
The PropertyCollector ... 1345
Creating a Snapshot ... 1351
Backing Up a Virtual Disk ... 1352
Deleting a Snapshot ... 1353
Query Allocated Blocks Function ... 1353
Changed Block Tracking on Virtual Disks ... 1355
Low Level Restore Procedures ... 1359
Restoring a Virtual Machine and Disk ... 1359
Restore Incremental Backup Data ... 1373
Restore with Direct Connection to ESXi Host ... 1374
Tips and Best Practices ... 1374
Best Practices for HotAdd Transport ... 1374
Best Practices for SAN Transport ... 1375
Best Practices for NBD Transport ... 1376
General Backup and Restore ... 1378
Best Practice for Restore with Pre-existing Snapshot ... 1378
Backup and Restore of Thin-Provisioned Disk ... 1378
About Changed Block Tracking ... 1379
Configure SCSI Controllers in Proxy VM ... 1379
Encrypted VM Backup and Restore ... 1380
Backup and Restore With vTPM ... 1382
Windows Backup Implementations ... 1384
Deactivate Automount in Windows Proxy ... 1384

Security and Remote Desktop ... 1384
Working with Microsoft Shadow Copy ... 1385
Enable Virtual Machine Application Consistent Quiescing ... 1386
Application-Consistent Backup and Restore ... 1386
New VSS Support Added in vSphere 6.5 ... 1387
The VMware VSS Implementation ... 1387
Linux Backup Implementation ... 1388
Backing Up vApps in vCloud Director ... 1388
Tenant vApps in VMware Cloud ... 1389
Prerequisites ... 1389
Other Information ... 1390
Conceptual Overview ... 1390
The Backup Process ... 1390
The Restore Process ... 1391
Use Cases Overview ... 1392
Managing Credentials ... 1392
Finding a vApp ... 1392
Protecting Specified vApps ... 1392
Recovering an Older Version of a vApp ... 1393
Recovering a Deleted vApp ... 1393
Recovering a Single Virtual Machine ... 1393
Backing Up vCloud Director ... 1393
vCloud API Operations ... 1393
Getting Access to vCloud Director ... 1394
Inventory Access ... 1394
Retrieving Catalog information ... 1399
Retrieving vApp Configuration ... 1401
Preventing Updates to a vApp During Backup or Restore ... 1403
Associating vCloud Resources with vSphere Entities ... 1404
Restoring vApps ... 1407
Conclusion for vApp Backup ... 1409
Virtual Disk Mount API ... 1409
The VixMntapi Library ... 1409
Types and Structures ... 1409
Function Calls ... 1410
Programming with VixMntapi ... 1415
File System Support ... 1415
Diagnostic Logging for VixMntapi ... 1415
Read-Only Mount on Linux ... 1416
Sample VixMntapi Code ... 1416

Restrictions on Virtual Disk Mount ... 1417
Error Codes and Open Source ... 1417
Recent Changes ... 1417
Finding Error Code Documentation ... 1417
Association With VIX API Errors ... 1418
Interpreting Error Codes ... 1418
Troubleshooting Dynamic Libraries ... 1418
Troubleshooting Changed Block Tracking (CBT) ... 1418
Verify Change Tracking Enabled ... 1418
Run the vixDiskCheck Utility ... 1419
Open Source Components ... 1420
HTML Console SDK Programming Guide ... 1421
Revision History ... 1421
Intended Audience ... 1421
Related Documentation ... 1421
The VMware HTML Console ... 1421
Architecture of the HTML Console ... 1421
Remote HTML Console Features ... 1423
Compared to Other VMware Consoles ... 1423
Compared to the VMRC SDK Plug-in ... 1423
Compared to vSphere Web Client Consoles ... 1423
Getting Started with the HTML Console SDK ... 1423
Download and Install ... 1424
Steps to Start a Console ... 1424
Sample Javascript Code ... 1425
Acquire MKS Ticket with a Browser ... 1426
Acquire MKS Ticket with vSphere API ... 1426
Run the HTML Console Sample Code ... 1427
The Create WebMKS Method and Options ... 1427
createMKS Parameters ... 1427
createMKS Options ... 1427
keyBoardLayoutId Details ... 1429
Handling Events with the HTML Console ... 1430
Event Binding in jQuery ... 1430
Setting the Event Handler ... 1431
Event Handlers ... 1431
Copy from Remote Desktop ... 1432
HTML Console SDK Methods ... 1434
General API Methods ... 1434
getVersion() ... 1434

getConnectionState() ... 1434
Lifecycle API Methods ... 1434
connect() ... 1434
disconnect() ... 1435
destroy() ... 1435
Display API Methods ... 1435
setRemoteScreenSize() ... 1435
getRemoteScreenSize() ... 1436
updateScreen() ... 1436
Full Screen API Methods ... 1436
canFullScreen() ... 1436
isFullScreen() ... 1436
enterFullScreen() ... 1437
exitFullScreen() ... 1437
Input API Methods ... 1437
sendInputString() ... 1437
sendKeyCodes() ... 1437
sendCAD() ... 1438
Mobile API Methods ... 1438
enableInputDevice() ... 1438
disableInputDevice() ... 1438
showKeyboard() ... 1438
hideKeyboard() ... 1439
toggleExtendedKeypad() ... 1439
toggleTrackpad() ... 1439
toggleRelativePad() ... 1439
setOption API Method ... 1440
Event Handler API Methods ... 1440
register() ... 1440
unregister() ... 1441
WMKS Constants ... 1441
Javascript Key Codes ... 1443
Alphanumeric Keys ... 1443
Table of Key Codes ... 1443
Getting Started with ESXCLI ... 1444
Intended Audience ... 1444
Overview of ESXCLI ... 1444
Using ESXCLI for Host Management ... 1444
ESXCLI Syntax ... 1444
Running ESXCLI Commands ... 1445

ESXCLI Command Support When Host and ESXCLI Versions Do Not Match ... 1446
Installing ESXCLI ... 1446
Installing and Uninstalling ESXCLI ... 1446
Install ESXCLI ... 1446
Uninstall ESXCLI ... 1446
Running Host Management Commands in the ESXi Shell ... 1447
ESXi Shell Access with the Direct Console ... 1447
Enabling Local ESXi Shell Access ... 1447
ESXi Shell Timeout ... 1448
Use the Local ESXi Shell ... 1448
Remote ESXi Shell Access with SSH ... 1448
Enable SSH Access in the Direct Console ... 1448
Enable SSH from the vSphere Client ... 1449
Access the Remote ESXi Shell with SSH ... 1449
Lockdown Mode ... 1449
Run an ESXCLI Command in the ESXi Shell ... 1449
Running ESXCLI Host Management Commands ... 1450
Overview of Running ESXCLI Host Management Commands ... 1450
Targeting the Host Directly ... 1450
Targeting a Host That is Managed by a vCenter Server System ... 1450
Protecting Passwords ... 1451
Order of Precedence for ESXCLI Host Management Commands ... 1451
Authenticating Through vCenter Server and vCenter Single Sign-On ... 1452
Example ... 1452
Authenticating Directly to the Host ... 1452
Using Environment Variables ... 1452
Using a Configuration File ... 1453
Using Command-Line Options ... 1453
Linux ... 1454
Windows ... 1454
Using the Microsoft Windows Security Support Provider Interface ... 1454
ESXCLI and Lockdown Mode ... 1455
Trust Relationship Requirement for ESXCLI Commands ... 1455
Download and Install the vCenter Server Certificate ... 1455
Using the --cacertsfile Option ... 1455
Using the --thumbprint Option ... 1456
Connection Options for ESXCLI Host Management Command Execution ... 1456
Using ESXCLI Commands in Scripts ... 1459
Run Host Management Commands from a Windows System ... 1460
Run Host Management Commands from a Linux System ... 1460

ESXCLI Concepts and Examples 1461
Intended Audience 1461
ESXCLI Commands Overview 1461
Introduction to ESXCLI 1461
Targets and Protocols for ESXCLI Host Management Commands 1461
ESXCLI Commands Available on Different ESXi Hosts 1461
Trust Relationship Requirement for ESXCLI Commands 1455
Download and Install the vCenter Server Certificate 1455
Using the --cacertsfile Option 1455
Using the --thumbprint Option 1456
Using ESXCLI Output 1462
Connection Options for ESXCLI Host Management Commands 1463
ESXCLI Host Management Commands and Lockdown Mode 1463
Managing Hosts 1463
Stopping and Rebooting Hosts with ESXCLI 1464
Enter and Exit Maintenance Mode with ESXCLI 1464
Manage Modules with ESXCLI 1464
Retrieve Information about Components, Base Image, and Add-On on a Host with ESXCLI 1465
Manage Components with ESXCLI 1466
Manage Base Images and Add-Ons with ESXCLI 1466
Updating Hosts 1468
Configuring the GuestStore Repository with ESXCLI 1469
Set the GuestStore Repository with ESXCLI 1469
Clear the GuestStore Repository Setting with ESXCLI 1469
Managing Files 1469
Introduction to Virtual Machine File Management 1470
Managing VMFS Volumes 1471
Managing Duplicate VMFS Datastores 1471
Mounting Datastores with Existing Signatures 1471
Resignaturing VMFS Copies 1472
Reclaiming Unused Storage Space 1473
Managing Storage 1473
Introduction to Storage 1473
How Virtual Machines Access Storage 1474
Datastores 1475
Storage Device Naming 1476
Examining LUNs 1477
Target and Device Representation 1477
Examining LUNs with ESXCLI 1477
Detach a Device and Remove a LUN 1478

Reattach a Device ... 1479
Working with Permanent Device Loss ... 1479
Removing a PDL LUN ... 1480
Reattach a PDL LUN ... 1480
Managing Paths ... 1480
Multipathing with Local Storage and FC SANs ... 1480
Listing Path Information with ESXCLI ... 1481
Disable a Path with ESXCLI ... 1482
Managing Path Policies ... 1482
Supported Path Policies ... 1482
Path Policy Effects ... 1483
Multipathing Considerations ... 1484
Change the Path Policy with ESXCLI ... 1484
Set Policy Details for Devices that Use Round Robin ... 1485
Scheduling Queues for Virtual Machine I/O ... 1485
Managing NFS/NAS Datastores ... 1486
Capabilities Supported by NFS/NAS ... 1486
Manage a NAS File System with ESXCLI ... 1486
Monitor and Manage FibreChannel SAN Storage ... 1487
Monitoring and Managing vSAN Storage ... 1487
Retrieve vSAN Information ... 1488
Manage a vSAN Cluster ... 1488
Add and Remove vSAN Storage ... 1488
Monitoring vSphere Flash Read Cache ... 1489
Monitoring and Managing Virtual Volumes ... 1490
Configuring FCoE Adapters ... 1491
Scanning Storage Adapters ... 1491
Retrieving SMART Information ... 1491
Managing iSCSI Storage ... 1492
iSCSI Storage Overview ... 1492
Discovery Sessions ... 1493
Discovery Target Names ... 1493
Protecting an iSCSI SAN ... 1494
Protecting Transmitted Data ... 1494
Securing iSCSI Ports ... 1494
Setting iSCSI CHAP ... 1495
Command Syntax for esxcli iscsi ... 1496
esxcli iscsi Command Syntax ... 1496
Key to esxcli iscsi Short Options ... 1497
iSCSI Storage Setup with ESXCLI ... 1498

Set Up Software iSCSI with ESXCLI ... 1498
Set Up Dependent Hardware iSCSI with ESXCLI ... 1500
Set Up Independent Hardware iSCSI with ESXCLI ... 1503
Listing and Setting iSCSI Options ... 1505
Listing iSCSI Options with ESXCLI ... 1505
Setting MTU with ESXCLI ... 1505
Listing and Setting iSCSI Parameters ... 1506
Listing and Setting iSCSI Parameters with ESXCLI ... 1506
Returning Parameters to Default Inheritance with ESXCLI ... 1507
Enabling iSCSI Authentication ... 1508
Enable iSCSI Authentication with ESXCLI ... 1508
Enable Mutual iSCSI Authentication with ESXCLI ... 1509
Set Up Ports for iSCSI Multipathing ... 1509
Managing iSCSI Sessions ... 1510
Introduction to iSCSI Session Management ... 1510
Listing iSCSI Sessions ... 1511
Logging in to iSCSI Sessions ... 1511
Removing iSCSI Sessions ... 1511
Managing Third-Party Storage Arrays ... 1512
Managing NMP with esxcli storage nmp ... 1512
Device Management with esxcli storage nmp device ... 1512
Listing Paths with esxcli storage nmp path ... 1513
Managing Path Selection Policy Plug-Ins with esxcli storage nmp psp ... 1514
Fixed Path Selection Policy Operations ... 1515
Customizing Round Robin Setup ... 1516
Managing SATPs ... 1517
Path Claiming with esxcli storage core claiming ... 1520
Using the Reclaim Troubleshooting Command ... 1520
Unclaiming Paths or Sets of Paths ... 1521
Managing Claim Rules ... 1522
Change the Current Claim Rules in the VMkernel ... 1522
Adding Claim Rules ... 1523
Removing Claim Rules ... 1525
Listing Claim Rules ... 1526
Loading Claim Rules ... 1526
Moving Claim Rules ... 1526
Load and Apply Path Claim Rules ... 1526
Running Path Claim Rules ... 1527
Managing Users ... 1528
Users in the vSphere Environment ... 1528

Assigning Permissions with ESXCLI ... 1528
Manage Permissions ... 1528
Account Management ... 1529
Managing Virtual Machines ... 1529
Forcibly Stop a Virtual Machine with ESXCLI ... 1529
Managing vSphere Networking ... 1530
Introduction to vSphere Networking ... 1530
Networking Using vSphere Standard Switches ... 1531
Networking Using vSphere Distributed Switches ... 1531
Retrieving Basic Networking Information ... 1532
Troubleshoot a Networking Setup ... 1533
Setting Up vSphere Networking with vSphere Standard Switches ... 1534
Setting Up Virtual Switches and Associating a Switch with a Network Interface ... 1535
Retrieving Information About Virtual Switches with ESXCLI ... 1535
Adding and Deleting Virtual Switches ... 1536
Managing Port Groups with ESXCLI ... 1536
Connecting and Disconnecting Uplink Adapters and Port Groups with ESXCLI ... 1537
Setting the Port Group VLAN ID with ESXCLI ... 1537
Managing Uplink Adapters ... 1538
Adding and Modifying VMkernel Network Interfaces ... 1540
Managing Standard Networking Services in the vSphere Environment ... 1542
Setting the DNS Configuration with ESXCLI ... 1542
Set Up a DNS Server with ESXCLI ... 1542
Modify DNS Setup for a Preconfigured Server with ESXCLI ... 1543
Setting Up IPsec ... 1544
Using IPsec with ESXi ... 1544
Managing Security Associations ... 1545
Managing Security Policies ... 1546
Manage the ESXi Firewall ... 1548
Monitor VXLAN ... 1549
Monitoring ESXi Hosts ... 1550
Managing Diagnostic Partitions ... 1550
Diagnostic Partition Creation ... 1550
Diagnostic Partition Management ... 1551
Managing Core Dumps ... 1551
Manage Local Core Dumps with ESXCLI ... 1551
Manage Core Dumps with ESXi Dump Collector ... 1551
Configuring ESXi Syslog Services ... 1552
esxcli system syslog Usage ... 1553
Managing ESXi SNMP Agents ... 1554

Configuring SNMP Communities ... 1554
Configuring the SNMP Agent to Send Traps ... 1554
Retrieving Hardware Information ... 1555
Managing Security ... 1555
Configuring and Managing the Audit System and Audit Data ... 1556
Enable Local Auditing with ESXCLI ... 1556
Enable the Transmission of Audit Records to a Remote Host with ESXCLI ... 1556
Manage CA Certificates with ESXCLI ... 1557
Configuring the Cryptographic Functionality ... 1558
Configuring Virtual Machine Access to Physical Devices ... 1558
Managing Configuration Settings ... 1559
Sticky Bit Configuration Files ... 1559
Namespaces from esx.conf ... 1561
OVF Tool User Guide ... 1575
Revision History ... 1575
Intended Audience ... 1575
Document Feedback ... 1575
VMware OVF Tool ... 1575
The Open Virtualization Format ... 1576
The OVF Standard ... 1576
Benefits of OVF ... 1576
Using the VMware OVF Tool ... 1576
VMware Platforms Using the OVF Standard ... 1576
Setting Up the OVF Tool ... 1576
OVF Tool Highlights ... 1577
Examples of OVF Tool Use ... 1577
Importing and Exporting OVF Using vSphere Client ... 1577
VMware OVF Tool Delta Disk Facilities ... 1578
Using OVF Tool Commands ... 1578
Definitions and Command Syntax ... 1578
Run VMware OVF Tool From the Command Line ... 1579
Run OVF Tool With ESXi and vCenter Target Locations ... 1579
Deploy a Virtual Machine to Static DVS Port Group ... 1581
Command-Line Options ... 1581
Creating and Using the VM ID ... 1589
Specifying Disk ID to Set Size ... 1590
Specifying the Storage Profile ID ... 1591
More Help Topics ... 1591
Using the Log Settings ... 1591
Specifying a Locator ... 1592

File Locators ... 1593
URL File or Web Syntax ... 1593
vSphere Locators ... 1594
vCloud Director Locators ... 1596
Partial Locators ... 1596
Partial vSphere Locators at the Command Line ... 1597
Partial vCloud DirectorLocators at the Command Line ... 1598
Configuration Files ... 1599
Handling Authentication ... 1600
Launching the OVF Tool as a Helper Process ... 1600
Examples of OVF Tool Syntax ... 1601
Supported File Types and Package Formats ... 1601
Changing File or Package Formats ... 1602
Converting an OVF Package to an OVA Archive ... 1602
Converting an OVA Archive to OVF Package ... 1602
Converting VMX Format to an OVF Package ... 1602
Converting VMX Format to an OVA Archive ... 1602
Converting an OVA Archive to VMX Format ... 1603
Converting an OVF Package to VMX Format ... 1603
Installing an ESXi host from an OVF Package ... 1603
Installing an ESXi host from an OVF Package on a Web Server ... 1603
Installing an ESXi host or Adding Files from a VMX Format ... 1603
Installing a vCenter Server or Adding Files from an OVF Package ... 1603
Converting a VM on ESXi or vCenter Server to an OVF Package ... 1604
Installing vCenter Server from an OVF Package Using an Inventory Path ... 1604
Setting OVF Package Properties ... 1604
Setting OVF Properties When Deploying to vSphere or vCloud Director ... 1604
Setting OVF Network Maps When Deploying to vSphere ... 1604
Setting a vService Dependency ... 1604
Modifying an OVF Package ... 1605
Renaming the OVF Package ... 1605
Omitting Disks in the VMware OVF Tool Output ... 1605
Compressing an OVF Package ... 1605
Chunking or Splitting OVF Packages ... 1605
TPM as a Virtual Device in OVF ... 1605
Deploying OVF Packages ... 1607
Deploying an OVF Package Directly on an ESXi Host ... 1607
Deploying an OVF Package using Pull Mode ... 1607
Deploying an OVF Package through vCenter Server ... 1608
Deploying an OVF Package and Powering It On ... 1608

Deploying an OVF Package into vCloud Director ... 1608
Deploying an OVF Package into a vApprun Workspace ... 1608
Importing an OVF Package ... 1608
Importing an OVF File into a vCloud instance ... 1608
Importing a Virtual Machine from vSphere to vCloud ... 1609
Importing VMX Files into a vApprun Workspace ... 1609
Exporting Virtual Machines to OVF Packages ... 1609
Exporting a Virtual Machine from a vCloud instance to an OVF Package ... 1609
Exporting a Running Virtual Machine or vApp from vSphere ... 1609
Exporting a vApprun Entity to an OVF Package ... 1610
Displaying Summary Information ... 1610
Validating an OVF 1.0 or OVF 1.1 Descriptor ... 1610
Downloading an OVF Package from a Protected Web Site ... 1610
Using a Proxy ... 1610
Overwriting a Running Virtual Machine or vApp from vSphere ... 1611
Canceling the VMware OVF Tool While it Is Running ... 1611
OVF Package Signing ... 1611
Publishing an OVF Package with a Digital Signature ... 1611
Creating an RSA Public/Private Key Pair and Certificate ... 1612
myself.pem File ... 1612
Signing an OVF Package ... 1613
Certificate File Created by OVF Tool ... 1613
Validating an OVF Package ... 1614
Using OVF Tool Probe Mode ... 1614
About OVF Tool Probe Mode ... 1614
Example of Probe Mode ... 1615
Using OVF Tool Machine Mode ... 1616
About Machine Mode ... 1616
Running Machine Output ... 1617
Running machineOutput in Probe Mode ... 1617
Running machineOutput in Validate Host Mode ... 1617
Running machineOutput in Import to vSphere Mode ... 1617
Running the Machine Mode Export from vSphere Operation ... 1618
Example Output ... 1619
Output from Running machineOutput in Probe Mode ... 1619
Output from Running machineOutput in Probe Mode ... 1619
Output from Running machineOutput in Validate Host Mode ... 1623
Output from Running machineOutput in Validate Host Mode ... 1623
Output from Running machineOutput in Import Mode ... 1624
Output from Running machineOutput in Import Mode ... 1624

Output from Running machineOutput in Export Mode 1625
Output from Running machineOutput in Export Mode 1625
Documentation Legal Notice 1626

Release Notes

Release notes include enhancements and notices, bug fixes, and resolved issues delivered for the various releases of the vSphere APIs, SDKs, and tools offered by VMware.

vSphere Management SDK

vSphere Management SDK 8.0 Update 3 Release Notes

This document contains the following sections

- Introduction
- Overview
- Distribution Kit
- Compatibility Notices
- New in this Release
- Deprecated Features
- Resolved Issues
- Known Issues
- Known Issues in VSLM

Introduction

vSphere Management SDK 8.0 Update 3 | 25 June 2024 | Build on developer.broadcom.com

For VMware Cloud Foundation and vSphere

Check back for additions and updates to these release notes, marked **New**.

Overview

The vSphere Management SDK contains the following packaged interfaces. All are Web services, in the sense that they communicate through a network endpoint in XML as defined by their standard Web Services Description Language (WSDL) definitions. In vSphere 8 update releases, applications can communicate in JSON also, which enhances interoperability with vSphere Automation APIs.

- ESX Agent Manager (EAM) SDK
- Storage Management Service (SMS) SDK
- Storage Policy Based Management (SPBM) SDK.
- Single Sign On (SSO) Client SDK
- vSphere Web Services SDK
- Virtual Storage Lifecycle Management (VSLM) SDK

The vSphere Web Services SDK is by far the most popular. It supports the development of applications that call the vSphere API to manage virtual machines and virtual infrastructure components such as data centers, datastores, resource pools, and networks. Refer to the *vSphere Web Services API Reference* for managed objects, methods, and data structures in this release.

EAM offers support for vSphere solutions such as High Availability, NSX virtual networks, and IO Filters. For details, refer to the *vSphere Solutions Manager, vServices, and ESX Agent Manager Release Notes*.

SMS provides access to storage capabilities, associations, and space usage. SMS has its own in-memory database manager (the SMS cache) that periodically synchronizes data with the storage information provider database.

SPBM allows administrators to use and define storage profiles that help automate storage provisioning for virtual machines. Client applications can manipulate storage policies with the API, as well as with the vSphere Client.

SSO provides a security token service for single sign-on authentication. Client applications call the API to obtain Security Assertion Markup Language (SAML) tokens for logging into vCenter Server or vCloud Suite. For multifactor authentication, programs must use the vSphere Automation SDK.

Introduced in vSphere 7.0, VSLM is a set of APIs to manage First Class Disk (FCD), infrequently called Improved Virtual Disk. FCD is storage that exists independently of a virtual machine.

Distribution Kit

All the above SDK subsets are available on the developer.vmware.com website as part of the vSphere Management SDK, a collection of related APIs. When you extract the contents of the ZIP archive, SDK subsets appear in these subdirectories:

VMware-vSphere-SDK-8.0.3-*buildNumber*.zip

SDK

eam

sms-sdk

spbm

ssoclient

vsphere-ws

vslm

These vSphere Management SDKs provide documentation, libraries, and code examples needed for developers to build solutions integrated with the industry's leading virtualization platform.

Compatibility Notices

VI/JSON protocol added to Web Services API. Starting with vSphere 8.0 U1, VMware adds support for a new HTTP and JSON-based wire protocol as an alternative to SOAP with XML. The new protocol is described using the industry-standard OpenAPI specification version 3.0, (see SwaggerIO website) and can access the popular VIM APIs, described in the *Web Services API Reference*. See Chapter 4 of the *Web Services SDK Programming Guide* for more details. Further improvements were made for vSphere 8.0 U2 and U3.

.NET SDK was discontinued. As announced in August 2020 (KB 80144) vSphere Automation SDK for .NET was deprecated. The vSphere Management SDK for .NET was also deprecated in vSphere 7.0 U3d (P04) as outlined in KB 87965. C# sample programs in the SDK were deleted. Programmers can still write Java programs for vSphere Web Services, or generate .NET bindings using WSDLs published in the SDK and using .NET libraries.

Java 11 and beyond. The SDK stubs and samples are compatible with JDK 8 and above. Starting with JDK 11, **java.xml.ws** (JAX-WS) and the related technologies SAAJ, Web Services Metadata, and **java.xml.bind** (JAXB) are not bundled with the JDK. If you re-build and run sample code with JDK 11 and above, use the JAX-WS and JAXB libraries (version 3.0.2) bundled with the SDK under the **lib** folder.

To compile the sample programs, run **build.bat -w** on Windows or **build.sh -w** on Linux. To run the sample programs, type **run.bat** or **run.sh** with appropriate arguments.

New in this Release

Here are additions in 8.0 U3, as described in the *vSphere Web Services API* reference.

- **New Methods in Managed Objects**
  - ProvisionServerPrivateKey added to HostCertificateManager: provisions an SSL private key for use before SSL certificate installation.
  - startDpuFailover added to HostNetworkSystem: launches DPU failover for a given distributed virtual switch.
  - HostQueryVirtualDiskUuid and HostSetVirtualDiskUuid_Task added to HostVStorageObjectManager: get and set the virtual disk UUID on host.
  - QueryVirtualDiskUuidEx and SetVirtualDiskUuidEx_Task added to VcenterVStorageObjectManager: get and set the virtual disk UUID on vCenter.
  - CreateCollectorWithInfoFilterForTasks added to TaskManager: create a specialized HistoryCollector to gather TaskInfo data objects.
- **New Data Objects**
  - ClusterComputeResourceCryptoModePolicy: encryption mode policy for a cluster.
  - ClusterComputeResourceHostEvacuationInfo: evacuation actions for the host.
  - ClusterComputeResourceMaintenanceInfo: how hosts will be put into maintenance mode.
  - ClusterFtVmHostRuleInfo: control placement of VMs across two host groups.
  - ComputeResourceHostSeedSpec: seed host specification.
  - DVSFilterSpecConnecteeSpec: base class for Connectee filters, with DVSFilterSpecPnicConnecteeSpec for physical NIC, DVSFilterSpecVmConnecteeSpec for VM, and DVSFilterSpecVmknicConnecteeSpec for vmknic.
  - DVSFilterSpecVlanSpec: base class for VlanSpec filters, with DVSFilterSpecPvlanSpec for private vLAN, DVSFilterSpecTrunkVlanSpec for trunking, and DVSFilterSpecVlanIdSpec for regular Vlan.
  - DistributedVirtualSwitchHostMemberHostUplinkState: runtime state of uplink on host.
  - HostCpuSchedulerInfo: information about CPU scheduler.
  - HostPartialMaintenanceModeRuntimeInfo: status of partial maintenance mode.
  - IoFilterManagerSslTrust: SSL trust policy, either PinnedCertificate or UntrustedCertificate.
  - SnapshotSelectionSpec: specification for removing snapshots from VM.
  - TaskInfoFilterSpec: filter for retrieving task history.
  - TaskInfoFilterSpecFilterTaskResults: further specify task history results.
  - VmwareDistributedVirtualSwitchDpuFailoverPolicy: failover policy for DPU.
  - VmwareDistributedVirtualSwitchNetworkOffloadConfig: network offload config for DVS.
- **New Enumerated Types**
  - HostCpuSchedulerInfoCpuSchedulerPolicyInfo: hyperthread security.
  - HostDistributedVirtualSwitchManagerFailoverReason: reasons for DPU failure.
  - HostDistributedVirtualSwitchManagerFailoverStage: stage of DPU failover.
  - HostGraphicsConfigVgpuMode: mixed or same size vGPU time slice.
  - HostGraphicsInfoVgpuMode: multi-instance or time-sliced virtual GPU.
  - HostPartialMaintenanceModeId: mode of quick patch partial maintenance mode.
  - HostPartialMaintenanceModeStatus: status modes of partial maintenance mode.
  - ScsiLunLunReservationStatus: whether SCSI LUN is reserved or not.
- **New Fault Type**
  - FtVmHostRuleViolation: if powered on, VM would violate Fault Tolerance.

Finer-grained VM storage policy privileges introduced to benefit VMware Cloud customers. See William Lam's 06.26.2012 blog, *Improved VM Storage Policy* in vSphere 8.x.

Deprecated Features

- **External identity providers surpass the vSphere Management SDK.**

The vSphere Management SDK interfaces support neither multifactor authentication through OAuth 2.0, nor third-party providers such as Active Directory Federation Services (ADFS), Okta, or Microsoft Entra AD. Programmers can use the vSphere Automation API to support these authorization and authentication services. SAML tokens are valid for both APIs. Applications can obtain a SAML token by using either the Issue service of the vSphere Management API or the Authentication Token service of the vSphere Automation API. The Token service can also be used to convert a JSON Web Token (JWT) to a SAML token suitable for vCenter Server authentication.

- **Use of TLS in old versions of Java**

  Java programmers should use Java 11 or possibly later versions. JDK 1.7 should not be used in a production environment because it does not support TLS 1.2. Stubs in the SDK were compiled with JDK 11 and do not need rebuilding.
- **See vSphere API Reference for deprecated interfaces**

  Several properties and types defined for the API were deprecated in vSphere API 7.0. You can see all the objects and types with indicators showing when they were added or deprecated by clicking “API Versions Reference” in the *VMware vSphere API Reference* manual.

Resolved Issues

- **Clone properly duplicates virtual machine FCD.**

  Previously if a virtual machine contained FCD, **CloneVM** cloned it as a regular disk. As of 8.0 U2, **CloneVM** creates a duplicate FCD, as it should.
- **vSphere API returns true vClock for FCD.**

  Cloud Native Storage (CNS) requires accurate virtual clock time, so as of 8.0 U2 the FCD API returns true vClock time in object **vslmVClockInfo**.
- **Linux virtual machines could not detect vNUMA topology.**

  The vSphere 8.0 API was extended to configure and query vNUMA enablement for HotAdd.

Known Issues

Currently known issues are as follows. For older issues, see the vSphere 7 Release Notes.

- **With Active Directory, vCenter returns only 2500 user groups.**

  When vCenter Server is integrated with Active Direcotry, method UserDirectory.RetrieveUserGroups returns only the first 2500 user groups, and no option exists to search for additional users beyond those 2500.
- **Client received SOAP fault from server: Signature is invalid.**

  When running Web Services sample code with JDK 8 build 232 and later, the Java client receives an exception in the main thread saying “ws.fault.Server SOAPFaultException: Client received SOAP Fault from server: Signature is invalid.” See KB 81799 for workarounds.
- **HBA Rescan might require delay before proceeding.**

  As larger storage devices are connected to the host bus adapter (HBA) in recent releases, HBA rescan could take longer to complete. As a workaround in these cases, developers can add a **sleep()** call before issuing subsequent I/O commands.
- **PropertyCollector notifications more sensitive in 6.7.**

  Programs should not expect **WaitForUpdatesEx** to provide notification of server-side changes. This method is intended as a data synchronization mechanism. In some situations, server-side logic can report change sets that include unchanged properties. Client programs can safely apply values in these change sets, but some items might not represent changes to previous values. Between vSphere 6.5 and 6.7 a change threshold was made more sensitive, which could result in more frequent notifications from **WaitForUpdatesEx**.
- **Java GetVMFiles code mishandles special characters in VM names.**

Under JAX-WS samples, the **GetVMFiles.java** program does not properly handle special characters of VM names passed in parameters.

- **SMS QueryService exceptions**

  You might see SMS **QueryService** exceptions in the SMS log, such as “com.vmware.vim.sms.fault.QsQueryException” and “com.vmware.vim.query.client.exception.ValidationException: Got status code: 400.” The resolution is to restart vCenter Server's Appliance Management Service from the vSphere Client.
- **VM is created despite an invalid storage policy.**

  Using the vSphere API, you can create a VM with a storage policy that is not in compliance with the associated storage capability, even if the subprofile constraint **forceProvision** is set to false. By contrast, the vSphere Client refuses to create the VM if the storage policy is not compliant.
- **Do not use VirtualMachine.Relocate to change VM storage profile.**

  The **VirtualMachine.Relocate** method specifies a new storage profile for a virtual disk or VM without specifying either a new host or datastore. This is invalid to vCenter Server, which does not change the storage profile. Use **VirtualMachine.Reconfigure** to assign a new storage profile to a virtual disk /or VM.
- **Description of VASA alarms and system events is "Unknown Event ID".**

  The vSphere Client can display VASA system events for the inventory root folder or the datastore associated with a storage device, but vCenter Server uses string "Unknown Event ID" for the event description instead of the message that the VASA provider specified.

Known Issues in VSLM

VSLM interfaces control First Class Disk (FCD) and perform lifecycle operations for persistent volumes outside the lifecycle of a VM or Tanzu pod. Cloud Native Storage (CNS)is an important use case for FCD. These are some limitations of the current product:

- vCenter Server does not serialize operations on an FCD, so applications cannot perform simultaneous operations on the same FCD. Multithreading of long running operations such as clone, relocate, delete, retrieve, and so on can have unpredictable results. To avoid problems, perform complete operations in sequential order on a single FCD.
- FCD is not a managed object and has no global lock mechanism protecting multiple writes to one FCD. As a result, VSLM does not support multiple vCenter instances managing one FCD. If you require multiple vCenter instances with FCDs, you have two options: (1) multiple vCenter Server instances managing different datastores, or (2) multiple vCenter Server instances not operating on the same FCD.
- With FCD attached to a VM, the vCenter inventory is not cache consistent with the FCD global catalog. In other words, if your application connects to the API endpoint for VSLM, and also to another API endpoint such as **vpxd** or **hostd**, timing issues can result in database consistency issues. After writes, VMware recommends including a 30 second delay for synchronization of databases.
- When a system error causes a long queue into the FCD global catalog, FCD and Cloud Native Storage (CNS) may become unresponsive.

vSphere Management SDK 8.0 Update 2 Release Notes

This document contains the following sections

- Introduction
- Overview
- Distribution Kit
- Compatibility Notices
- New in this Release
- Deprecated Features

- Resolved Issues
- Known Issues
- Known Issues in VSLM

Introduction

vSphere 8.0 U2 release 21 September 2023 | SDK build on developer.vmware.com

Initial Availability became GA | Last document update 19 February 2024

Check back for additions and updates to these release notes, marked **New**.

Overview

The vSphere Management SDK contains the following packaged interfaces. All are Web services, in the sense that they communicate through a network endpoint in XML as defined by their standard Web Services Description Language (WSDL) definitions. In vSphere 8 update releases, applications can communicate in JSON also, which enhances interoperability with vSphere Automation APIs.

- ESX Agent Manager (EAM) SDK
- Storage Management Service (SMS) SDK
- Storage Policy Based Management (SPBM) SDK.
- Single Sign On (SSO) Client SDK
- vSphere Web Services SDK
- Virtual Storage Lifecycle Management (VSLM) SDK

The vSphere Web Services SDK is by far the most popular. It supports the development of applications that call the vSphere API to manage virtual machines and virtual infrastructure components such as data centers, datastores, resource pools, and networks. Refer to the *vSphere Web Services API Reference* for managed objects, methods, and data structures in this release.

EAM offers support for vSphere solutions such as High Availability, NSX virtual networks, and IO Filters. For details, refer to the *vSphere Solutions Manager, vServices, and ESX Agent Manager Release Notes*.

SMS provides access to storage capabilities, associations, and space usage. SMS has its own in-memory database manager (the SMS cache) that periodically synchronizes data with the storage information provider database.

SPBM allows administrators to use and define storage profiles that help automate storage provisioning for virtual machines. Client applications can manipulate storage policies with the API, as well as with the vSphere Client.

SSO provides a security token service for single sign-on authentication. Client applications call the API to obtain Security Assertion Markup Language (SAML) tokens for logging into vCenter Server or vCloud Suite. For multifactor authentication, programs must use the vSphere Automation SDK.

Introduced in vSphere 7.0, VSLM is a set of APIs to manage First Class Disk (FCD), infrequently called Improved Virtual Disk. FCD is storage that exists independently of a virtual machine.

Distribution Kit

All the above SDK subsets are available on the developer.vmware.com website as part of the vSphere Management SDK, a collection of related APIs. When you extract the contents of the ZIP archive, SDK subsets appear in these subdirectories:

VMware-vSphere-SDK-8.0.2-*buildNumber*.zip

SDK

eam

sms-sdk

spbm

ssoclient

vsphere-ws

vslm

These vSphere Management SDKs provide documentation, libraries, and code examples needed for developers to build solutions integrated with the industry's leading virtualization platform.

Compatibility Notices

New: VI/JSON protocol added to Web Services API. Starting with vSphere 8.0 U1, VMware adds support for a new HTTP and JSON-based wire protocol as an alternative to SOAP with XML. The new protocol is described using the industry-standard OpenAPI specification version 3.0, (see SwaggerIO website) and can access the popular VIM APIs, described in the *Web Services API Reference*. See Chapter 4 of the *Web Services SDK Programming Guide* for more details. Further improvements were made for vSphere 8.0 U2.

.NET SDK was discontinued. As announced in August 2020 (KB 80144) vSphere Automation SDK for .NET was deprecated. The vSphere Management SDK for .NET was also deprecated in vSphere 7.0 U3d (P04) as outlined in KB 87965. C# sample programs in the SDK were deleted. Programmers can still write Java programs for vSphere Web Services, or generate .NET bindings using WSDLs published in the SDK and using .NET libraries.

Java 11. The SDK stubs and samples are compiled with JDK 11, but are backward compatible with Java 8. A recent Java version, JDK 11, discontinued libraries supporting SOAP and JAX-WS. Consequently VMware provides the necessary JAX-WS libraries to run with JDK 11. When you build and run sample code with JDK 11, and for your own code, JAX-WS and JAXB libraries are present in the SDK **lib** folder.

To compile the sample programs, run **build.bat -w** on Windows or **build.sh -w** on Linux. To run the sample programs, type **run.bat** or **run.sh** with appropriate arguments.

New in this Release

New Here are additions in 8.0 U2, as described in the Web Services API reference:

- **New Methods in Managed Objects**
  - SetServiceAccount method added to ExtensionManager object: update authentication service account for an extension.
  - QueryFileLockInfo method added to FileManager object: show any lock information that can be queried from the host.
  - Extended VStorageObject methods in VStorageObjectManagerBase object: rename, revert, create/delete snapshot, and extend disk.
- **New Data Objects**
  - FileLockInfo and FileLockInfoResult for QueryFileLockInfo method.
  - HostVvolNQN and HostVvolVolumeHostVvolNQN to hold NVMe Qualified Name.
  - VStorageObjectSnapshot for VStorageObjectCreateSnapshotEx_Task.
  - VirtualMachineVMotionStunTimeInfo to measure efficiency of vMotion.
- **New Enumerated Types**
  - ClusterSystemVMsConfigInfoDeploymentMode: is VM managed or not?
  - HostBIOSInfoFirmwareType: BIOS or UEFI
  - HostFirewallSystemRuleSetId: lists firewall ruleset IDs.

  - HostFirewallSystemServiceName: lists firewall service names.
  - VirtualNVMEControllerSharing: can devices share NVMe controller?
- **New Fault Type**

  - InvalidScheduledTask: unauthorized user tried to run a scheduled task.

Finer-grained VM storage policy privileges introduced to benefit VMware Cloud customers. See William Lam's 06.26.2012 blog, *Improved VM Storage Policy* in vSphere 8.x.

Deprecated Features

- **External identity providers surpass the vSphere Management SDK.**

  The vSphere Management SDK interfaces support neither multifactor authentication through OAuth 2.0, nor third-party providers such as Active Directory Federation Services (ADFS), Okta, or Microsoft Entra AD. Programmers can use the vSphere Automation API to support these authorization and authentication services. SAML tokens are valid for both APIs. Applications can obtain a SAML token by using either the Issue service of the vSphere Management API or the Authentication Token service of the vSphere Automation API. The Token service can also be used to convert a JSON Web Token (JWT) to a SAML token suitable for vCenter Server authentication.
- **Use of TLS in old versions of Java**

  Java programmers should use Java 11 or possibly later versions. JDK 1.7 should not be used in a production environment because it does not support TLS 1.2. Stubs in the SDK were compiled with JDK 11 and do not need rebuilding.
- **See vSphere API Reference for deprecated interfaces**

  Several properties and types defined for the API were deprecated in vSphere API 7.0. You can see all the objects and types with indicators showing when they were added or deprecated by clicking “API Versions Reference” in the *VMware vSphere API Reference* manual.

Resolved Issues

- **New: Clone properly duplicates virtual machine FCD.**

  Previously if a virtual machine contained FCD, CloneVM cloned it as a regular disk. In 8.0 U2, CloneVM creates a duplicate FCD, as it should.
- **New: vSphere API returns true vClock for FCD.**

  Cloud Native Storage (CNS) requires accurate vClick time, so in this release the FCD API returns true vClock time in objectVClock.
- **Linux virtual machines could not detect vNUMA topology.**

  The vSphere 8.0 API was extended to configure and query vNUMA enablement for HotAdd.

Known Issues

Currently known issues are as follows. For older issues, see the vSphere 7 Release Notes.

- **New: With Active Directory, vCenter returns only 2500 user groups.**

  When vCenter Server is integrated with Active Direcotry, method UserDirectory.RetrieveUserGroups returns only the first 2500 user groups, and no option exists to search for additional users beyond those 2500.
- **Client received SOAP fault from server: Signature is invalid.**

  When running Web Services sample code with JDK 8 build 232 and later, the Java client receives an exception in the main thread saying “ws.fault.Server SOAPFaultException: Client received SOAP Fault from server: Signature is invalid.” See KB 81799 for workarounds.
- **HBA Rescan might require delay before proceeding.**

As larger storage devices are connected to the host bus adapter (HBA) in recent releases, HBA rescan could take longer to complete. As a workaround in these cases, developers can add a **sleep()** call before issuing subsequent I/O commands.

- **PropertyCollector notifications more sensitive in 6.7.**

  Programs should not expect **WaitForUpdatesEx** to provide notification of server-side changes. This method is intended as a data synchronization mechanism. In some situations, server-side logic can report change sets that include unchanged properties. Client programs can safely apply values in these change sets, but some items might not represent changes to previous values. Between vSphere 6.5 and 6.7 a change threshold was made more sensitive, which could result in more frequent notifications from **WaitForUpdatesEx**.
- **Java GetVMFiles code mishandles special characters in VM names.**

  Under JAX-WS samples, the **GetVMFiles.java** program does not properly handle special characters of VM names passed in parameters.
- **SMS QueryService exceptions**

  You might see SMS **QueryService** exceptions in the SMS log, such as “com.vmware.vim.sms.fault.QsQueryException” and “com.vmware.vim.query.client.exception.ValidationException: Got status code: 400.” The resolution is to restart vCenter Server's Appliance Management Service from the vSphere Client.
- **VM is created despite an invalid storage policy.**

  Using the vSphere API, you can create a VM with a storage policy that is not in compliance with the associated storage capability, even if the subprofile constraint **forceProvision** is set to false. By contrast, the vSphere Client refuses to create the VM if the storage policy is not compliant.
- **Do not use VirtualMachine.Relocate to change VM storage profile.**

  The **VirtualMachine.Relocate** method specifies a new storage profile for a virtual disk or VM without specifying either a new host or datastore. This is invalid to vCenter Server, which does not change the storage profile. Use **VirtualMachine.Reconfigure** to assign a new storage profile to a virtual disk /or VM.
- **Description of VASA alarms and system events is "Unknown Event ID".**

  The vSphere Client can display VASA system events for the inventory root folder or the datastore associated with a storage device, but vCenter Server uses string "Unknown Event ID" for the event description instead of the message that the VASA provider specified.

Known Issues in VSLM

VSLM interfaces control First Class Disk (FCD) and perform lifecycle operations for persistent volumes outside the lifecycle of a VM or Tanzu pod. Cloud Native Storage (CNS)is an important use case for FCD. These are some limitations of the current product:

- vCenter Server does not serialize operations on an FCD, so applications cannot perform simultaneous operations on the same FCD. Multithreading of long running operations such as clone, relocate, delete, retrieve, and so on can have unpredictable results. To avoid problems, perform complete operations in sequential order on a single FCD.
- FCD is not a managed object and has no global lock mechanism protecting multiple writes to one FCD. As a result, VSLM does not support multiple vCenter instances managing one FCD. If you require multiple vCenter instances with FCDs, you have two options: (1) multiple vCenter Server instances managing different datastores, or (2) multiple vCenter Server instances not operating on the same FCD.
- With FCD attached to a VM, the vCenter inventory is not cache consistent with the FCD global catalog. In other words, if your application connects to the API endpoint for VSLM, and also to another API endpoint such as **vpxd** or **hostd**, timing issues can result in database consistency issues. After writes, VMware recommends including a 30 second delay for synchronization of databases.
- When a system error causes a long queue into the FCD global catalog, FCD and Cloud Native Storage (CNS) may become unresponsive.

vSphere Management SDK 8.0 Update 1 Release Notes

This document contains the following sections

- Introduction
- Overview
- Distribution Kit
- Compatibility Notices
- New in this Release
- Deprecated Features
- Known Issues
- Known Issues in VSLM
- Known Issues in Documentation
- Resolved Issues

Introduction

vSphere 8.0 U1 release date 18 April 2023 | SDK build on developer.vmware.com

Initial Availability may become GA on 9 May 2023 | Last document update 18 April 2023

Check back for additions and updates to these release notes, marked **New**.

Overview

The vSphere Management SDK contains the following packaged interfaces. All are Web services, in the sense that they communicate through a network endpoint in XML as defined by their standard Web Services Description Language (WSDL) definitions.

- ESX Agent Manager (EAM) SDK
- Storage Management Service (SMS) SDK
- Storage Policy Based Management (SPBM) SDK.
- Single Sign On (SSO) Client SDK
- vSphere Web Services SDK
- Virtual Storage Lifecycle Management (VSLM) SDK

The vSphere Web Services SDK is by far the most popular. It supports the development of applications that call the vSphere API to manage virtual machines and virtual infrastructure components such as data centers, datastores, resource pools, and networks. Refer to the *vSphere Web Services API Reference* for managed objects, methods, and data structures in this release.

EAM offers support for vSphere solutions such as High Availability, NSX virtual networks, and IO Filters. For details, refer to the *vSphere Solutions Manager, vServices, and ESX Agent Manager Release Notes*.

SMS provides access to storage capabilities, associations, and space usage. SMS has its own in-memory database manager (the SMS cache) that periodically synchronizes data with the storage information provider database.

SPBM allows administrators to use and define storage profiles that help automate storage provisioning for virtual machines. Client applications can manipulate storage policies with the API, as well as with the vSphere Client.

SSO provides a security token service for single sign-on authentication. Client applications call the API to obtain Security Assertion Markup Language (SAML) tokens for logging into vCenter Server or vCloud Suite. For multifactor authentication, programs must use the vSphere Automation SDK.

Introduced in vSphere 7.0, VSLM is a set of APIs to manage First Class Disk (FCD), infrequently called Improved Virtual Disk. FCD is storage that exists independently of a virtual machine.

Distribution Kit

All the above SDK subsets are available on the developer.vmware.com website as part of the vSphere Management SDK, a collection of related APIs. When you extract the contents of the ZIP archive, SDK subsets appear in these subdirectories:

VMware-vSphere-SDK-8.0.1-*buildNumber*.zip

SDK

eam

sms-sdk

spbm

ssoclient

vsphere-ws

vslm

These vSphere Management SDKs provide documentation, libraries, and code examples needed for developers to build solutions integrated with the industry's leading virtualization platform.

Compatibility Notices

New JSON RPC protocol added to Web Services API. Starting with vSphere 8.0 Update 1, VMware adds support for a new HTTP and JSON-based wire protocol as an alternative to SOAP with XML. The new protocol is described using the industry-standard OpenAPI specification version 3.0, (see SwaggerIO website) and can access the popular VIM APIs, described in the *Web Services API Reference*. See Chapter 4 of the *Web Services SDK Programming Guide* for more details.

.NET SDK was discontinued. As announced in August 2020 (KB 80144) vSphere Automation SDK for .NET was deprecated. The vSphere Management SDK for .NET was also deprecated in vSphere 7.0 U3d (P04) as outlined in KB 87965. C# sample programs in the SDK were deleted. Programmers can still write Java programs for vSphere Web Services, or generate .NET bindings using WSDLs published in the SDK and using .NET libraries.

Java 11. The SDK stubs and samples are compiled with JDK 11, but are not backward compatible with Java 8. A recent Java version, JDK 11, discontinued libraries supporting SOAP and JAX-WS. Consequently VMware provides the necessary JAX-WS libraries to run with JDK 11. When you build and run sample code with JDK 11, and for your own code, JAX-WS and JAXB libraries are present in the SDK **lib** folder.

Null pointer exception. Running vSphere Management SDK with Java 11 can result in NullPointerException when com.vmware.vim25#VimService() calls the javax.xml.ws.Service constructor. This is due to a known bug in old versions of **jaxws-ri**. The fix is to run version 3.0.1 or later of **jaxws-ri**. See KB 89346 for details.

To compile the sample programs, run **build.bat -w** on Windows or **build.sh -w** on Linux. To run the sample programs, type **run.bat** or **run.sh** with appropriate arguments.

New in this Release

New 8.0 U1 features, further described in the Web Services API reference:

- **New Methods in Managed Objects**

  - GetCryptoKeyStatus in CryptoManagerHost
  - SetKeyCustomAttributes in CryptoManagerKmip
  - IncreaseDirectorySize and QueryDirectoryInfo in DatastoreNamespaceManager
  - RetrieveCertificateInfoList in HostCertificateManager
  - property complianceCheckTime in managed object HostProfile
- **New Data Objects**
  - CryptoManagerHostKeyStatus
  - CryptoManagerKmipCertSignRequest
  - CryptoManagerKmipCustomAttributeSpec
  - DatastoreNamespaceManagerDirectoryInfo
  - HostCertificateManagerCertificateSpec
  - HostTpmBootCompleteEventDetails
  - VimVasaProviderVirtualHostConfig
- **New Enumerated Types**
  - CryptoManagerHostKeyManagementType
  - DeviceProtocol
  - HostCertificateManagerCertificateKind

Important features in previous releases:

- **Namespace support for Kubernetes.** In vSphere 7.0 U3, the “namespace” property was added to Folder and ResourcePool. Namespace is a resource that divides up cluster resources so that administrators can assign Kubernetes environments to specific developers or development teams.
- The Web Services SDK includes new JAX-WS and JAXB libraries so you don't need to download them separately.
- Sticky bit for ESXi configuration files has been replaced by use of **esxcli** configuration settings.
- HTTP PUT access to host files was discontinued for security reasons. GET access is still allowed.

Deprecated Features

- **vSphere Automation identity providers surpass SSO interfaces**

  SSO interfaces support neither multifactor authentication, nor third party providers such as Active Directory Federation Services (ADFS), OAuth2, Okta, or OpenID Connect. Programmers can use the vSphere Automation APIs to support these authorization and authentication services. SAML tokens are valid for both APIs. Applications can either use **LoginByToken** with SOAP-based SSO, or use the vSphere Automation API token service to exchange a JSON Web Token (JWT) for a SAML token.
- **Use of TLS in old versions of Java**

  Java programmers should use Java 11 or possibly later versions. JDK 1.7 should not be used in a production environment because it does not support TLS 1.2. Stubs in the SDK were compiled with JDK 11 and do not need rebuilding.
- **See vSphere API Reference for deprecated interfaces**

  Several properties and types defined for the API were deprecated in vSphere API 7.0. You can see all the objects and types with indicators showing when they were added or deprecated by clicking “API Versions Reference” in the *VMware vSphere API Reference* manual.

Known Issues

Currently known issues are as follows. For older issues, see the vSphere 7 Release Notes.

- **Client received SOAP fault from server: Signature is invalid.**

When running Web Services sample code with JDK 8 build 232 and later, the Java client receives an exception in the main thread saying “ws.fault.Server SOAPFaultException: Client received SOAP Fault from server: Signature is invalid.” See KB 81799 for workarounds.

- **HBA Rescan might require delay before proceeding.**

  As larger storage devices are connected to the host bus adapter (HBA) in recent releases, HBA rescan could take longer to complete. As a workaround in these cases, developers can add a **sleep()** call before issuing subsequent I/O commands.
- **PropertyCollector notifications more sensitive in 6.7.**

  Programs should not expect **WaitForUpdatesEx** to provide notification of server-side changes. This method is intended as a data synchronization mechanism. In some situations, server-side logic can report change sets that include unchanged properties. Client programs can safely apply values in these change sets, but some items might not represent changes to previous values. Between vSphere 6.5 and 6.7 a change threshold was made more sensitive, which could result in more frequent notifications from **WaitForUpdatesEx**.
- **Java GetVMFiles code mishandles special characters in VM names.**

  Under JAX-WS samples, the **GetVMFiles.java** program does not properly handle special characters of VM names passed in parameters.
- **SMS QueryService exceptions**

  You might see SMS **QueryService** exceptions in the SMS log, such as “com.vmware.vim.sms.fault.QsQueryException” and “com.vmware.vim.query.client.exception.ValidationException: Got status code: 400.” The resolution is to restart vCenter Server's Appliance Management Service from the vSphere Client.
- **VM is created despite an invalid storage policy.**

  Using the vSphere API, you can create a VM with a storage policy that is not in compliance with the associated storage capability, even if the subprofile constraint **forceProvision** is set to false. By contrast, the vSphere Client refuses to create the VM if the storage policy is not compliant.
- **Do not use VirtualMachine.Relocate to change VM storage profile.**

  The **VirtualMachine.Relocate** method specifies a new storage profile for a virtual disk or VM without specifying either a new host or datastore. This is invalid to vCenter Server, which does not change the storage profile. Use **VirtualMachine.Reconfigure** to assign a new storage profile to a virtual disk /or VM.
- **Description of VASA alarms and system events is "Unknown Event ID".**

  The vSphere Client can display VASA system events for the inventory root folder or the datastore associated with a storage device, but vCenter Server uses string "Unknown Event ID" for the event description instead of the message that the VASA provider specified.

Known Issues in VSLM

VSLM interfaces control First Class Disk (FCD) and perform lifecycle operations for persistent volumes outside the lifecycle of a VM or Tanzu pod. Cloud Native Storage (CNS)is an important use case for FCD. These are some limitations of the current product:

- vCenter Server does not serialize operations on an FCD, so applications cannot perform simultaneous operations on the same FCD. Multithreading of long running operations such as clone, relocate, delete, retrieve, and so on can have unpredictable results. To avoid problems, perform complete operations in sequential order on a single FCD.
- FCD is not a managed object and has no global lock mechanism protecting multiple writes to one FCD. As a result, VSLM does not support multiple vCenter instances managing one FCD. If you require multiple vCenter instances with FCDs, you have two options: (1) multiple vCenter Server instances managing different datastores, or (2) multiple vCenter Server instances not operating on the same FCD.

- With FCD attached to a VM, the vCenter inventory is not cache consistent with the FCD global catalog. In other words, if your application connects to the API endpoint for VSLM, and also to another API endpoint such as **vpxd** or **hostd**, timing issues can result in database consistency issues. After writes, VMware recommends including a 30 second delay for synchronization of databases.
- When a system error causes a long queue into the FCD global catalog, FCD and Cloud Native Storage (CNS) may become unresponsive.

Known Issues in Documentation

- **Revised documentation for tracking Task completion.**

  The *Web Services SDK Programming Guide* contains revised sections about PropertyCollector and polling for Task status. Furthermore, although VMware does not officially support Python bindings, the GitHub project for **pyvmomi** contains the relatively short **task.py** program that can help developers understand Task tracking algorithms.
- **VirtualMachine.CloneVM_Task method failure.**

  The *VMware vSphere API Reference* does not document a limitation with regard to **VirtualMachineCloneSpec**:

  When **VirtualMachine.CloneVM_Task** fails, it produces the following error: “The specified delta disk format 'nativeFormat' is not supported.” The vCenter Server returns the error when you:

  (1) Create a VAAI NAS native linked clone VM that uses **nativeFormat** for the delta disk format **VirtualMachineCloneSpec.location.disk.diskBackingInfo.deltaDiskFormat**, or

  (2) Create a second-level clone from the first clone, and the disk for the second-level clone is on a different datastore. If you do not specify **seSparseFormat** or **redoLogFormat** for the second-level clone delta disk format, the clone operation fails.

  If you are using a VM clone with a native delta disk format, you must specify either **seSparseFormat** or **redoLogFormat** for any clones that you create from the original native clone when the second-level clone is on a different datastore.
- **The fault parameter is required for the SetTaskState method.**

  The *VMware vSphere API Reference* does not document this limitation with regard to the **SetTaskState** method:

  If you specify an error state when you call the **SetTaskState** method, you must also specify the **fault** parameter in the calling sequence.
- **The seSparse virtual disk type is intended for internal use only.**

  The documentation does not specify internal use only for **VirtualDiskType.seSparse**.

Resolved Issues

- **Linux virtual machines could not detect vNUMA topology.**

  The vSphere API was extended to configure and query vNUMA enablement for HotAdd.
- **Various USB issues fixed.**

  USB 3.2 Gen 2x2 devices work and report the corrected speed when connected to USB 3.2 Gen 2x2 root or external hub. USB 3.2 device passthrough to virtual machines also works.
- **Guest customization custom script enhancements.**

  Increased the size limit to 64KB in software, which also applies to vCD and the API.

vSphere Management SDK 8.0 Release Notes

This document contains the following sections

- Introduction
- Overview
- Distribution Kit
- Compatibility Notices
- New in this Release
- Deprecated Features
- Known Issues
- Known Issues in VSLM
- Known Issues in Documentation
- Resolved Issues

Introduction

vSphere 8.0 release date: 11 October 2022 | SDK build on developer.vmware.com

Initial Availability became GA 8 November 2022 | Last document update: 9 November 2022

Check back for additions and updates to these release notes, marked **New**.

Before 7.0 U3, the vSphere Management SDK was not compatible with Java 11. For advice on Java 11, see Compatibility Notices below.

Overview

The vSphere Management SDK contains the following packaged interfaces. All are Web services, in the sense that they communicate through a network endpoint in XML as defined by their standard Web Services Description Language (WSDL) definitions.

- ESX Agent Manager (EAM) SDK
- Storage Management Service (SMS) SDK
- Storage Policy Based Management (SPBM) SDK.
- Single Sign On (SSO) Client SDK
- vSphere Web Services SDK
- Virtual Storage Lifecycle Management (VSLM) SDK

The vSphere Web Services SDK is by far the most popular. It supports the development of applications that call the vSphere API to manage virtual machines and virtual infrastructure components such as data centers, datastores, resource pools, and networks. Refer to the *vSphere Web Services API Reference* for managed objects, methods, and data structures in this release.

EAM offers support for vSphere solutions such as High Availability, NSX virtual networks, and IO Filters. For details, refer to the *vSphere Solutions Manager, vServices, and ESX Agent Manager Release Notes*.

SMS provides access to storage capabilities, associations, and space usage. SMS has its own in-memory database manager (the SMS cache) that periodically synchronizes data with the storage information provider database.

SPBM allows administrators to use and define storage profiles that help automate storage provisioning for virtual machines. Client applications can manipulate storage policies with the API, as well as with the vSphere Client.

SSO provides a security token service for single sign-on authentication. Client applications call the API to obtain Security Assertion Markup Language (SAML) tokens for logging into vCenter Server or vCloud Suite. For multifactor authentication, programs must use the vSphere Automation SDK.

Introduced in vSphere 7.0, VSLM is a set of APIs to manage First Class Disk (FCD), also called Improved Virtual Disk. FCD is storage that exists independently of a virtual machine.

Distribution Kit

All the above SDK subsets are available on the developer.vmware.com website as part of the vSphere Management SDK, a collection of related APIs. When you extract the contents of the ZIP archive, SDK subsets appear in these subdirectories:

VMware-vSphere-SDK-8.0.0-*buildNumber*.zip

SDK

eam

sms-sdk

spbm

ssoclient

vsphere-ws

vslm

These vSphere Management SDKs provide documentation, libraries, and code examples needed for developers to build solutions integrated with the industry's leading virtualization platform.

Compatibility Notices

.NET SDK discontinued. As announced in August 2020 (KB 80144) vSphere Automation SDK for .NET was deprecated. The vSphere Management SDK for .NET was also deprecated in vSphere 7.0 U3d (P04) as outlined in KB 87965. C# sample programs in the SDK have been deleted. Programmers can still write Java programs for vSphere Web Services, or generate .NET bindings using WSDLs published in the SDK and using .NET libraries.

Java 11. The SDK stubs and samples are compiled with JDK 11, but are not backward compatible with Java 8. A recent Java version, JDK 11, discontinued libraries supporting SOAP and JAX-WS. Consequently VMware provides the necessary JAX-WS libraries to run with JDK 11. When you build and run sample code with JDK 11, and for your own code, JAX-WS and JAXB libraries are present in the SDK **lib** folder.

Null pointer exception. Running vSphere Management SDK with Java 11 can result in NullPointerException when com.vmware.vim25#VimService() calls the javax.xml.ws.Service constructor. This is due to a known bug in old versions of **jaxws-ri**. The fix is to run version 3.0.1 or later of **jaxws-ri**. See KB 89346 for details.

To compile the sample programs, run **build.bat -w** on Windows or **build.sh -w** on Linux. To run the sample programs, type **run.bat** or **run.sh** with appropriate arguments.

New in this Release

New 8.0 features, further described in the Web Services API reference:

- **New Methods in Managed Objects**
  - Methods QueryCompatibleVmnicsFromHosts and QuerySupportedNetworkOffloadSpec were added to DistributedVirtualSwitchManager to check compatible vmNICs and network offload specifications.
  - Method RetrieveVendorDeviceGroupInfo was added to HostAssignableHardwareManager to return a list of types in the vendor device group.
  - Methods QueryMaxQueueDepth and SetMaxQueueDepth were added to HostDatastoreSystem to get and change queue depth for the specified NFS datastore.

- **New Data Objects**
  - ClusterClusterInitialPlacementAction describes VM location in a cluster. ClusterPreemptibleVmPairInfo and ClusterPreemptibleVmPairSpec contain the monitored and preemptible VM pair in an HA cluster.
  - DVSManagerPhysicalNicsList stores valid physical NICs for an ESXi host, and DistributedVirtualSwitchNetworkOffloadSpec specifies network offload for a DVS.
  - DpuStatusInfo and DpuStatusInfoOperationalInfo are for operational and health status of a data processing unit (DPU), also known as a Smart NIC.
  - HostDvxClass gives information about device virtualization extensions (DVX) class for direct hardware access.
  - HostFru represents a hardware component that is a field replaceable unit (FRU) in an ESXi host.
  - Security related objects HostProxySwitchEnsInfo (enhanced network stack), HostSgxRegistrationInfo (software guard extensions), HostTpmSignerEventDetails, and HostTpmVersionEventDetails (trusted platform module) were added.
  - VMwareDvsDpuCapability and VirtualMachineDvxClassInfo involve DPU and DVX, previously mentioned.
  - VirtualDeviceDeviceGroupInfo, VirtualMachineVendorDeviceGroupInfo, VirtualMachineVendorDeviceGroupInfoComponentDeviceInfo, VirtualMachineVirtualDeviceGroups, VirtualMachineVirtualDeviceSwap, and VirtualMachineVirtualDeviceSwapDeviceSwapInfo are for managing device groups in virtual machines.
  - VirtualMachineVirtualNuma and VirtualMachineVirtualNumaInfo describe non-uniform memory access (NUMA) for a virtual machine.
  - VirtualPCIPassthroughDvxBackingInfo and VirtualPCIPassthroughDvxBackingOption describe device virtualization extensions (DVX), previously mentioned.
- **New Enumerated Types**
  - HostDVSConfigSpecSwitchMode defines the distributed virtual switch mode.
  - HostMountInfoMountFailedReason enumerates causes for NFS mounts failing.
  - HostSgxRegistrationInfoRegistrationStatus and HostSgxRegistrationInfoRegistrationType relate to SGX, previously mentioned.
  - VirtualHardwareMotherboardLayout enumerates motherboard layout of the VM.
  - VirtualMachineCloneSpecTpmProvisionPolicy says whether to copy or replace vTPM in a cloned VM.
  - VirtualMachineVirtualDeviceSwapDeviceSwapStatus shows status of a swap device.
  - vslmDiskInfoFlag represents various information about first class disk (FCD).
- **Namespace support for Kubernetes**
  - In vSphere 7.0 U3, the “namespace” property was added to Folder and ResourcePool.
  - Namespace is a resource that divides up cluster resources so that administrators can assign Kubernetes environments to specific development teams.
- The Web Services SDK includes new JAX-WS and JAXB libraries so you don't need to download them separately.
- Sticky bit for ESXi configuration files has been replaced by use of **esxcli** configuration settings.
- HTTP PUT access to host files was discontinued for security reasons. GET access is still allowed.

Deprecated Features

- **vSphere Automation identity providers surpass SSO interfaces**

  SSO interfaces support neither multifactor authentication, nor third party providers such as Active Directory Federation Services (ADFS), OAuth2, Okta, or OpenID Connect. Programmers can use the vSphere Automation APIs to support these authorization and authentication services. SAML tokens are valid for both APIs. Applications can either use **LoginByToken** with SOAP-based SSO, or use the vSphere Automation API token service to exchange a JSON Web Token (JWT) for a SAML token.
- **Use of TLS in old versions of Java**

Java programmers should use Java 11 or possibly later versions. JDK 1.7 should not be used in a production environment because it does not support TLS 1.2. Stubs in the SDK were compiled with JDK 11 and do not need rebuilding.

- **See vSphere API Reference for deprecated interfaces**

  Several properties and types defined for the API were deprecated in vSphere API 7.0. You can see all the objects and types with indicators showing when they were added or deprecated by clicking “API Versions Reference” in the *VMware vSphere API Reference* manual.

Known Issues

Currently known issues are as follows. For older issues, see the vSphere 7 Release Notes.

- **Client received SOAP fault from server: Signature is invalid.**

  When running Web Services sample code with JDK 8 build 232 and later, the Java client receives an exception in the main thread saying “ws.fault.Server SOAPFaultException: Client received SOAP Fault from server: Signature is invalid.” See KB 81799 for workarounds.
- **HBA Rescan might require delay before proceeding.**

  As larger storage devices are connected to the host bus adapter (HBA) in recent releases, HBA rescan could take longer to complete. As a workaround in these cases, developers can add a **sleep()** call before issuing subsequent I/O commands.
- **PropertyCollector notifications more sensitive in 6.7.**

  Programs should not expect **WaitForUpdatesEx** to provide notification of server-side changes. This method is intended as a data synchronization mechanism. In some situations, server-side logic can report change sets that include unchanged properties. Client programs can safely apply values in these change sets, but some items might not represent changes to previous values. Between vSphere 6.5 and 6.7 a change threshold was made more sensitive, which could result in more frequent notifications from **WaitForUpdatesEx**.
- **Java GetVMFiles code mishandles special characters in VM names.**

  Under JAX-WS samples, the **GetVMFiles.java** program does not properly handle special characters of VM names passed in parameters.
- **SMS QueryService exceptions**

  You might see SMS **QueryService** exceptions in the SMS log, such as “com.vmware.vim.sms.fault.QsQueryException” and “com.vmware.vim.query.client.exception.ValidationException: Got status code: 400.” The resolution is to restart vCenter Server's Appliance Management Service from the vSphere Client.
- **VM is created despite an invalid storage policy.**

  Using the vSphere API, you can create a VM with a storage policy that is not in compliance with the associated storage capability, even if the subprofile constraint **forceProvision** is set to false. By contrast, the vSphere Client refuses to create the VM if the storage policy is not compliant.
- **Do not use VirtualMachine.Relocate to change VM storage profile.**

  The **VirtualMachine.Relocate** method specifies a new storage profile for a virtual disk or VM without specifying either a new host or datastore. This is invalid to vCenter Server, which does not change the storage profile. Use **VirtualMachine.Reconfigure** to assign a new storage profile to a virtual disk /or VM.
- **Description of VASA alarms and system events is "Unknown Event ID".**

  The vSphere Client can display VASA system events for the inventory root folder or the datastore associated with a storage device, but vCenter Server uses string "Unknown Event ID" for the event description instead of the message that the VASA provider specified.

Known Issues in VSLM

VSLM interfaces control First Class Disk (FCD) and perform lifecycle operations for persistent volumes outside the lifecycle of a VM or Tanzu pod. Cloud Native Storage (CNS)is an important use case for FCD. These are some limitations of the current product:

- vCenter Server does not serialize operations on an FCD, so applications cannot perform simultaneous operations on the same FCD. Multithreading of long running operations such as clone, relocate, delete, retrieve, and so on can have unpredictable results. To avoid problems, perform complete operations in sequential order on a single FCD.
- FCD is not a managed object and has no global lock mechanism protecting multiple writes to one FCD. As a result, VSLM does not support multiple vCenter instances managing one FCD. If you require multiple vCenter instances with FCDs, you have two options: (1) multiple vCenter Server instances managing different datastores, or (2) multiple vCenter Server instances not operating on the same FCD.
- With FCD attached to a VM, the vCenter inventory is not cache consistent with the FCD global catalog. In other words, if your application connects to the API endpoint for VSLM, and also to another API endpoint such as **vpxd** or **hostd**, timing issues can result in database consistency issues. After writes, VMware recommends including a 30 second delay for synchronization of databases.
- When a system error causes a long queue into the FCD global catalog, FCD and Cloud Native Storage (CNS) may become unresponsive.

Known Issues in Documentation

- **Revised documentation for tracking Task completion.**

  The *Web Services SDK Programming Guide* contains revised sections about PropertyCollector and polling for Task status. Furthermore, although VMware does not officially support Python bindings, the GitHub project for **pyvmomi** contains the relatively short **task.py** program that can help developers understand Task tracking algorithms.
- **VirtualMachine.CloneVM_Task method failure.**

  The *VMware vSphere API Reference* does not document a limitation with regard to **VirtualMachineCloneSpec**:

  When **VirtualMachine.CloneVM_Task** fails, it produces the following error: “The specified delta disk format 'nativeFormat' is not supported.” The vCenter Server returns the error when you:

  (1) Create a VAAI NAS native linked clone VM that uses **nativeFormat** for the delta disk format **VirtualMachineCloneSpec.location.disk.diskBackingInfo.deltaDiskFormat**, or

  (2) Create a second-level clone from the first clone, and the disk for the second-level clone is on a different datastore. If you do not specify **seSparseFormat** or **redoLogFormat** for the second-level clone delta disk format, the clone operation fails.

  If you are using a VM clone with a native delta disk format, you must specify either **seSparseFormat** or **redoLogFormat** for any clones that you create from the original native clone when the second-level clone is on a different datastore.
- **The fault parameter is required for the SetTaskState method.**

  The *VMware vSphere API Reference* does not document this limitation with regard to the **SetTaskState** method:

  If you specify an error state when you call the **SetTaskState** method, you must also specify the **fault** parameter in the calling sequence.
- **The seSparse virtual disk type is intended for internal use only.**

  The documentation does not specify internal use only for **VirtualDiskType.seSparse**.

Resolved Issues

- **Linux virtual machines could not detect vNUMA topology.** The vSphere API was extended to configure and query vNUMA enablement for HotAdd.
- **Various USB issues fixed.** USB 3.2 Gen 2x2 devices work and report the corrected speed when connected to USB 3.2 Gen 2x2 root or external hub. USB 3.2 device passthrough to virtual machines also works.
- **Guest customization custom script enhancements.** Increased the size limit to 64KB in software, which also applies to vCD and the API.

vSphere Solutions Manager, vServices, and ESX Agent Manager 8.0 Update 2 Release Notes

This document contains the following sections

- Introduction
- About the vSphere Solutions Manager, vServices, and ESX Agent Manager SDK
- Requirements
- What's New

Introduction

ESX Agent Manager (EAM) SDK 8.0 Update 2| 21 SEPT 2023

Check for additions and updates to these release notes.

About the vSphere Solutions Manager, vServices, and ESX Agent Manager SDK

With vSphere Solutions, vServices, and ESX Agents you can extend the functionality of vCenter Server.

- A vSphere Solution is an object or program that you create by extending specific classes in the vSphere Web Services API. After you register your solution with the instance of the vSphere Web Services API's ExtensionManager managed object associated with your vCenter Server, you see your new solution in the vCenter Server Extensions of your vSphere Client.
  - You can create a vService solution to provide access for a specific application to connect to a service across the network.
  - You can create an ESX Agent to extend the functions of an ESXi host and provide additional services that a vSphere solution requires.
- The vServices Manager and ESX Agent Manager are pre-built solutions that are part of the vCenter Server Extensions functionality within the vCenter Server.
- The VMware ESX Agent Manager includes an API library that allows you to adapt and modify your solutions and view them using the vSphere Client.

For more details about each of these extensibility options, see *Developing and Deploying vSphere Solutions, vServices, and ESXi Agents* for vSphere 8.0 Update 1.

Requirements

These release notes apply to the vSphere 8.0 Update 2 release. You should install the following software to use the 8.0 Update 2 version of the vSphere Solutions Manager, vServices, and ESX Agent Manager SDK.

To use the vSphere Solutions Manager, vServices, and ESX Agent Manager SDKs, download and/or install:

- ESXi 8.0 Update 2 | 21 SEPT 2023
- vCenter Server 8.0 Update 2 | 21 SEPT 2023
- com.vmware.eam.apiref.doc.jar | API Reference
- vsphere-vcenter-801-solutions-vservices-esxi-agent-manager-developer-guide.pdf | Programming Guide

What's New

Deprecation of ESX Agent Manager (EAM) VIB lifecycle API
With vSphere 8.0 Update 2, you can use the EAM API to manage the life cycle of VIBs available on the hosts in a clusters. This support will be dropped in a future vSphere release. Instead of creating EAM agents with VIB URLs, you can use the vSphere Lifecycle Manager Solutions API to perform tasks on a cluster level such as installing, updating, and uninstalling a desired solution version on all hosts in a cluster.

The ESX Agent VM API is not affected by this deprecation.

vSphere Solutions Manager, vServices, and ESX Agent Manager 8.0 Update 1 Release Notes

This document contains the following sections

- Introduction
- About the vSphere Solutions Manager, vServices, and ESX Agent Manager SDK
- Requirements
- What's New
- Known Issues

Introduction

ESX Agent Manager (EAM) SDK 8.0 Update 1 | 18 APR 2023

Check for additions and updates to these release notes.

About the vSphere Solutions Manager, vServices, and ESX Agent Manager SDK

With vSphere Solutions, vServices, and ESX Agents you can extend the functionality of vCenter Server.

- A vSphere Solution is an object or program that you create by extending specific classes in the vSphere Web Services API. After you register your solution with the instance of the vSphere Web Services API's ExtensionManager managed object associated with your vCenter Server, you see your new solution in the vCenter Server Extensions of your vSphere Client.
  - You can create a vService solution to provide access for a specific application to connect to a service across the network.
  - You can create an ESX Agent to extend the functions of an ESXi host and provide additional services that a vSphere solution requires.
- The vServices Manager and ESX Agent Manager are pre-built solutions that are part of the vCenter Server Extensions functionality within the vCenter Server.
- The VMware ESX Agent Manager includes an API library that allows you to adapt and modify your solutions and view them using the vSphere Client.

For more details about each of these extensibility options, see *Developing and Deploying vSphere Solutions, vServices, and ESXi Agents* for vSphere 8.0 Update 1.

Requirements

These release notes apply to the vSphere 8.0Update 1 release. You should install the following software to use the 8.0 Update 1 version of the vSphere Solutions Manager, vServices, and ESX Agent Manager SDK.

To use the vSphere Solutions Manager, vServices, and ESX Agent Manager SDKs, download and/or install:

- ESXi 8.0 Update 1 | 18 APR 2023
- vCenter Server 8.0 Update 1| 18 APR 2023

- com.vmware.eam.apiref.doc.jar | API Reference
- vsphere-vcenter-801-solutions-vservices-esxi-agent-manager-developer-guide.pdf | Programming Guide

What's New

This release of the vSphere Solutions Manager, vServices, and ESX Agent Manager SDKs introduces the following changes:

- The EAM Sample Solution is removed from the ESX Agent Manager SDK.

Known Issues

There are no issues.

vSphere Solutions Manager, vServices, and ESX Agent Manager 8.0 Release Notes

This document contains the following sections

- Introduction
- About the vSphere Solutions Manager, vServices, and ESX Agent Manager SDK
- Requirements
- What's New
- Resolved Issues
- Known Issues

Introduction

ESX Agent Manager (EAM) SDK 8.0 | 11 OCT 2022 Check for additions and updates to these release notes.

About the vSphere Solutions Manager, vServices, and ESX Agent Manager SDK

With vSphere Solutions, vServices, and ESX Agents you can extend the functionality of vCenter Server.

- A vSphere Solution is an object or program that you create by extending specific classes in the vSphere Web Services API. After you register your solution with the instance of the vSphere Web Services API's ExtensionManager managed object associated with your vCenter Server, you see your new solution in the vCenter Server Extensions of your vSphere Web Client.
  - You can create a vService solution to provide access for a specific application to connect to a service across the network.
  - You can create an ESX Agent to extend the functions of an ESXi host and provide additional services that a vSphere solution requires.
- The vServices Manager and ESX Agent Manager are pre-built solutions that are part of the vCenter Server Extensions functionality within the vCenter Server.
- The VMware ESX Agent Manager includes an API library that allows you to adapt and modify the ESX Agent Manager (EAM) Sample Solution and view it using the vSphere Web Client.

For more details about each of these extensibility options, see *Developing and Deploying vSphere Solutions, vServices, and ESXi Agents* for vSphere 8.0.

Requirements

These release notes apply to the vSphere 8.0 release. You should install the following software to use the 8.0 version of the vSphere Solutions Manager, vServices, and ESX Agent Manager SDK.

To use the vSphere Solutions Manager, vServices, and ESX Agent Manager SDKs, download and/or install:

- ESXi 8.0 | 11 OCT 2022
- vCenter Server 8.0 | 11 OCT 2022
- com.vmware.eam.apiref.doc.jar | API Reference
- vsphere-vcenter-80-solutions-vservices-esxi-agent-manager-developer-guide.pdf | Programming Guide

What's New

This release of the vSphere Solutions Manager, vServices, and ESX Agent Manager SDKs introduces the following changes:

- vSphere ESX Agent Manager is integrated with the vSphere Lifecycle Manager feature introduced with vSphere 7.0.

  If you use the vSphere Lifecycle Manager to remediate an ESXi cluster with a desired state specification, the vSphere ESX Agent Manager takes care of configuring the components that are part of the solutions deployed on the ESXi hosts in that cluster. When the desired state is applied on the cluster, the vSphere Lifecycle Manager installs the solution components configured by the ESX Agent Manager.

The EAM Sample Solution is compatible with Apache Tomcat ® 8.5.5 and later and Pivotal® tc Server 3.0 and later.

Resolved Issues

There are no issues.

Known Issues

There are no issues.

pyVmomi SDK 8.0 Release Notes

This document contains the following sections

- Introduction
- Maintenance Patch 1 for 8.0U3
- vSphere 8.0 Update 3
- Maintenance Patch 1 for 8.0U2
- vSphere 8.0U2 Release
- Maintenance Patch 2 for 8.0U1
- Maintenance Patch 1 for 8.0U1
- vSphere 8.0U1 Release
- Maintenance Patch 2 for 8.0
- Maintenance Patch 1 for 8.0
- vSphere 8.0 Release

Introduction

pyVmomi SDK

vSphere 8.0 Update 3 | 25 June 2024

Last document update 27 June 2024

Check for additions and updates to these release notes.

Maintenance Patch 1 for 8.0U3

Resolved Issues

The maintenance patch release resolved the following issue.

- Due to a dependency issue in the **setup.py** script, the greenfield installation of pyVmomi fails.

  This issue is resolved in this release. See the **e4fd849** commit.

vSphere 8.0 Update 3

What's New

With vSphere 8.0 Update 3, pyVmomi offers the following improvements:

- Support is added for the vSAN Health APIs.
- The bindings with support for the vSphere 8.0 Update 3 APIs are updated. For more information, see the vSphere Web Services API reference documentation.
- Support for the new vSphere Automation APIs provided with vSphere 8.0 Update 3 is included in the vSphere Automation SDK for Python.
- Type stubs are added for vSAN APIs.
- All type stubs are refactored.
- Stub adapters allow the usage of an existing session. See the 35f2743 commit.
- pyVmomi version is added in the **User-Agent** request header. See the 5ad215f commit.
- **certFile** and **certKeyFile** attributes are added to SoapAdapter.py. See the 2bbfb62 commit.
- Project wide variables are added to hold the current pyVmomi version - **version_info** and **version_info_str**. See the 5ad215f commit.
- The following functions are added to VmomiSupport.py to list all types - **ListManagedTypes()**, **ListDataTypes()**, and **ListEnumTypes()**. See the 9e303c1 and e43a287commits.
- The CHANGELOG file is updated.
- The dependency on **pywin32** is removed. See the 4bc1f52 commit.
- The non-remote ManagedObject and DataObject methods are documented. See the 472bdfc commit.
- The support statement now reflects the Broadcom support policy. See the c68913e commit.
- The copyright is updated. See the 10c3732 commit.

Compatibility

- As of vSphere 8.0 Update 3, the pyVmomi client is compatible with Python 3.12. For more information, see Python 3.12 compatibility (44d7b9f).

Deprecated

- **sso.SsoAuthenticator.get_bearer_saml_assertion_gss_api()** is removed. See the 11dc306 commit.
- **SSLTunnelConnection** is trimmed down to handle only tunnel connections. The code that handles remote proxy doubles the **HTTPProxyConnection** logic and therefore is removed. See the 44d7b9f commit.

Support Notices

Future pyVmomi releases will be compatible with the Supported versions of Python at the time of the release. The list of supported Python versions is documented in the Python Support section of the pyVmomi Readme file. With the next major pyVmomi release, pyVmomi will no longer support Python 2 and all versions earlier than 3.9.

In a future pyVmomi release, VMware intends to discontinue the following features:

- **pyVmomi**
  - All pyVmomi settings defined in the **pyVmomiSettings.py** will be removed - **allowGetSet**, **allowCapitalizedNames**,**binaryIsBytearray**, **legacyThumbprintException**.
  - The Feature.py script will be removed along with the pyVmomi feature states logic.
  - The **publicVersions** and **dottedVersions** version aliases will be removed from the **VmomiSupport.py** script. Instead, use the **ItsVersions** allias.

  - **pyVmomi.VmomiSupport.VmomiJSONEncoder** will be removed. Instead, use VmomiJSONEncoder.
  - **pyVmomi.VmomiSupport.templateOf()** will be removed. Instead, use VmomiJSONEncoder.templateOf().
  - **_legacyThumbprintException** in the SoapAdapter.py will be removed. Instead, use the **ThumbprintMismatchException** class in Security.py.
- **pyVim**
  - **b64token** and **mechanism** will be removed. Instead, use **token** and **tokenType**.
  - **__GetElementTreeFromUrl()** will be removed. Instead, use **__GetElementTree()**.
  - **OpenUrlWithBasicAuth()** will be removed.
  - **OpenPathWithStub()** will be removed.

Maintenance Patch 1 for 8.0U2

Released 17 Nov 2023

What's New

Added PyPI classifiers for Python 3.10 and Python 3.11. Support for both versions is verified.

Resolved Issues

Fixed: #978 and #1053 - Fix SmartConnect()'s handling of IPv6 address with square brackets

vSphere 8.0U2 Release

Released 28 Sep 2023

What' New

- Updated bindings and type hints with support for vSphere 8.0U2. Includes updates to VIM, PBM, EAM, SMS and VSLM namespaces. For details, refer “What’s New in vSphere API 8.0U2?” section in the API reference guide: https://developer.vmware.com/apis/1720/vsphere
- New features of vSphere 8.0U2 based on REST APIs are available via the vSphere Automation SDK for Python https://github.com/vmware/vsphere-automation-sdk-python

Maintenance Patch 2 for 8.0U1

Released 19 Jul 2023

What's New

Added bindings and type hints for Virtual Storage Lifecycle Management for vSphere 8.0U1 - VSLM namespace.

Resolved Issues

- Fixed: #1021 - Switch to static imports for type info modules
- Fixed: #1022 - Support proxy authentication
- Add custom HTTP headers support to connect.SmartStubAdapter()
- Missing filters are no longer treated as task failures
- Various small fixes for docstrings and linter checks
- Type hints fixes:
  - Fixed: #1026 - Use the "from Y import X as X" format to re-export submodules
  - Fixed: #1030 - Use a fully qualified name when the type is from another namespace/package
- Tests
  - **vcrpy** dependency is updated to the latest version with Python 2 support

  - **testtools** dependency is removed
  - Travis CI is no longer used
  - Various test updates and fixes. All tests are enabled.

Maintenance Patch 1 for 8.0U1

Released 25 May 2023

Resolved Issues

- Added missing VMODL1 classes to the type hints
- Enum values now match the letter case of the values from **typeinfo** files
- Fixed: #1115 - Syntax error in **vim/__init__pyi**
- Fixed: #1117 - Type stubs: Writable properties are marked as read-only
- Fixed: #1118 - Type stubs: Missing **vim.fault.*** and **vmodl.fault.*** types
- Fixed: #1119 - Type stubs: Enum fields should also accept **Literal[] str** type
- Fixed: #1120 - Type **stubs: Exception** types must inherit from **(Base)Exception**

vSphere 8.0U1 Release

Released 6 May 2023

What's New

- Updated bindings with support for vSphere 8.0U1. Includes updates to VIM, PBM, EAM and SMS namespaces. For details, refer “What’s New in vSphere API 8.0U1?” section in the API reference guide: https://developer.vmware.com/apis/1639/vsphere
- New features of vSphere 8.0U1 based on REST APIs are available via the vSphere Automation SDK for Python https://github.com/vmware/vsphere-automation-sdk-python
- Added support for type hints
- #892 - Allow passing ssl_context when login in with a token

Resolved Issues

- Fixed: #750 - Bumped vcrpy tests dependency to 2.1
- Fixed: #812 - Fix exception leaks
- Fixed: #1004 - Fix SmartConnect()'s SOAP and OAuth token login

Maintenance Patch 2 for 8.0

Released 18 Jan 2023

What's New

- This release restores the legacy behavior when retrieving managed properties and array of instances of ManagedObject

Resolved Issues

- Fixed: #995 - 8.0.0.1 breaks Ansible vmware_content_deploy_ovf_template folder lookup

Maintenance Patch 1 for 8.0

Released 16 Dec 2022

What's New

- Replace publicVersions with ltsVersions
- Updated VIM namespace for the next vSphere 8.0 patch release

Resolved Issues

- Fixed: #993 - WaitForTask broken on version >8.0.0
- Fixed: #994 - Pyvomi module failing in connect method

vSphere 8.0 Release

Released 24 Nov 2022

What's New

- **Bindings**
  - Updated bindings with support for vSphere 8.0. Includes updates to VIM, PBM, EAM, SMS and QUERY namespaces. For details, refer “What’s New in vSphere API 8.0.0.1?” section in the API reference guide: https://developer.vmware.com/apis/1355/vsphere
  - New features of vSphere 8.0 based on REST APIs are available via the vSphere Automation SDK for Python https://github.com/vmware/vsphere-automation-sdk-python
- **Breaking Changes**
  - Minimum Python 2 requirement is 2.7.9
  - **DynamicTypeManagerHelper.py** is removed
  - **ManagedMethodExecutorHelper.py** is removed
  - **connect.ConnectNoSSL()** and **connect.SmartConnectNoSSL()**are removed. Use **connect.Connect(disableSslCertValidation=True)** and **connect.SmartConnect(disableSslCertValidation=True)**
  - **VmomiSupport.UncallableManagedMethod** is renamed to **VmomiSupport.UnknownManagedMethod**
- **New Modules**
  - **Security.py** A new module is added to handle thumbprint verification. There is a predefined set of available crypto functions to verify the certificate thumbprints. Its possible to configure during runtime which of the available crypto functions are allowed.
  - **Feature.py**A new module related to pyVmomi development within VMware.
  - **VmomiJSONEncoder.py** The **VmomiJSONEncoder** is moved into a dedicated module.
- **Miscellaneous**
  - A new **binaryIsBytearray** setting is added to select the base type for the binary type. By default, the binary type is **str** for Python 2 and **bytes** for Python 3. If **binaryIsBytearray** is **True**, the binary type for Python 2 is set to **bytearray**. Required for **VmomiJSONEncoder** to work properly.
  - The license note is removed from the Python files. **LICENSE.txt** holds the Apache 2 license note.
  - pyVmomi now uses relative imports
  - Dependency on “requests” is removed
  - Added support for SAML token authentication
  - Added timeout for HTTP requests
  - Added option to set the maximum amount of time a task is allowed to run. On timeout, an exception is generated if **raiseOnError** is **True**.
  - Add option to get all updates for the task.
  - Add option to use a logger instead of the standard output
  - Various bug fixes
  - Code style improvements

Deprecated

- **connect.OpenUrlWithBasicAuth()**
- **connect.OpenPathWithStub()**

vSphere Client SDK

vSphere Client SDK 8.0 Update 3 Release Notes

This document contains the following sections

- Introduction
- About the vSphere Client SDK
- Changes and New Features
- Compatibility Notices
- Recently Resolved Issues
- Known Issues and Workarounds
- Use of /sdkTunnel Endpoint

Introduction

vSphere Client SDK 8.0 Update 3 | 25 June 2024 | Build on developer.broadcom.com

For VMware Cloud Foundation and vSphere

Check back for additions and updates to these release notes, marked **New**.

About the vSphere Client SDK

The vSphere Client SDK provides sample plug-ins, tools, libraries, and documentation to help developers build user interfaces with vSphere Client extensions for vSphere and VMware Cloud.

Changes and New Features

The vSphere Client SDK 8.0 U3 offers the following **New** features:

- **Extend the Snapshots tab of a virtual machine.**

Added a possibility for remote plug-ins to extend the "Snapshots" tab of a virtual machine in the same way as "Monitor" and "Configure" tabs. The extensibility can be achieved by specifying snapshots on a Virtual Machine object in the plugin.json file. For example:

```
"VirtualMachine": {
  "snapshots": {
    "views": [
      {
       ...view-information
      }
    ]
  }
}
```

- **Allow extensibility of Supervisor Namespace and Supervisor Cluster objects.**

Added a possibility for remote plug-ins to extend the Supervisor Namespace and Supervisor Cluster inventory objects. The extensibility can be achieved by specifying a "SupervisorNamespace" or "SupervisorClusterComputeResource" object in the plugin.json file. For example:

```
"objects": {
  "SupervisorNamespace": {
    "summary": {
      "view": {
        "uri": "myCard.html"
      }
    },
    "monitor": {
      "views": [
        {
          "uri": "myMonitor.html"
        }
      ]
    },
    "configure": {
      "views": [
        {
          "uri": "myConfigure.html"
        }
      ]
    },
    "menu": {
      "actions": [
        {
          "id": "myAction",
          "labelKey": "my.action.label",
          "icon": {
            "name": "action"
```

```
        },
        "trigger": {
          "type": "modal",
          "uri": "index.html#/my-action-modal",
          "titleKey": "my.action.modal.title",
          "size": {
            "width": 600,
            "height": 205
          }
        }
      }
    ]
  }
 }
}
```

- **Define plug-in card sizes.**

Added an option to choose the size of the plug-in defined summary cards. The options are "small", "medium", or "large" cards. The default (if no size is specified) is "medium". Size can be specified as in the plugin.json file below:

```
"VirtualMachine": {
  "summary": {
    "view": {
      "id": "vmCard",
      "uri": "index.html#/vm-card",
      "size": {
        "width": "small"
      }
    }
  }
}
```

- **More flexible recognition of certificate thumbprint formats.**

The Extension Registration tool now supports passing thumbprints in different formats. Until now, the only supported format of thumbprints in the Extension Registration tool was the colon-separated one (XX:XX:XX). The tool now supports passing thumbprints that are space-separated (XX XX XX) or ones that have no separation at all (XXXXXX).

- **Integrated Solution Installer improvements.**
  - Inject the associated vCenter Server(s) version in the deployed solution appliance. In addition to vCenter Servers count, FQDNs, thumbprints and GUIDs, the vCenter Server version will also be injected into the deployed solution appliance (solutionInstall.targetVcenterServers.<index>.version).
  - The /solutioninstall API endpoint is extended to allow passing vcenterExtensionsPerGuid in addition to vcenterExtensions. The new payload will indicate which extensions to register for a given vCenter Server GUID. Currently, vcenterExtensions only allows passing a list of extensions and the vSphere Client will register the extensions in all user-selected vCenter Servers. With the new payload vcenterExtensionsPerGuid, plug-in developers can specify a list of extensions for every vCenter Server.
    Either vcenterExtensions or vcenterExtensionsPerGuid can be used, but not both.
  - Inject the vSphere Client version in the deployed solution appliance. Since a new payload is introduced for the /solutioninstall API, plug-in developers need to know the version of the vSphere Client that they must call. This is why, in addition to solutionInstall.postDeployData.endpoint and solutionInstall.postDeployData.thumbprint, there is a new solutionInstall.postDeployData.version property.

For new features in the previous versions, see the *vSphere Client SDK 8.0.2 Release Notes* and before.

Compatibility Notices

Partners should migrate all local plug-ins to remote plug-ins as soon as possible. Local plug-ins for vSphere Client remain functional in the vSphere 8.0 U3 release, but they must be upgraded for compatibility with a number of security-related changes. Local plug-ins must become FIPS-compliant and must isolate their 3rd-party libraries.

Recently Resolved Issues

The following fixes are **New** in the 8.0 U3 release:

- **Multi Manifest validation.**

  Fixed an issue with the Multi Manifest feature where a plug-in may be reported as "incompatible" if it uses the Multi Manifest feature along with requirements for a specific vSphere Client or vCenter Server version. Now, requirements checking is part of the plug-in manifest validation. If a given plug-in manifest does not satisfy the environment requirements, it fails on the parsing stage and the next plug-in manifest in line will be tried to be deployed.
- **Remote Plug-in sample.**
  - Addressed an issue with web sockets.
  - Tomcat server has been updated to mitigate a security issue.
- **Integrated Solution Installer.**
  - All running solution install tasks are now canceled when the solution install task fails or times out.
  - Solution install default properties are removed from the wizard because they are not user configurable. Users will no longer see the default (non-configurable) solution install properties.
- **No verification for pinned certificates.**

  Hostname verification when downloading plug-ins is not needed and is no longer performed for self-signed plug-in server certificates. Using plug-in server certificates was introduced in the vSphere 8.0 U2, and the vSphere Client was wrongly doing hostname verification for pinned certificates.
- **Localization.**

  Fixed localization issues when showing or hiding plug-ins from Client Plug-ins view.
- **Accessibility.**

  Improved the accessibility of modal dialogs opened through the vSphere Client SDK.

- **Bundled libraries discontinued.**

  The vSphere Client SDK no longer bundles Spring and other libraries used by local plug-ins. After the local plug-ins isolation, local plug-ins are required to bundle their own libraries.

Known Issues and Workarounds

There are no unresolved issues known at this time.

Use of /sdkTunnel Endpoint

The vSphere Client 8.0 U2 release notes contained a section about proper use of the **/sdkTunnel** endpoint: calling **LoginExtensionByCertificate** to authenticate. Because **LoginExtensionByCertificate** is deprecated, you should consider the **/sdkTunnel** deprecated as well.

vSphere Client SDK 8.0 Update 2 Release Notes

This document contains the following sections

- Introduction
- About the vSphere Client SDK
- Changes and New Features
- Compatibility Notices
- Recently Resolved Issues
- Known Issues and Workarounds
- Avoid Use of /sdkTunnel Endpoint

Introduction

vSphere Client SDK 8.0.2 | 21 September 2023 | Find build on developer.vmware.com
For vSphere 8.0 Update 2 | Last document update 10 June 2024
Check back for additions and updates to these release notes, marked **New**.

About the vSphere Client SDK

The vSphere Client SDK provides sample plug-ins, tools, libraries, and documentation to help developers build user interfaces with vSphere Client extensions for vSphere and VMware Cloud.

Changes and New Features

The vSphere Client SDK 8.0 U2 offers the following **New** features:

- **Support for security certificates.**

  Plug-in servers can be registered with a security certificate in addition to, or instead of, a thumbprint (fingerprint). Using a certificate is recommended for greater security. The security certificate of a plug-in server is obtained with a web browser, saved as PEM in a certificate file, and specified by the **-serverCertificateFile** option of the registration script. You can find registration instructions in the 8.0 U2 developer guide.
- **Properties for vCenter Server injected into OVF.**

  Additional vCenter Server properties are injected into OVF. In previous releases, properties **token**, **endpoint**, and endpoint **thumbprint** were injected. As of 8.0 U2, properties for **count** of associated vCenter Servers, vCenter **hostname**, vCenter guid, and vCenter **thumbprint** are also injected.
- **Tasks console shows installation progress.**

Users can monitor progress of solution installation in the Tasks console, and cancel the solution install task if desired. Previously the Tasks console showed solution was installing but not installation progress.

- **Number of defined cards increased from 3 to 5.**

  In vSphere 8.0 U1, it became possible for a remote plug-in to create multiple cards. In this release, the limit was raised from three to five.
- **File vim25.jar no longer included in SDK.**

  Programmers without **vim25.jar** in their development environment must obtain this JAR file from vSphere Management SDK (under Web Services).

For new features in the previous versions, see the *vSphere Client SDK 8.0.1 Release Notes* and before.

Compatibility Notices

Local plug-ins for vSphere Client remain functional in the vSphere 8.0 release, but they must be upgraded for compatibility with a number of security-related changes. Local plug-ins must become FIPS-compliant and must isolate their 3rd-party libraries. Partners should migrate all local plug-ins to remote plug-ins as soon as possible.

Recently Resolved Issues

- **Multi-manifest no longer requires plugin.json.**

  It's no longer necessary to create a file specifically named **plugin.json** for multi-manifest support.

Known Issues and Workarounds

There are no unresolved issues known at this time.

Avoid Use of /sdkTunnel Endpoint

When vCenter extensions or vSphere clients make connections to the **/sdk** endpoint, it is on port 443 over HTTPS using TLS to the Envoy proxy, which quickly passes HTTP1 or HTTP2 connections to **vpxd** for servicing. The current limit is 2K simultaneous connections. Another endpoint **/sdkTunnel** is largely undocumented. Limited to 1K connections, it is on open port 80 also over HTTPS but the client must first connect with plain HTTP, whereas for **/sdk** the client initiates TLS directly. Rather than being mediated by the proxy, packets go directly to **vpxd**, authenticated with mutual TLS (mTLS) so the server can ensure that the client owns the respective private key, on top of TLS where the client validates the server. By design, there is no **/sdkTunnel** for port 443.

Before deprecation of **LoginExtensionByCertificate** in vSphere 6.0, direct connection was useful for exchanging security credentials, but because the Envoy proxy cannot see inside TCP packets on **/sdkTunnel**, it was unable to detect and close zombie connections. So when there were more than 1K simultaneous connections, and the client neglected to close any of them, the Envoy proxy queued requests, the **/sdkTunnel** endpoint clogged, and long service delays resulted.

Use of **/sdkTunnel** to authenticate by calling **LoginExtensionByCertificate** should be discontinued, and replaced with **LoginByToken**. Going forwared vSphere clients should instead use the **/sdk** endpoint, which is far less prone to congestion and service delays. You can find sample code that calls **LoginByToken** in topic “Using LoginByToken in Java” of the *vSphere Web Services SDK Programming Guide*.

vSphere Client SDK 8.0 Update 1 Release Notes

This document contains the following sections

- Introduction
- About the vSphere Client SDK
- Changes and New Features
- Compatibility Notices
- Recently Resolved Issues
- Known Issues and Workarounds

Introduction

vSphere Client SDK 8.0.1 | 18 April 2023 | vCenter Server, build on developer.vmware.com

For vSphere 8.0 Update 1 release | Last document update 17 April 2023

Check back for additions and updates to these release notes, marked **New**.

About the vSphere Client SDK

The vSphere Client SDK provides sample plug-ins, tools, libraries, and documentation to help developers build user interfaces with vSphere Client extensions for vSphere and VMware Cloud.

Changes and New Features

The vSphere Client SDK 8.0 U1 offers the following **New:** features:

- **Ability to define action separators.**

Plug-ins are now able to define separators between their menu actions. This can be achieved by specifying a new object {"type":"separator"} between the plug-in defined actions. For example:

```
"actions": [
  {
    "id": "myAction1",
    "labelKey": "vm.action.label1",
    "icon": {
      "name": "action-vm"
    },
    "trigger": {
      "type": "modal",
      "uri": "index.html#/vm-action-modal",
      "titleKey": "vm.action.modal.title",
      "size": {
        "width": 600,
        "height": 205
      }
    }
  },
  {
    "type": "separator"
  },
```

```
  {
    "labelKey": "vm.action.label2",
    "icon": {
      "name": "action-vm"
    },
    "trigger": {
      "type": "modal",
      "uri": "index.html#/vm-action-modal",
      "titleKey": "vm.action.modal.title",
      "size": {
        "width": 600,
        "height": 205
      }
    }
  },
]
```

- **Ability to define multiple summary cards.**

Plug-ins are now able to define up to 3 cards in a given vSphere object's summary view. In addition, each card can have its own icon and title. If no icon or title are provided, the default plugin icon and name will be used. This can be achieved by specifying an array of views instead of a single view in a given object's summary section. For example:

```
"summary": {
  "views": [
    {
      "id": "card1Id",
      "uri": "resources/card1.html"
    },
    {
      "id": "card2Id",
      "uri": "resources/card2.html",
      "icon": {
        "name": "card2Icon"
      },
```

```
        "titleKey": "card2Title"
      }
    ]
}
```

- **Integrated Solution Installation**

Solution installer with ability to specify required vCenter Server version support. Integrated solutions can now specify which vCenter Server version they support. This will prevent users from installing the solution on a vCenter Server with lower than the specified version. To achieve this, the vmw.vsphereui.solutioninstall.requiredVcVersion property must be specified in the solution's OVF. For example:

```
<Property ovf:qualifiers="ValueMap{"8.0.1"}"
        ovf:userConfigurable="true"
        ovf:value="8.0.1"
        ovf:type="string"
        ovf:key="vmw.vsphereui.solutioninstall.requiredVcVersion">
  <Label>Required vCenter Server version</Label>
  <Description/>
</Property>
```

- **Remote Plug-in Sample**

Upgraded Angular and Clarity versions for improved security.

- **Tools**

The extension-registration tool now accepts a list of events. This means that plug-in registration may include a list of plug-in defined events.

For new features in the previous versions, see the *vSphere Client SDK 8.0 Release Notes* and before.

Compatibility Notices

Local plug-ins for vSphere Client remain functional in the vSphere 8.0 release, but they must be upgraded for compatibility with a number of security-related changes. Local plug-ins must become FIPS-compliant and must isolate their 3rd-party libraries. Partners should migrate all local plug-ins to remote plug-ins as soon as possible.

Recently Resolved Issues

The vSphere Client SDK 8.0.1 release resolves the following issue.

- Actions are now ordered in the way they are written in the plugin.json.
- Fixed an issue where a plug-in may be shown as *Partially Deployed*.
- The plug-in action menus are properly live refreshed when the user *shows/hides* a plug-in.
- Fixed an issue where an *Incompatible* plug-in could be shown as *Failed*.
- The htmlClientSdk is no longer cached by the browser and does not break plug-in upgrades.
- Fixed *Confirmation Modal* layout issues.
- Plug-in deployment will no longer fail if it is using an unknown locale. This means that adding a new supported language to the vSphere Client and the plug-in will not break the plug-in's compatibility with older vSphere Client versions.

- Fixed an issue where an *Incompatible* plug-in could not be redeployed.

Known Issues and Workarounds

These are unresolved issues known at this time.

- **Multiple Manifest Issue.**

The *Multi Manifest* feature requires a default manifest called plugin.json. If a plug-in is using the multi manifest feature, it is required that at least one of the plug-in manifests is named plugin.json. It is recommended that the least desired plug-in manifest is named plugin.json and placed at the bottom of the array of manifests. For example:

```
{
  "manifestVersion": "1.0.0",
  "manifests": [
    "plugin-80.json",
    "plugin-70.json"
    "plugin.json"
  ]
}
```

vSphere Client SDK 8.0 Release Notes

This document contains the following sections

- Introduction
- Notice of Local Plug-in Deprecation
- What's New
- Resolved Issues
- Known Issues

Introduction

vSphere Client SDK 8.0 | 11 OCT 2022

Check for additions and updates to these release notes.

Notice of Local Plug-in Deprecation

Local plug-ins are being deprecated with the 8.0 release. VMware plans to discontinue the support for local plug-ins with the next major release.

The certification program has been updated and local plug-ins will no longer be certified.

What's New

This release of the vSphere Client and the SDK comes with an improved Plug-in Administration view, Integrated Solution Installer for the vSphere Client, new capabilities, enhanced features and bug fixes.

- **vSphere Client plug-ins administration view overhaul**

There has been a significant change in the way the client plug-ins are displayed in the vSphere Client. A more detailed plug-in topology is shown, making it easy to verify which vCenter Server a particular plug-in is registered with, and which plug-in servers are in use. The enhanced view allows actions to be applied to multiple plug-ins and to directly navigate to a given plug-in global view.

To check out the new Client Plug-ins view functionality, navigate in the vSphere Client to:

Administration > Client Plugins

- **Integrated Solution Installer for the vSphere Client**

Partners now have the ability to integrate their solutions with the Solution Installer in the vSphere Client. This will simplify the solution's installation flow and will greatly reduce the need for the user to visit the solution's standalone UI for configuration and plug-in registration operations.

Sample files to help with the integration process can be found in the vSphere Client SDK deliverable under /samples/solutioninstall/

- **Navigating actions defined in the remote plug-in manifest file**

Remote plug-in developers are now able to define navigating actions along with the modal menu actions in the plug-in manifest file. This will enable users to use the action to navigate to a view defined by the plug-in.

- **Newly promoted core client views for remote plug-in navigation**

The list of views to which a remote plug-in can navigate with the navigateToClientView() JavaScript API has been extended with the following views:

- cluster.configure.settings.drs
- cluster.configure.settings.ha
- cluster.configure.settings.quickstart
- folder.configure.keyProviders
- folder.configure.settings.general
- host.configure.networking.physicalAdapters
- host.summary
- namespace.storage.persistentVolumeClaims
- network.summary
- profilesAndPolicies.vmStoragePolicies
- vm.monitor.performance.overview
- vm.summary
- workload-platform.supervisor.summary

The full list of navigable views can be found in the Javascript-API-Remote.html

- **Sharing modals between remote plug-ins**

Remote plug-ins can create more distributed content sharing. Modals defined by one remote plug-in can be reused by a different remote plug-in.

For more information, see Javascript-API-Remote.html

- **Refreshing remote plug-in related items**

Remote plug-ins can refresh their related items by using the new JavaScript API - refreshPluginItemsState(). A related item of a plug-in may be a monitor/configure item, an action or a summary view.

- **Support for Italian localization**

The vSphere Client SDK now allows localization of plug-ins in the Italian language.

- **Support dynamic object cards**

In addition to dynamic monitor, configure and action extensions, remote plug-in developers are able to define dynamic cards. This means that a particular plug-in card in an object's summary view can be shown or hidden based on a chosen condition.

- **Support for additional plug-in servers for dynamic extensions**

Developers can now specify a server endpoint for dynamic extension queries. Formerly, the manifest server was required to handle all dynamic extension queries.

- **Remote Plug-in Sample updates**
- New library versions: Clarity Design System 3.0.0 and Angular 9.0.0.
- The sample demonstrates the use of dynamic cards.

The sample demonstrates improved security by using a more robust trust strategy.

Resolved Issues

- **Open a modal with a recursive object**

The **htmlClientSdk.modal.open()** method throws an error when a configuration object with recursive references is passed. A new flag **sanitizeData** was introduced indicating whether the passed modal configuration data should be serialized/deserialized, or not. The default value of the flag is true (i.e., do serialization), which is the previous behavior. Set the flag to false if you pass objects containing recursive references.

- **Modified the Remote Plug-in Sample to use a more secure trust strategy**

Previous versions of the sample used a "trust-all" strategy, which was insecure. This version correctly verifies server thumbprints.

- **Plug-in manifest conversion tool is modified to support both light and dark themes**

The tool is modified to generate two sprite sheets, one for the light theme and one for the dark theme. The sprite sheets now contain placeholder icons which should be replaced by the plug-in developers with real icons.

- **Fixed a number of layout issues with the plug-in modal dialogs**

Fixed the layout of the modal dialogs opened through the vSphere Client SDK JavaScript APIs to eliminate issues previously reported.

Known Issues

None

Virtual Disk Development Kit

Virtual Disk Development Kit 8.0 Update 3 Release Notes

This document contains the following sections

- Introduction
- About the Virtual Disk Development Kit
- Changes and New Features
- Compatibility Notices
- Recently Resolved Issues
- Known Issues and Workarounds

Introduction

VDDK 8.0.3 with ESXi 8.0 Update 3 | 25 June 2024 GA | VDDK build 23950268

Table continued on next page

Continued from previous page

For VMware Cloud Foundation and vSphere

Check back for additions and updates to these release notes, marked **New**.

About the Virtual Disk Development Kit

The Virtual Disk Development Kit (VDDK) 8.0.3 is an update to support vSphere 8.0 Update 3 and respond to partner requests for enhancement.

VDDK is used with vSphere Storage APIs for Data Protection (VADP) to develop backup and restore software. For general information about this development kit, how to obtain the software, programming details, and redistribution, see the VDDK landing page on *developer.vmware.com*.

Changes and New Features

VDDK 8.0.3 offers improved support for NVMe over Fabrics:

- **New: VDDK supports SAN transport with NVMe over Fabrics.**

  vSphere 8 contained many enhancements regarding NVMe over Fabrics (NVMe-oF), such as more namespaces, extended reservations, discovery services, and vVol support. Interoperability issues were discovered with SAN transport, so backup applications used NBD/NBDSSL or HotAdd transport to back up NVME-oF datastores. As of 8.0 U3, VDDK also supports SAN transport for NVME-oF backup on a Linux backup proxy.
- **New: Documentation for backing up VM Service managed VM.**

  For instructions to backup and restore virtual machines managed by VM Service, a VMware Tanzu facility to simplify DevOps programming on Kubernetes, see chapter “Practical Programming Tasks” in the *VDDK Programming Guide*. This is a documentation-only feature; no related code changes were made to VDDK.
- **New: Open source components.**

  The following open source components were upgraded: curl 8.4.0 to 8.5.0, expat 2.5.0 to 2.6.2, libuv to 1.43.0, openssl 3.0.12 to 3.0.13, openssl FIPS 3.0.8 to 3.0.9, and zlib 1.3 to 1.3.1.

VDDK 8.0.2 offered the following features:

- **CBT troubleshooting procedure.**

  Steps for customers to troubleshoot failures in changed block tracking (CBT) were expanded and moved to chapter 10 of the *VDDK Programming Guide*.
- **Network resiliency.**

  Rather than return Error 13 “cannot open file” or “file read failed” after network connection errors, `VixDiskLib_Open` now returns the underlying network error, such as `VIX_E_HOST_SERVER_SHUTDOWN` for machine powered off, or `VIX_E_HOST_NETWORK_CONN_REFUSED` for network dysfunction. This allows backup applications to better handle network errors and react to them by retrying file open and read.
- **TKGS support for container snapshots.**

  Container Storage Interface added support for container snapshots in Tanzu Kubernetes Grid Service (TKGS). For details, see section “Container Storage Snapshot” in the *VDDK Programming Guide*.
- **Telemetry enhancement.**

  VDDK “phone home” operates more efficiently. Changes were transparent to backup applications.

For new features in previous VDDK versions, see the *VDDK 8.0 U1 Release Notes* and before

Compatibility Notices

New: Microsoft ReFS not supported. All mentions of Resilient File System (ReFS) were removed from the 8.0.3 *VDDK Programming Guide*. VDDK 7.0 release notes stated that ReFS writes do not work with VixMntapi, but VMware recently discovered that ReFS mount does not work either.

VDDK 8.0 U3 was tested with the following systems for HotAdd proxy backup. Backup vendors may support additional operating systems for proxy backup. See the *VMware Compatibility Guide* for a complete list of guest operating systems usable on ESXi hosts.

- Windows Server 2022
- CentOS stream 9
- Oracle Linux 8.9
- Red Hat Enterprise Linux (RHEL) 9.1, 9.2, 9.3
- SUSE Linux Enterprise Server (SLES) 12 SP5, 15 SP5
- Photon v4, v5

This table shows recommended VDDK versions for various VMC milestones:

VMC Milestone	Compatible VDDK Versions
1.16 – 1.19	6.7.x, 7.0.x, 8.0.x
1.20, 1.22, 1.24, 1.26	7.0.x, 8.0.x

The VMware policy concerning backward and forward compatibility is for VDDK to support N-2 and N+1 releases. In other words, VDDK 8.0 and all its update releases work with vSphere 6.7, 7.0.x, and (except for new features) the next major release. Note however that vSphere 6.7 reached End of Technical Guidance (EoTG) in November 2023.

Here are possible issues with backward and forward compatibility.

- **More difficult configuration of NFC logs.**

  When customers want to change the NFC server log level to help diagnose NBD backup issues, they must do so with JSON edits of Config Store settings. For instructions, see “How to Change NFC Logging Level” in the *VDDK 8.0.2 Release Notes*.
- **Best practice for restore with pre-existing snapshot.**

  When a backed-up VM had snapshots, disaster recovery software can restore a fresh VM with its pre-existing snapshots in place. However for partial file recovery, or to recover a VM back to a point in time, it is difficult to determine how to restore a VM that has an existing snapshot. Should parts of the snapshot be restored to a previous state (if applicable), or only the VM by ignoring the snapshot? Certain backup-restore applications do not handle this situation well; snapshot contention causes a GenericVmConfigFault. One good solution is for customers to delete any pre-existing snapshots before recovery, especially if not needed. This is mandatory for recovery with SAN mode transport, anyway. Restore applications could refuse to recover a VM until any pre-existing snapshot is deleted. If not, a customer work-around is to recover the VM to a different location, avoiding the issue, then retain the recovered VM and delete the VM with pre-existing snapshot

Recently Resolved Issues

The VDDK 8.0.3 release resolves the following issues.

- **New: NBD transport on separate vmk network adapter.**

  When customers attempted backup using NBD transport on an alternate network adapter, for example vmk2 instead of vmk0, NBD network traffic did not bind to the separate vmk when adapters were on the same subnet. It worked only when the network adapters were on different subnets. In this release, the backup traffic always binds to the vmk with “vSphere Backup NFC” service.
- **New: Quiesced snapshot support for a Windows VM with NVMe disks.**

VMTools 12.4.0 and later support quiesced snapshot for Windows VMs with NVMe disks. However due to issues with the old Windows NVMe driver, application-consistent snapshot with NVMe disks is supported only for Windows Server 2022 build 1906 and later.

- **New: Documentation correction regarding NFC server timeouts.**

  Settings `vixDiskLib.nfcFssrvr.TimeoutMs` and `vixDiskLib.nfcFssrvrWrite.TimeoutMs` are not used in VDDK, so references to them were removed from the 8.0 U3 *VDDK Programming Guide*.

VDDK 8.0.1 and 8.0.2 resolved the following issues.

- **Customers complained about nonconfigurable crash dumps.**

  On Windows, `vmacore` crash dumps now appear in the VDDK configured `tmpDir`. On Linux, crash dumps appear as configured in the kernel, like they did before.
- **Open SSL library upgraded for FIPS support.**

  The Open SSL library `openssl` was upgraded to version 3.0 for vSphere 8.0 U1. To enable FIPS with Open SSL 3.0, see section “How to Enable FIPS on VDDK Proxy VM” in the *VDDK 8.0.1 Release Notes*.
- **Reminder about CBT Check in VDDK 7.0.**

  VDDK 7.0 included `vixDiskCheck` utility with `cbtCheck` option, and now `readPerf` and `writePerf` options. For details, find the VDDK `bin64` folder and run `vixDiskCheck` for help on all its options.

For resolved issued in previous VDDK versions, see the *VDDK 8.0 Release Notes* and before

Known Issues and Workarounds

These are unresolved issues known at this time.

- **New: Non-quiesced snapshot on Linux VM.**

  After a successful quiesced snapshot of a Linux guest, the snapshot may be reported as not quiesced. With VMware Tools versions older than 10.2.0, or ESXi versions older than 6.7, taking a quiesced snapshot on a Linux VM succeeds but appears as a non-quiesced snapshot, even though the snapshot is actually quiesced. A snapshot is identified as quiesced only if a backup manifest file was provided for the snapshot. This works on a Windows VM with VSS plugin, but is not accurate on a Linux VM using the sync driver to quiesce snapshots. To fix this, customers can upgrade `vmtools` to 10.2.0 or later, and ESXi to 6.7 or later, so it appears as a File-System quiesced snapshot. {{2607088 2601331}}
- **Clean-up function not working with default tmpDirectory.**

  With a default temporary directory, the folder name can change from backup to backup, if the backup job is a new OS process, so the `VixDiskLib_Cleanup` function might not clean up everything left behind. To avoid this, programs can set `tmpDirectory` in the configuration file for `VixDiskLib_InitEx`.

Virtual Disk Development Kit 8.0 Update 2 Patch 1 Release Notes

This document contains the following sections

- Introduction
- About the Virtual Disk Development Kit
- Changes and New Features
- Compatibility Notices
- How to Change NFC Logging Level
- Recently Resolved Issues
- Known Issues and Workarounds

Introduction

VDDK 8.0.2.1 for vSphere 8.0 U2 | 7 February 2024 | VDDK build number 23124249

VMware Cloud Foundation and vSphere Foundation | Last document update 01 April 2024

Check back for additions and updates to these release notes, marked **New**.

About the Virtual Disk Development Kit

The Virtual Disk Development Kit (VDDK) 8.0.2 was an update to support vSphere 8.0 Update 2 and respond to partner requests for enhancement. VDDK 8.0.2.1 is an asynchronous release with security and bug fixes.

The VMware policy concerning backward and forward compatibility is for VDDK to support N-2 and N+1 releases. In other words, VDDK 8.0 and all its update releases will support vSphere 6.7, 7.0.x, and (except for new features) the next major release.

VDDK is used with vSphere Storage APIs for Data Protection (VADP) to develop backup and restore software. For general information about this development kit, how to obtain the software, programming details, and redistribution, see the VDDK landing page on developer.vmware.com.

Changes and New Features

VDDK 8.0.2.1 offers the following **New** features:

- **Component upgrades and security fixes.**

  The following open source components were upgraded: `libcurl` to 8.4.0, `libxml2` to 2.11.5, `zlib` to 1.3, and `openssl` to 3.0.12. Upgrade of the Curl library fixes vulnerability CVE-2023-38545.
- **Confusing error message when HotAdd finds no free slots.**

  When there were no free SCSI controller slots for HotAdd transport mode, the reported error was “You do not have access rights to this file.” In this release, more detail follows that error: “*Failed to find free slots for HotAdd.*”

VDDK 8.0.2 offered the following features:

- **CBT troubleshooting procedure.**

  Steps for customers to troubleshoot failures in changed block tracking (CBT) were expanded and moved to chapter 10 of the *VDDK Programming Guide*.
- **Network resiliency.**

  Rather than return Error 13 “cannot open file” or “file read failed” after network connection errors, `VixDiskLib_Open` now returns the underlying network error, such as `VIX_E_HOST_SERVER_SHUTDOWN` for machine powered off, or `VIX_E_HOST_NETWORK_CONN_REFUSED` for network dysfunction. This allows backup applications to better handle network errors and react to them by retrying file open and read.
- **TKGS support for container snapshots.**

  In this release, Container Storage Interface added support for container snapshots in Tanzu Kubernetes Grid Service (TKGS). For details, see section “Container Storage Snapshot” on pages 85-86 of the *VDDK Programming Guide*.
- **Telemetry enhancement.**

  VDDK “phone home” operates more efficiently in this release. Changes are transparent to backup applications.
- **Quiesced snapshot for Windows VM with NVMe disks.**

  VMware Tools 12.4.0 is scheduled for release in spring 2024. It will support application quiesced snapshots for a Windows guest (with NVMe HotAdd issue fixed) running Server 2022 build 1906 or later, on virtual hardware version 21 or later. Failing these requirements, fallback is to filesystem quiescing.

Compatibility Notices

More difficult configuration of NFC logs. When customers want to change the NFC server log level to help diagnose NBD backup issues, they must do so with JSON edits of Config Store settings. See How to Change NFC Logging Level below.

VDDK 8.0 U2 supports the following operating systems for proxy backup (mostly the same as VDDK 8.0):

- Windows Server 2022, 2019, and 2016
- Red Hat Enterprise Linux (RHEL) 7.9, 8.1, 8.2, 8.6, 9.0
- **New:** Red Hat Enterprise Linux (RHEL) 8.8
- CentOS 7
- CentOS 8.5
- Oracle Linux 8.7
- SUSE Linux Enterprise Server (SLES) 12 SP5 and 15.1
- Ubuntu 18.04
- Photon v3, v4

This table shows recommended VDDK versions for various VMC milestones:

VMC Milestone	Compatible VDDK Versions
1.16 – 1.19	6.7.x, 7.0.x, 8.0.x
1.20, 1.22, 1.24	7.0.x, 8.0.x

Here is a possible issue with backward and forward compatibility.

Best practice for restore with pre-existing snapshot. When a backed-up VM had snapshots, disaster recovery software can restore a fresh VM with its pre-existing snapshots in place. However for partial file recovery, or to recover a VM back to a point in time, it is difficult to determine how to restore a VM that has an existing snapshot. Should parts of the snapshot be restored to a previous state (if applicable), or only the VM by ignoring the snapshot? Certain backup-restore applications do not handle this situation well; snapshot contention causes a GenericVmConfigFault. One good solution is for customers to delete any pre-existing snapshots before recovery, especially if not needed. This is mandatory for SAN transport recovery, anyway. Restore applications could refuse to recover a VM until any pre-existing snapshot is deleted. If not, a customer work-around is to recover the VM to a different location, avoiding the issue, then retain the recovered VM and delete the VM with pre-existing snapshot.

How to Change NFC Logging Level

For security reasons ESXi 8.0 U2 and VDDK 8.0.2 make it difficult for NFC clients to change the NFC server log level. Now it must be done with JSON edits of Config Store. Because NFC log level set in the VDDK configuration file accounts for changes in various vSphere releases, customers are not affected until they run vSphere 8.0 U2.

For troubleshooting NBD(SSL) backups, configuration of NFC logging changed in almost every vSphere release. Here is a summary:

- In vSphere 7.0 and before, NFC log level was set in the **vpxa.cfg** file.
- In vSphere 7.0 U1, NFC logs are created by hostd rather than vpxa.
- In vSphere 7.0 U2, configuration of NFC log level uses Config Store, set with JSON.
- VDDK 8.0.2 and vSphere 8.0 U2 do not allow NFC clients to change the NFC server log level.

Summarizing the various combinations of ESXi and VDDK:

With ESXi 8.0 U1 and earlier paired with VDDK 8.0.1 and earlier, users can still use `vixDiskLib.nfc.LogLevel` in the VDDK configuration file to set the NFC server log level.

With ESXi 7.0 and earlier paired with VDDK 8.0.2, users must change the vpxa configuration file and restart vpxa.

1. ssh to the ESXi host
2. Edit */etc/vmware/vpxa/vpxa.cfg*

```
<nfc>
  <loglevel>debug</loglevel>
</nrc>
```

3. Run command: /etc/init.d/vpxa restart

With ESXi 7.0 U1 paired with VDDK 8.0.2, users must change the hostd log level and restart hostd.

1. ssh to the ESXi host
2. Run command: `vim-cmd hostsvc/advopt/update Config.HostAgent.log.level string trivia`
3. Run command: `/etc/init.d/hostd restart`

With ESXi 7.0 U2 and later paired with VDDK 8.0.2, users must create a temporary file to change settings in Config Store and restart hostd.

1. ssh to the ESXi host
2. Run command: `configstorecli config current get -c esx -g services -k hostd > /tmp/cs.json`
3. Edit /tmp/cs.json

```
"nfcsvc": {
 "log_level": "TRIVIA",
 "max_memory": 100663296,
 "max_stream_memory": 35651584
},
```

4. Run command: `configstorecli config current set -c esx -g services -k hostd -infile /tmp/cs.json`
5. Run command:`/etc/init.d/hostd restart`

With ESXi 8.0 U2 and any VDDK release, users must follow the JSON steps above.

Recently Resolved Issues

The VDDK 8.0.1 release resolved the following issues.

- **Customers complained about nonconfigurable crash dumps.**

  On Windows, vmacore crash dumps now appear in the VDDK configured tmpDir. On Linux, crash dumps appear as configured in the kernel, as they did before.
- **Open SSL library upgraded for FIPS support.**

  The Open SSL library openssl is upgraded to version 3.0 for vSphere 8.0 U1. To enable FIPS with Open SSL 3.0, see below.
- **Reminder about CBT Check in VDDK 7.0.**

VDDK 7.0 included vixDiskCheck utility with cbtCheck option, and now readPerf and writePerf options. For details, find the VDDK bin64 folder and run vixDiskCheck for help on all its options.

For resolved issued in previous VDDK versions, see the *VDDK 8.0 Release Notes* and before.

Known Issues and Workarounds

These are unresolved issues known at this time.

- **New: Non-quiesced snapshot on Linux VM.**

  After a successful quiesced snapshot of a Linux guest, the snapshot may be reported as not quiesced. With VMware Tools versions older than 10.2.0, or ESXi versions older than 6.7, taking a quiesced snapshot on a Linux VM succeeds but appears as a non-quiesced snapshot, even though the snapshot is actually quiesced. A snapshot is identified as quiesced only if a backup manifest file was provided for the snapshot. This works on a Windows VM with VSS plugin, but is not accurate on a Linux VM using the sync driver to quiesce snapshots. To fix this, customers can upgrade vmtools to 10.2.0 or later, and ESXi to 6.7 or later, so it appears as a File-System quiesced snapshot.
- **VDDK non-support of NVMe over Fabrics.**

  vSphere 8 contains many enhancements regarding NVMe over Fabrics (NVMe-oF) such as more namespaces, extend reservations, discovery services, and vVol support. However it seems unlikely that VDDK handles backup for all these new features. More information will be forthcoming. Meanwhile applications can use NBD/NBDSSL and HotAdd transport to back up NVME-oF datastores.
- **Clean-up function not working with default tmpDirectory.**

  With a default temporary directory, the folder name can change from backup to backup, if the backup job is a new OS process, so the `VixDiskLib_Cleanup` function might not clean up everything left behind. To avoid this, programs should set `tmpDirectory` in the configuration file.

Virtual Disk Development Kit 8.0 Update 2 Release Notes

This document contains the following sections

- Introduction
- About the Virtual Disk Development Kit
- Changes and New Features
- Compatibility Notices
- How to Change NFC Logging Level
- Recently Resolved Issues
- Known Issues and Workarounds

Introduction

VDDK 8.0.2 with vSphere 8.0 U2 | 21 September 2023 | VDDK build 22388865

For ESXi and vCenter Server GA | Last document update 01 April 2024

Check back for additions and updates to these release notes, marked **New**.

About the Virtual Disk Development Kit

The Virtual Disk Development Kit (VDDK) 8.0.2 is an update to support vSphere 8.0 Update 2 and respond to partner requests for enhancement.

The VMware policy concerning backward and forward compatibility is for VDDK to support N-2 and N+1 releases. In other words, VDDK 8.0 and all its update releases will support vSphere 6.7, 7.0.x, and (except for new features) the next major release.

VDDK is used with vSphere Storage APIs for Data Protection (VADP) to develop backup and restore software. For general information about this development kit, how to obtain the software, programming details, and redistribution, see the VDDK landing page on developer.vmware.com.

Changes and New Features

VDDK 8.0.2 offers the following features:

- **New: CBT troubleshooting procedure.**

  Steps for customers to troubleshoot failures in changed block tracking (CBT) were expanded and moved to chapter 10 of the *VDDK Programming Guide*.
- **New: Network resiliency.**

  Rather than return Error 13 “cannot open file” or “file read failed” after network connection errors, VixDiskLib_Open now returns the underlying network error, such as VIX_E_HOST_SERVER_SHUTDOWN for machine powered off, or VIX_E_HOST_NETWORK_CONN_REFUSED for network dysfunction. This allows backup applications to better handle network errors and react to them by retrying file open and read.
- **New: TKGS support for container snapshots.**

  In this release, Container Storage Interface added support for container snapshots in Tanzu Kubernetes Grid Service (TKGS). For details, see section “Container Storage Snapshot” on pages 85-86 of the *VDDK Programming Guide*.
- **New: Telemetry enhancement.**

  VDDK “phone home” operates more efficiently in this release. Changes are transparent to backup applications.
- **Quiesced snapshot for Windows VM with NVMe disks.**

  VMware Tools 12.4.0 is scheduled for release in spring 2024. It will support application quiesced snapshots for a Windows guest (with NVMe HotAdd issue fixed) running Server 2022 build 1906 or later, on virtual hardware version 21 or later. Failing these requirements, fallback is to filesystem quiescing.

VDDK 8.0.1 offered the following features:

- **Datastore-level access to VMDK.**

  In this release, it is possible to directly access virtual disks on a datastore. Programs do this by specifying VixDiskLibSpecType as Datastore. At the datastore level, there is no support for snapshots and CBT. See below for steps to employ this feature. Currently only NBD/NBDSSL and HotAdd transport modes are supported. Note that HotAdd transport does not support RDM disks. For datacenters with encrypted disks, NBDSSL transport mode is required, and clusters must activate vSphere HA to guarantee that all hosts have a crypto key ID.
- **Wildcards in allow and deny lists for SAN.**

  After configuring allowList and denyList for SAN transport mode, customers and partners requested a wildcard mechanism for ease of use. See section “Initialize Virtual Disk API” in the *VDDK Programming Guide* for details. VDDK 8.0.1 supports a new wildcard mechanism to shorten list making. You can use meta character expressions * ? [xyz] [a-z] [0-9] with the same meanings as in Bash shell.
- **Cache file for SAN disk device discovery.**

  Before doing a SAN mode backup, each instance of VDDK scans disks to determine the device name and VMFS ID of the LUN. In large datacenters with many SAN based disks, this has a significant negative performance impact. Partners requested a cache mechanism to reduce frequency of scans. As implemented, cache file scsiDiskList.json is created in the tmpDirectory specified by the VDDK configuration file. When VDDK fails to open a disk with cached LUN path, because disk was added or deleted since last scan, LUNs are rescanned and the JSON file overwritten. This cache mechanism operates without customer intervention.

For new features in the previous VDDK versions, see the *VDDK 8.0 Release Notes* and before.

Compatibility Notices

More difficult configuration of NFC logs. When customers want to change the NFC server log level to help diagnose NBD backup issues, they must do so with JSON edits of Config Store settings. See How to Change NFC Logging Level below.

VDDK 8.0 U2 supports the following operating systems for proxy backup (mostly the same as VDDK 8.0):

- Windows Server 2022, 2019, and 2016
- Red Hat Enterprise Linux (RHEL) 7.9, 8.1, 8.2, 8.6, 9.0
- **New:** Red Hat Enterprise Linux (RHEL) 8.8
- CentOS 7
- CentOS 8.5
- Oracle Linux 8.7
- SUSE Linux Enterprise Server (SLES) 12 SP5 and 15.1
- Ubuntu 18.04
- Photon v3, v4

This table shows recommended VDDK versions for various VMC milestones:

VMC Milestone	Compatible VDDK Versions
1.16 – 1.19	6.7.x, 7.0.x, 8.0.x
1.20, 1.22, 1.24	7.0.x, 8.0.x

Here is a possible issue with backward and forward compatibility.

- **Best practice for restore with pre-existing snapshot.** When a backed-up VM had snapshots, disaster recovery software can restore a fresh VM with its pre-existing snapshots in place. However for partial file recovery, or to recover a VM back to a point in time, it is difficult to determine how to restore a VM that has an existing snapshot. Should parts of the snapshot be restored to a previous state (if applicable), or only the VM by ignoring the snapshot? Certain backup-restore applications do not handle this situation well; snapshot contention causes a GenericVmConfigFault. One good solution is for customers to delete any pre-existing snapshots before recovery, especially if not needed. This is mandatory for SAN transport recovery, anyway. Restore applications could refuse to recover a VM until any pre-existing snapshot is deleted. If not, a customer work-around is to recover the VM to a different location, avoiding the issue, then retain the recovered VM and delete the VM with pre-existing snapshot.

How to Change NFC Logging Level

For security reasons ESXi 8.0 U2 and VDDK 8.0.2 make it difficult for NFC clients to change the NFC server log level. Now it must be done with JSON edits of Config Store. Because NFC log level set in the VDDK configuration file accounts for changes in various vSphere releases, customers are not affected until they run vSphere 8.0 U2.

For troubleshooting NBD(SSL) backups, configuration of NFC logging changed in almost every vSphere release. Here is a summary:

- In vSphere 7.0 and before, NFC log level was set in the **vpxa.cfg** file.
- In vSphere 7.0 U1, NFC logs are created by hostd rather than vpxa.
- In vSphere 7.0 U2, configuration of NFC log level uses Config Store, set with JSON.
- VDDK 8.0.2 and vSphere 8.0 U2 do not allow NFC clients to change the NFC server log level.

Summarizing the various combinations of ESXi and VDDK:

With ESXi 8.0 U1 and earlier paired with VDDK 8.0.1 and earlier, users can still use `vixDiskLib.nfc.LogLevel` in the VDDK configuration file to set the NFC server log level.

With ESXi 7.0 and earlier paired with VDDK 8.0.2, users must change the vpxa configuration file and restart vpxa.

1. ssh to the ESXi host
2. Edit */etc/vmware/vpxa/vpxa.cfg*

```
            <nfc>
    <loglevel>debug</loglevel>
  </nrc>
```

3. Run command: /etc/init.d/vpxa restart

With ESXi 7.0 U1 paired with VDDK 8.0.2, users must change the hostd log level and restart hostd.

1. ssh to the ESXi host
2. Run command: `vim-cmd hostsvc/advopt/update Config.HostAgent.log.level string trivia`
3. Run command: `/etc/init.d/hostd restart`

With ESXi 7.0 U2 and later paired with VDDK 8.0.2, users must create a temporary file to change settings in Config Store and restart hostd.

1. ssh to the ESXi host
2. Run command: `configstorecli config current get -c esx -g services -k hostd > /tmp/cs.json`
3. Edit /tmp/cs.json

```
"nfcsvc": {
 "log_level": "TRIVIA",
 "max_memory": 100663296,
 "max_stream_memory": 35651584
},
```

4. Run command: `configstorecli config current set -c esx -g services -k hostd -infile /tmp/cs.json`
5. Run command:`/etc/init.d/hostd restart`

With ESXi 8.0 U2 and any VDDK release, users must follow the JSON steps above.

Recently Resolved Issues

The VDDK 8.0.1 release resolved the following issues.

- **Customers complained about nonconfigurable crash dumps.**

  On Windows, vmacore crash dumps now appear in the VDDK configured tmpDir. On Linux, crash dumps appear as configured in the kernel, as they did before.
- **Open SSL library upgraded for FIPS support.**

  The Open SSL library openssl is upgraded to version 3.0 for vSphere 8.0 U1. To enable FIPS with Open SSL 3.0, see below.
- **Reminder about CBT Check in VDDK 7.0.**

VDDK 7.0 included vixDiskCheck utility with cbtCheck option, and now readPerf and writePerf options. For details, find the VDDK bin64 folder and run vixDiskCheck for help on all its options.

For resolved issued in previous VDDK versions, see the *VDDK 8.0 Release Notes* and before.

Known Issues and Workarounds

These are unresolved issues known at this time.

- **New: Non-quiesced snapshot on Linux VM.**

  After a successful quiesced snapshot of a Linux guest, the snapshot may be reported as not quiesced. With VMware Tools versions older than 10.2.0, or ESXi versions older than 6.7, taking a quiesced snapshot on a Linux VM succeeds but appears as a non-quiesced snapshot, even though the snapshot is actually quiesced. A snapshot is identified as quiesced only if a backup manifest file was provided for the snapshot. This works on a Windows VM with VSS plugin, but is not accurate on a Linux VM using the sync driver to quiesce snapshots. To fix this, customers can upgrade vmtools to 10.2.0 or later, and ESXi to 6.7 or later, so it appears as a File-System quiesced snapshot.
- **VDDK non-support of NVMe over Fabrics.**

  vSphere 8 contains many enhancements regarding NVMe over Fabrics (NVMe-oF) such as more namespaces, extend reservations, discovery services, and vVol support. However it seems unlikely that VDDK handles backup for all these new features. More information will be forthcoming. Meanwhile applications can use NBD/NBDSSL and HotAdd transport to back up NVME-oF datastores.
- **Clean-up function not working with default tmpDirectory.**

  With a default temporary directory, the folder name can change from backup to backup, if the backup job is a new OS process, so the `VixDiskLib_Cleanup` function might not clean up everything left behind. To avoid this, programs should set `tmpDirectory` in the configuration file.

Virtual Disk Development Kit 8.0 Update 1 Release Notes

This document contains the following sections

- Introduction
- About the Virtual Disk Development Kit
- Changes and New Features
- Compatibility Notices
- Recently Resolved Issues
- Known Issues and Workarounds

Introduction

VDDK 8.0.1 | 18 April 2023 | ESXi and vCenter Server, VDDK build 21562716

For vSphere 8.0 U1 release | Last document update 2 November 2023

Check back for additions and updates to these release notes, marked **New**.

About the Virtual Disk Development Kit

The Virtual Disk Development Kit (VDDK) 8.0.1 is an update to support vSphere 8.0 Update 1 and respond to partner requests for enhancement.

The VMware policy concerning backward and forward compatibility is for VDDK to support N-2 and N+1 releases. In other words, VDDK 8.0 and all its update releases will support vSphere 6.7, 7.0.x, and (except for new features) the next major release.

VDDK is used with vSphere Storage APIs for Data Protection (VADP) to develop backup and restore software. For general information about this development kit, how to obtain the software, programming details, and redistribution, see the VDDK landing page on developer.vmware.com.

Changes and New Features

VDDK 8.0.1 offers the following features:

- **New: Datastore-level access to VMDK.**

  In this release, it is possible to directly access virtual disks on a datastore. Programs do this by specifying VixDiskLibSpecType as Datastore. At the datastore level, there is no support for snapshots and CBT. See below for steps to employ this feature. Currently only NBD/NBDSSL and HotAdd transport modes are supported. Note that HotAdd transport does not support RDM disks. For datacenters with encrypted disks, NBDSSL transport mode is required, and clusters must activate vSphere HA to guarantee that all hosts have a crypto key ID.
- **New: Wildcards in allow and deny lists for SAN.**

  After configuring allowList and denyList for SAN transport mode, customers and partners requested a wildcard mechanism for ease of use. See section “Initialize Virtual Disk API” in the *VDDK Programming Guide* for details. VDDK 8.0.1 supports a new wildcard mechanism to shorten list making. You can use meta character expressions * ? [xyz] [a-z] [0-9] with the same meanings as in Bash shell.
- **New: Cache file for SAN disk device discovery.**

  Before doing a SAN mode backup, each instance of VDDK scans disks to determine the device name and VMFS ID of the LUN. In large datacenters with many SAN based disks, this has a significant negative performance impact. Partners requested a cache mechanism to reduce frequency of scans. As implemented, cache file scsiDiskList.json is created in the tmpDirectory specified by the VDDK configuration file. When VDDK fails to open a disk with cached LUN path, because disk was added or deleted since last scan, LUNs are rescanned and the JSON file overwritten. This cache mechanism operates without customer intervention.

For new features in the previous VDDK versions, see the *VDDK 8.0 Release Notes* and before.

Datastore-Level Access to VMDK

VDDK header file vixDiskLib.h has a new structure definition for VixDiskLibDatastoreSpec containing two strings: a managed object reference (MoRef) to the datastore, and a folder path to the disk. Datastore specification is enumerated as type 2 in VixDiskLibSpecType. Programs call VixDiskLib_ConnectEx to connect to a datastore instead of a virtual machine or first class disk (FCD). Then programs call VixDiskLib_Open on that datastore with a folder path to open the specified disk.

1. Connect to datastore.

   ```
   VixDiskLibConnection theDSconn;

   VixDiskLibConnectParams *cnxParams = VixDiskLib_AllocateConnectParams();

   connParams->specType = VIXDISKLIB_SPEC_DATASTORE;

   cnxParams->spec.dsSpec.datastoreMoRef = “datastore-19”;

   cnxParams->spec.dsSpec.diskFolder = “RHELXfsTargetClone_dest”;

   VixDiskLib_ConnectEx(cnxParams, ..., &theDSconn);
   ```
2. Open disk on the datastore.

   ```
   VixDiskLibHandle diskHdl;

   VixDiskLib_Open(theDSconn, /* diskPath */, /* open flags */, &diskHdl);
   ```
3. Read from or write to the disk.

4. Close the disk.
5. Disconnect from VixDiskLib.

Wildcards in Allow and Deny Lists

The vixDiskLib.transport.san.allowlisttakes precedence over vixDiskLib.transport.san.denylist in the case of duplicates. Wildcards will not match directory separator slash (/) nor folder separator backslash (\). On Windows, deny and allow lists must contain “\\?\” or “\\.*PhysicalDrive*” prefix, and sharp (#) must be inside double-quotes to avoid interpretation as a comment, allowlist="\\?\scsi#disk*" for example. Also see combined list below the table.

Metacharacters	Examples of Wildcard Use
*	Match the preceding character or subexpression zero or more times,for example /dev/* matches /dev/, /dev/ab, and /dev/cde.
?	Match any single character,for example /dev/sd? matches /dev/sda and /dev/sdb.
[a-z]	Match a character in the specified range,for example /dev/test[1-2] matches /dev/test1 and /dev/test2.
[xyz]	Match any character included in the set,for example /dev/sd[abc] matches /dev/sda, /dev/sdb, and /dev/sdc.

vixDiskLib.transport.san.allowlist="\\?\scsi#disk&ven_scst_bio&prod_disk[1-3]*"

vixDiskLib.transport.san.denylist="\\?\scsi#disk&ven_scst_bio&prod_disk4*"

Compatibility Notices

VDDK 8.0 U1 supports the following operating systems for proxy backup (mostly the same as VDDK 8.0):

- Windows Server 2022, 2019, and 2016
- Red Hat Enterprise Linux (RHEL) 7.9, 8.1, 8.2, 8.6, 9.0
- **New:** RHEL 8.8 was retroactively verified after support in VDDK 8.0.3
- CentOS 7
- CentOS 8.5
- Oracle Linux 8.7
- SUSE Linux Enterprise Server (SLES) 12 SP5 and 15.1
- Ubuntu 18.04
- Photon v3, v4

This table shows recommended VDDK versions for various VMC milestones:

VMC Milestone	Compatible VDDK Versions
1.16 – 1.19	6.7.x, 7.0.x, 8.0.x
1.20, 1.22	7.0.x, 8.0.x

Here is a possible issue with backward and forward compatibility.

- **Best practice for restore with pre-existing snapshot.** When a backed-up VM had snapshots, disaster recovery software can restore a fresh VM with its pre-existing snapshots in place. However for partial file recovery, or to recover a VM back to a point in time, it is difficult to determine how to restore a VM that has an existing snapshot. Should parts of the snapshot be restored to a previous state (if applicable), or only the VM by ignoring the snapshot? Certain backup-restore applications do not handle this situation well; snapshot contention causes a GenericVmConfigFault. One good solution is for customers to delete any pre-existing snapshots before recovery, especially if not needed. This is mandatory for SAN transport recovery, anyway. Restore applications could refuse to

recover a VM until any pre-existing snapshot is deleted. If not, a customer work-around is to recover the VM to a different location, avoiding the issue, then retain the recovered VM and delete the VM with pre-existing snapshot.

Recently Resolved Issues

The VDDK 8.0.1 release resolves the following issue.

- **Customers complained about nonconfigurable crash dumps.**

  On Windows, vmacore crash dumps now appear in the VDDK configured tmpDir. On Linux, crash dumps appear as configured in the kernel, as they did before.
- **Open SSL library upgraded for FIPS support.**

  The Open SSL library openssl is upgraded to version 3.0 for vSphere 8.0 U1. To enable FIPS with Open SSL 3.0, see below.
- **Reminder about CBT Check in VDDK 7.0.**

  VDDK 7.0 included vixDiskCheck utility with cbtCheck option, and now readPerf and writePerf options. For details, find the VDDK bin64 folder and run vixDiskCheck for help on all its options. Also see Troubleshooting CBT Corruption below.

The VDDK 8.0 release resolved the following issues.

- **Snapshot support for cloud native storage (CNS).**

  A new section in the programming guide describes how to create volume snapshots in a Kubernetes cluster and retrieve snapshot handles using container storage interface (CSI) based on first class disk (FCD). Cloud native storage (CNS) is a vSphere feature that allows Kubernetes to auto-provision scalable storage on demand. CNS also provides vSphere administrator visibility into container volumes through vCenter Server.

For resolved issued in previous VDDK versions, see the *VDDK 8.0 Release Notes* and before.

Known Issues and Workarounds

These are unresolved issues known at this time.

- **VDDK non-support of NVMe over Fabrics.**

  vSphere 8 contains many enhancements regarding NVMe over Fabrics (NVMe-oF) such as more namespaces, extend reservations, discovery services, and vVol support. However it seems unlikely that VDDK handles backup for all these new features. More information will be forthcoming. Meanwhile applications can use NBD/NBDSSL and HotAdd transport to back up NVME-oF datastores.

How to Enable FIPS on VDDK Proxy VM

Open SSL 3.0 is required, so the proxy VM must run VDDK 8.0.1 or later.

1.Location of the FIPS dynamic library differs. To install on Windows or Linux, run one of these commands:

openssl.exe fipsinstall -out \path\of\fipsmodule.cnf -module *VDDKpackage*\bin\fips.dll

openssl fipsinstall -out /path/of/fipsmodule.cnf -module *VDDKpackage*/lib64/fips.so

2. In the Open SSL configuration file, dot-include fipsmodule.cnf must be updated with an absolute path, and other values should be set as in this example:

openssl_conf = openssl_init

.include /path/of/fipsmodule.cnf

[openssl_init]

providers = provider_sect

alg_section = algorithm_sect

[provider_sect]

default = default_sect

fips = fips_sect

[default_sect]

activate = 1

[algorithm_sect]

default_properties = "fips=yes"

3. Set environment variable OPENSSL_CONF to the path of the Open SSL configuration file. Set environment variable OPENSSL_MODULES to the path of fips.dll or fips.so, as above.

4. Before VixDiskLib initialization, add vixDiskLib.ssl.enableSslFIPS=1 to the VDDK configuration file.

5. With FIPS enabled, the VDDK information log will record “SSL is in FIPS mode” when VixDiskLibInitEx() is called.

Troubleshooting CBT Corruption

The vixDiskCheck utility was provided to verify data consistency of CBT enabled incremental backups of VMDK virtual disk. In addition to authorization options, similar to options of the VixDiskLib sample program, the -cbtdir option specifies a temporary directory that should be as large as the VMDK being checked. As of VDDK 8.0.1, the utility also measures I/O performance to help debug customer backup environments. Examples:

bin64/vixDiskCheck cbtCheck [auth-options] -cbtdir directory -disk disk.vmdk bin64/vixDiskCheck readPerf [auth-options] -cbtdir directory -disk disk1.vmdk [-disk disk2.vmdk ...] bin64/vixDiskCheck writePerf [auth-options] -cbtdir directory -disk disk1.vmdk [-disk disk2.vmdk ...]

To debug CBT on a customer site, download and install the VDDK package, then locate the bin64/vixDiskCheck utility in the installed folder. Set LD_LIBRARY_PATH (or PATH on Windows) to the lib64 (or bin) folder.

Delete previous snapshots, take a new snapshot, collect the base disk logs *-ctk.vmdk, and run a command like the following. Note that *bigCBTbackupFile* must be as large as VMDK. Repeat those steps with more snapshots if necessary.

vixDiskCheck cbtCheck -host *IPaddr* -user *username* -password *password* -vm moref=*VmMoref*

-ssmoref *SnapshotMoRef* -libdir *VDDKinstallDir* -thumb *SSLhostThumbprint* -cbtdir *bigCBTbackupFile*

-disk Base-disk-000001.vmdk -transportMode nbdssl 2>&1 | tee vixDiskCheck-000001.log

Virtual Disk Development Kit 8.0 Release Notes

This document contains the following sections

- Introduction
- About the Virtual Disk Development Kit
- Changes and New Features
- Compatibility Notices
- Recently Resolved Issues
- Known Issues and Workarounds

Introduction

VDDK for vSphere 8.0 | 11 October 2022 | ESXi and vCenter Server, VDDK build 20521017

Initial Availability became GA 8 November 2022 | Last document update 2 November 2023

Check back for additions and updates to these release notes, marked **New**.

About the Virtual Disk Development Kit

The Virtual Disk Development Kit (VDDK) 8.0 is an update to support vSphere 8.0 and resolve some reported issues.

The VMware policy concerning backward and forward compatibility is for VDDK to support N-2 and N+1 releases. In other words, VDDK 8.0 and all its update releases will support vSphere 6.7, 7.0.x, and (except for new features) the next major release.

VDDK is used with vSphere Storage APIs for Data Protection (VADP) to develop backup and restore software. For general information about this development kit, how to obtain the software, programming details, and redistribution, see the VDDK landing page on developer.vmware.com.

Changes and New Features

VDDK 8.0 offers the following features:

New: Backing up Docker containers.

RedHat Enterprise Linux 8 (RHEL8) Universal Base Images (UBI) as a container image has been tested with VDDK backups using NBD/NBDSSL transport. HotAdd transport is still under investigation. VDDK backup of CentOS 8 containers with Base OS Photon V3 or V4 is supported using NBD/NBDSSL mode.

New: DataSets backup.

A new section in the programming guide describes how to back up DataSets, a vSphere 8.0 feature for sending and storing data in virtual machines.

Better order matching of controller numbering.

KB 67757 and the *VDDK Programming Guide* for 7.0 U3 describe how to power cycle and correct the backup proxy VM when there is a mismatch between SCSI controllers and host initiators. Disordered SCSI numbering causes HotAdd backup to find the wrong disk. HotAdd logic is changed to automatically match SCSI host initiator numbers with SCSI controller labels. This fix made in VDDK 7.0.3.2 is for Linux, although Windows backup proxies sometimes have a similar issue.

Improved NBD backup performance.

On Windows, NBD transport performance was improved in VDDK 7.0.3.2 by changing the default size for NFC send and receive socket buffers to 128KB. On Linux, various socket buffer sizes have minimal performance impact. Customers do not need to change defaults, in case of performance issues, they can change four buffer settings under vixDiskLib.nfcAio.SocketOption in the VDDK configuration file.

Fewer network connections.

A fix made in VDDK 7.0.3.2 reduces the number of connections from VDDK to vCenter Server, especially CEIP queries, as requested by backup partners.

Updated Vix mount functions.

Version 1.1 of VixMntapi library with new GetVolumeInfoEx function that propagates these fields: bytesPerSector, capacityInBytes, usedSpace, and clusterSizeInBytes (usedSpace and clusterSizeInBytes could be inaccurate if the volume is not mounted).

FCD change tracking.

VDDK documentation describes how to use QueryChangedDiskAreas for VM disk. This function can be used for FCD as well. There are two cases for FCD, attached to VM or not. If FCD is attached to a VM, you can use the same API as for VM disk. If FCD is detached, you must call FCD's QueryChangedDiskAreas API directly to get diffs.

Containers in NBD mode.

This release supports NBD and NBDSSL backup of containers. VDDK programs can run in a general container instance.

New proxy OS support.

VDDK 8.0 supports five additional operating systems as backup proxy VMs for HotAdd and other transports: Red Hat Enterprise Linux (RHEL) 8.1, 8.2, Photon v3, v4, and Windows 2022, in addition to those supported by VDDK 7.0.3.

For new features in previous VDDK versions, see the *VDDK 7.0.3.2 Release Notes* and before.

Compatibility Notices

VDDK 8.0 supports the following operating systems for proxy backup:

- Windows Server 2022, 2019, and 2016
- Red Hat Enterprise Linux (RHEL) 7.9, 8.1, 8.2, 8.6, 9.0
- **New:** RHEL 8.8 was retroactively verified after support in VDDK 8.0.3
- CentOS 7
- SUSE Linux Enterprise Server (SLES) 12 SP5 and 15.1
- Ubuntu 18.04
- Photon v3, v4

This table shows recommended VDDK versions for various VMC milestones:

VMC Milestone	Compatible VDDK Versions
M16 – M19	6.7.x, 7.0.x, 8.0.x
M20	7.0.x, 8.0.x

Here are possible issues with backward and forward compatibility.

- **New:Best practice for restore with pre-existing snapshot.** When a backed-up VM had snapshots, disaster recovery software can restore a fresh VM with its pre-existing snapshots in place. However for partial file recovery, or to recover a VM back to a point in time, it is difficult to determine how to restore a VM that has an existing snapshot. Should parts of the snapshot be restored to a previous state (if applicable), or only the VM by ignoring the snapshot? Certain backup-restore applications do not handle this situation well; snapshot contention causes a GenericVmConfigFault. One good solution is for customers to delete any pre-existing snapshots before recovery, especially if not needed. This is mandatory for SAN transport recovery, anyway. Restore applications could refuse to recover a VM until any pre-existing snapshot is deleted. If not, a customer work-around is to recover the VM to a different location, avoiding the issue, then retain the recovered VM and delete the VM with pre-existing snapshot.
- Superseded years ago by the CreateSnapshotEx_Task method, CreateSnapshot_Task is deprecated in this release.
- In the configuration file for SAN transport mode, the original keywords that denyList and allowList replaced are no longer accepted for backward compatibility, as they have been since the changeover in vSphere 7.0 U1.
- Missing from documentation (until 7.0.1) was this caveat about multithreading: QueryAllocatedBlocks should be called in the same thread as open and close disk, not from read and write worker threads.
- vVol and vSAN datastores do not support SAN mode transport. SAN transport is explicitly rejected for vSAN and vVols.

Recently Resolved Issues

The VDDK 8.0 release resolves the following issues.

- **Snapshot support for cloud native storage (CNS).**

  A new section in the programming guide describes how to create volume snapshots in a Kubernetes cluster and retrieve snapshot handles using container storage interface (CSI) based on first class disk (FCD). Cloud native storage (CNS) is a vSphere feature that allows Kubernetes to auto-provision scalable storage on demand. CNS also provides vSphere administrator visibility into container volumes through vCenter Server.
- **Could not set NFC server log level.**

  In VDDK 8.0 updates, the NFC server logging level will no longer configurable from the VDDK client configuration file. This was originally for debugging but will be discontinued for security reasons.
- **Open source libraries upgraded.**

  The Open SSL library openssl was upgraded to version 1.0.2zf, the XML streaming library libexpat to version 2.4.9, and the data compression library Zlib to version 1.2.12.

For resolved issued in previous VDDK versions, see the *VDDK 7.0.3.2 Release Notes* and before.

Known Issues and Workarounds

These are unresolved issues known at this time.

- **New: VDDK non-support of NVMe over Fabrics.**

  vSphere 8 contains many enhancements regarding NVMe over Fabrics (NVMe-oF) such as more namespaces, extend reservations, discovery services, and vVol support. However it seems unlikely that VDDK handles backup for all these new features. More information will be forthcoming. Meanwhile applications can use NBD/NBDSSL and HotAdd transport to back up NVME-oF datastores.
- **Linux backup proxy requires minimum vmx=10.**

  Because of a resiliency improvement made for VDDK 7.0.3.2 (corrected HotAdd failures due to mismatch of controller ordering) a Linux backup proxy no longer works with virtual hardware version 9 (vmx=9). Error message is “Could not find the label of controller 0.” The workaround is to upgrade backup proxies to virtual hardware version 10 or later.
- **On VMC with NFS, Storage vMotion may conflict with backup.**

  During backup a VM's disks are locked against removal. If Storage vMotion is requested against that VM, VM disks remain afterwards, and must be removed manually. The longer backup takes, the higher the likelihood of this occurring. Customers should avoid Storage vMotion during the backup window, but if not, they can find VMDKs not associated with VM or FCD, and remove them.
- **Linked clone problems with HotAdd backup on vVol datastores.**

  If backup software creates a linked clone target VM from a linked clone VM on a vVol datastore, then performs a backup using HotAdd transport, the proxy VM might become invalid when opening a disk on that target VM. This was a regression from VDDK 7.0.2 to 7.0.3. One alternative is to create the linked clone target VM from a full-clone source VM, rather than from a linked clone VM. Another alternative is to choose a transport mode other than HotAdd.
- **Clean-up function not working with default tmpDirectory.**

  With a default temporary directory, the folder name can change from backup to backup, if the backup job is a new OS process, so the VixDiskLib_Cleanup function might not clean up everything left behind. To avoid this, programs should set tmpDirectory in the configuration file.

vSphere Guest SDK

Guest SDK 12.3.0 Release Notes

This document contains the following sections

- Introduction
- About the Guest SDK
- Compatibility Notices
- Changes and New Features
- Known Issues and Workarounds

Introduction

Guest SDK 12.3.0 for vSphere 8.0 U2 | 21 September 2023 | SDK on developer.vmware.com

For ESXi 8.0 U2 and vCenter Server 8.0 U2 | Last document update: 20 September 2023

Check back for additions and updates to these release notes.

About the Guest SDK

You can download the VMware Guest SDK from the developer.vmware.com website. Documentation is provided on the landing page with SDK bits: tarball for Linux and Zip file for Windows. SDK version number 12.3.0 was prepared for and released with vSphere 8.0 Update 2.

The Guest SDK includes four components for different use cases:

- **Guest API.** The Guest API provides C/C++ language functions to monitor virtual machine statistics. Management agents running in a virtual machine's guest OS can use this data for reacting to changes in the application layer.
- **Guest Statistics.** The statistics fetch facility returns detailed data in several structured formats about virtual machines running on an ESXi host.
- **DataSets.** New in vSphere 8.0, the DataSets API can manipulate VM-related key-value entries using vSphere Automation from vCenter Server, or with VM Tools inside a guest OS.
- **HA Application Monitoring.** The vSphere High Availability (HA) feature for ESXi hosts in a cluster provides protection for a guest OS and applications running in a virtual machine by restarting the virtual machine if a failure occurs. Using the HA Application Monitoring APIs, developers can write software to monitor the process heartbeat of a guest OS.

Compatibility Notices

The Guest SDK in vSphere 8.0 U2 is compatible with VMware Tools version 12.3.0 and later.

As of vSphere 8.0, VMware Tools gets built by Visual Studio 2019, so on Windows the Guest SDK depends on runtime vc140.

The never-documented guest API VMGuestLib_GetHostMemKernOvhdMB was deprecated as of VMware Tools 11.2.0. The GUESTLIB_HOST_MEM_KERN_OVHD_MB performance counter that it returns was documented in *vSphere Virtual Machine Administration* for ESXi 5.5, but dropped in subsequent documentation.

For vSphere 6.0, HA Application Monitoring communication was revised to use the VMCI (virtual machine communication interface). The VMCI driver is preinstalled in Linux kernel 3.9 and higher, and in earlier kernel versions it can be installed with VMware Tools. On Windows, VMware Tools must be installed to obtain the VMCI driver.

Changes and New Features

The DataSets facility was new in version 12.1.0 for vSphere 8.0. The *Guest SDK Programming Guide* has a chapter about it, and the *Virtual Machine Administration* manual discusses it also.

The VMGuestLib_GetMemShares64 call was added in vSphere 7.0 to support large-memory virtual machines.

Known Issues and Workarounds

The HA Application Monitoring library is not thread safe. It assumes that only one monitor process generates heartbeats in the VM. If a program runs multiple monitor threads, the library will leak memory, and RPC processes might not exit when the program terminates. Moreover, unexpected VM resets may occur owing to lack of heartbeats in a different thread. Monitoring programs should either avoid multithreading, or call the library from a single thread only.

Security enforcement for the Guest and HA application monitoring SDK using the secure authentication VMX parameter guest_rpc.rpci.auth.app.APP_MONITOR=TRUE does not work for FT (fault tolerant) virtual machines. The vSphere platform supports only the non-secure channel for FT virtual machines.

Sample code for HA Application Monitoring does not demonstrate use of the new API to request reset of the guest virtual machine, VMGuestAppMonitor_PostAppState.

Guest SDK 12.1.0 Release Notes

This document contains the following sections

- Introduction
- About the Guest SDK
- Compatibility Notices
- Changes and New Features
- Known Issues and Workarounds

Introduction

Guest SDK 12.1.0 for vSphere 8.0 | 11 October 2022 | SDK on developer.vmware.com

Initial Availability became GA 8 November 2022 | Last document update: 9 November 2022

Check back for additions and updates to these release notes.

About the Guest SDK

You can download the VMware Guest SDK from the developer.vmware.com website. Documentation is provided on the landing page with SDK bits: tarball for Linux and Zip file for Windows. SDK version number 12.1.0 was prepared for and released with vSphere 8.0.

The Guest SDK includes four components for different use cases:

- **Guest API.** The Guest API provides C/C++ language functions to monitor virtual machine statistics. Management agents running in a virtual machine's guest OS can use this data for reacting to changes in the application layer.
- **Guest Statistics.** The statistics fetch facility returns detailed data in several structured formats about virtual machines running on an ESXi host.
- **DataSets.** New in vSphere 8.0, the DataSets API can manipulate VM-related key-value entries using vSphere Automation from vCenter Server, or with VM Tools inside a guest OS.
- **HA Application Monitoring.** The vSphere High Availability (HA) feature for ESXi hosts in a cluster provides protection for a guest OS and applications running in a virtual machine by restarting the virtual machine if a failure occurs. Using the HA Application Monitoring APIs, developers can write software to monitor the process heartbeat of a guest OS.

Compatibility Notices

The Guest SDK in vSphere 8.0 is compatible with VMware Tools version 12.1.0 and later.

The never-documented guest API VMGuestLib_GetHostMemKernOvhdMB was deprecated as of VMware Tools 11.2.0. The GUESTLIB_HOST_MEM_KERN_OVHD_MB performance counter that it returns was documented in *vSphere Virtual Machine Administration* for ESXi 5.5, but dropped in subsequent documentation.

For vSphere 6.0, HA Application Monitoring communication was revised to use the VMCI (virtual machine communication interface). The VMCI driver is preinstalled in Linux kernel 3.9 and higher, and in earlier kernel versions it can be installed with VMware Tools. On Windows, VMware Tools must be installed to obtain the VMCI driver.

Changes and New Features

The DataSets facility is new in version 12.1.0 for vSphere 8.0. The *Guest SDK Programming Guide* has a chapter about it, and the *Virtual Machine Administration* manual discusses it also.

As of vSphere 8.0, VMware Tools gets built by Visual Studio 2019, so on Windows the Guest SDK depends on runtime vc140.

The VMGuestLib_GetMemShares64 call was added in vSphere 7.0 to support large-memory virtual machines.

Known Issues and Workarounds

The HA Application Monitoring library is not thread safe. It assumes that only one monitor process generates heartbeats in the VM. If a program runs multiple monitor threads, the library will leak memory, and RPC processes might not exit when the program terminates. Moreover, unexpected VM resets may occur owing to lack of heartbeats in a different thread. Monitoring programs should either avoid multithreading, or call the library from a single thread only.

Security enforcement for the Guest and HA application monitoring SDK using the secure authentication VMX parameter guest_rpc.rpci.auth.app.APP_MONITOR=TRUE does not work for FT (fault tolerant) virtual machines. The vSphere platform supports only the non-secure channel for FT virtual machines.

Sample code for HA Application Monitoring does not demonstrate use of the new API to request reset of the guest virtual machine, VMGuestAppMonitor_PostAppState.

VMware HTML Console SDK 2.2.0 Release Notes

This document contains the following sections

- Introduction
- About the HTML Console SDK
- What's New in this Release
- Quick Start
- Compatibility Notices
- Recently Resolved Issues
- Known Issues and Workarounds

Introduction

VMware HTML Console SDK 2.2.0 | 14 November 2022 | Zip on developer.vmware.com

For VMware vSphere and vCloud Director 8.0 | Last document update: 16 November 2022

Check back for additions and updates to these release notes.

About the HTML Console SDK

The HTML Console SDK allows developers to add virtual machine console functionality to an existing web user interface running on vSphere. Using applications created by this SDK, end users can interact with a virtual machine console by typing on the keyboard and moving the mouse.

The HTML Console SDK provides APIs with the following functionality:

- Mouse and keyboard input, including mouse scroll without VMware Tools
- Send Ctrl + Alt + Delete
- Full screen mode
- Client and guest keyboards for Japanese, German, Italian, Spanish, Portuguese (pt_pt), French, Swiss-French, and Swiss-German.

What's New in this Release

This version follows vSphere 8.0 and is the first update since 2019. The programming guide contains new sample code showing how to copy from a remote virtual machine to the local HTML Console client.

The SDK is available in the Download section of the landing page. You might have to sign in using your VMware Customer Connect account. The SDK is delivered as a ZIP file. After you download and unzip the file, you use Javascript and CSS directly.

See compatibility notices below for browser and OS support.

Quick Start

To see how it works, place this HTML code in a folder with the unzipped SDK.

```
<!DOCTYPE html PUBLIC"-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="content-type" content="text/html; charset=utf-8" />

<title>Console</title>

</head>

<body>

<link rel="stylesheet" type="text/css" href="css/wmks-all.css" />

<script type="text/javascript" src="http://code.jquery.com/jquery-1.8.3.min.js"></script>

<script type="text/javascript" src="http://code.jquery.com/ui/1.8.16/jquery-ui.min.js"></script>

<script type="text/javascript" src="wmks.min.js"></script>

<div id="wmksContainer" style="position:absolute;width:100%;height:100%"></div>

<script>

var wmks = WMKS.createWMKS("wmksContainer",{})

.register(WMKS.CONST.Events.CONNECTION_STATE_CHANGE, function(event,data){

if(data.state == WMKS.CONST.ConnectionState.CONNECTED){

  console.log("connection state change : connected");}
```

});

wmks.connect("wss://***ESXi.host.IP.Address***:443/ticket/***webmksTicket***");

</script>

</body>

</html>

Replace ***ESXi.host.IP.Address*** with the IP address or fully-qualified hostname of the ESXi host where the virtual machine resides.

To get the ***webmksTicket***, find the vCenter Server that manages the above ESXi host. You can get a ticket using the vSphere API, or with the managed object browser:

1. In a browser go to https://***vCenter.Server.IP.Address***/mob
2. Provide your login credentials for Administrator@vsphere.local or another SSO account
3. Click content > rootFolder > Datacenter > vmFolder
4. Click the VM number of the virtual machine you want to connect to
5. Under Methods, click AcquireTicket
6. Type **webmks** in the value field and click Invoke Method
7. Copy and paste the ticket string to replace ***webmksTicket***
8. Double-click the above HTML file to open it in your browser.

Once you know the managed object ID (moid) of the virtual machine, you can quickly get another ticket at this URL. Replace ***NN*** with the VM number:

https://***vCenter.Server.IP.Address***/mob/?moid=vm-***NN***&method=acquireTicket

Compatibility Notices

The HTML Console SDK was tested with recent versions of the following Web browsers on Windows, MacOS, and Linux:

- Google Chrome
- Mozilla Firefox
- Apple Safari (MacOS only)
- Microsoft Edge
- Internet Explorer (Windows only)

The test/ folder is new in the 2.2.0 release. Rather than follow steps in the README.md file, because unit test source files are VMware internal, follow these steps for testing:

1. Install the nodejs tools from http://notejs.org/download.

   This provides the npm command for installing other Javascript packages.
2. Install grunt-cli globally on your system.

   This adds the grunt command, and only needs to be done once.
3. Add the grunt install location to your PATH environment.
4. Download the Node libraries. You can do this in a single step.

   This puts a bunch of node libraries within the node_modules.

   npm install
5. Run unit test by typing grunt with or without the test argument.

This runs all the unit tests and reports status on the command line before exiting.

grunt test

6. After you run the unit tests, look for the Jasmine HTML report in the wmkssdk/test/karma_html/ folder.

Recently Resolved Issues

The HTML Console 2.1 release resolved the following issues.

- **HTML Console support for vCloud Director.**

  Version 9.x of vCloud Director (vCD) supports HTML Console 2.1 and the stand-alone VMware Remote Console (VMRC).
- **More international keyboard support.**

  The HTML Console 1.0 release supported only English and Japanese keyboards. Release 2.0 added support for German, Italian, Spanish, and Portuguese keyboards. This release also supports French, Swiss-French, and Swiss-German keyboards.
- **The css and img folders went too deep.**

  In the 2.0 release, style sheets in the css/css folder were not able to reference the touch-sprite images in the img/img folder. In this release all css/css style sheets and img/img images are at the correct folder level.

Known Issues and Workarounds

The following issues are found in the HTML Console 2.2 release.

- **Incomplete support for international keyboards.**

  In this release no other international keyboards are supported other than Japanese, German, Italian, Spanish, Portuguese (pt_pt), French, Swiss-French, and Swiss-German.

ESXCLI 8.0 Release Notes

This document contains the following sections

- Introduction
- ESXCLI 8.0 Release Notes
- About ESXCLI
- What's New
- Supported Platforms

Introduction

ESXCLI 8.0 | 23 AUG 2023 | Build 22179150

Check for additions and updates to these release notes.

ESXCLI 8.0 Release Notes

Welcome to the ESXCLI 8.0 release notes. Version numbers referenced in this document are placeholders and do not represent any commitment by VMware to have any features included in specific software versions.

About ESXCLI

The ESXCLI command set allows you to run common system administration commands from any machine with network access to those systems. ESXCLI includes a set of host management commands. You can run most commands against a vCenter Server system and target any ESXi system that the vCenter Server system manages.

What's New

This release changes the way ESXCLI is packaged and installed. The previous release included OS-specific binary installers for Linux and Windows. Now, ESXCLI is distributed as a single PyPI-compatible Python package that you can install on Linux, Windows, and macOS platforms by using the *pip install* command.

Supported Platforms

For this release, ESXCLI is supported on Linux, Windows, and macOS platforms that run the following Python versions:

- Python 3 version 3.6.0 or later
- Python 2 version 2.7.9 or later

VMware OVF Tool

VMware OVF Tool 4.6.3 Release Notes

This document contains the following sections

- Introduction
- About the OVF Tool
- Before You Begin
- What's New
- Resolved Issues
- Compatibility Notices
- Known Issues and Workarounds

Introduction

OVF Tool 4.6.3 | 25 June 2024 | Builds under developer.broadcom.com/tools

Released with vSphere 8.0 Update 3

Check back for additions and updates to these release notes, marked **New**.

About the OVF Tool

The OVF Tool 4.6.3 is released with vSphere 8.0 U3. The previous OVF Tool 4.6.2 was released with vSphere 8.0 U2.

VMware OVF Tool is a command-line utility that imports and exports OVF packages to and from virtual machines running on VMware virtualization platforms. OVF Tool gets called internally by many VMware products.

Before You Begin

You can download the OVF Tool for installation on Windows 64-bit or 32-bit, Linux 64-bit or 32-bit, and MacOS 64-bit. The OVF Tool landing page provides a link to the downloads for each release.

OVF Tool 4.6.3 supports the following operating systems:

- Windows 11 64-bit (x86_64)
- Windows 10 32-bit (x86) and 64-bit (x86_64)
- Windows 8.1 and Windows 7 SP1, 32-bit and 64-bit
- Windows Server 2022, 2019, 2016, and 2012 R2
- MacOS versions including 12 Monterey to 14 Sonoma
- Red Hat Enterprise Linux (RHEL) recent releases
- Recent releases of CentOS and Fedora

- SUSE Linux Enterprise Server (SLES) recent releases
- Ubuntu Linux and variants, recent releases
- VMware Photon OS and Oracle Linux

Support for vCloud Director (VCD) 10.4.1 has been added. OVF Tool 4.6.0 was tested with VCD 10.2, 10.3, and 10.4.

What's New

These are **new** features in OVF Tool 4.6.3:

Using OpenSSL3 with TLS 1.3 protocol.

In vSphere 8.0 Update 3, vCenter Server moves to full compliance with FIPS (the Federal Information Processing Standard), including mandatory use of OpenSSL version 3 and TLS 1.3 protocol. OVF Tool has been updated to use OpenSSL version 3 with TLS 1.3 as well.

Add multi-page support for VCD Catalogs.

Previous releases of OVF Tool had a maxium page limit of 25 items. When importing VM and vApp templates into the vCloud Director (VCD) Catalog, customers ran up against this limit, resulting in “Catalog not found” and “No vAppTemplate found” errors. This release adds support for VCD Catalogs with multiple page results. When queries have multi-page results for Catalogs and VDC, OVF Tool searches again if the name is not found initially, rather than returning “not found” if the first page result lacks the searched-for name.

Call Datacenter power-on for compatibility with DRS.

Rather than use VirtualMachine specific **powerOnVM**, which fails to account for the current state of Distributed Resource Scheduling (DRS), this release calls **powerOnMultiVM** if the VI target is a VC, in other words managed by vCenter in a vSphere cluster. VirtualMachine specific **powerOnVM** is still used for a stand-alone VM.

Support for 4Kn virtual disk format.

OVF Tool can support native 4K disk format (4Kn) with the options below, but might not communicate properly with servers until the release of vSphere 9.

vmw:Config ovf:required="false" vmw:key="virtualDiskFormat" vmw:value="native_4k"

Open source components.

These constituent open source components were upgraded: **openssl** to version 3.0.13 (and 1.0.2zj for Apache Xerces), **curl** to 8.7, **nghttp2** to 1.61, and **zlib** to 1.3.1.

For a summary of new features in previous releases, see the *OVF Tool 4.6.0 Release Notes* and the *OVF Tool 4.6.2 Release Notes*.

Resolved Issues

These issues are fixed in OVF Tool 4.6.3:

- **Fix host name selection bug.**

  When using OVF Tool to upload and download VMs by host name from vSphere infrastructure, the error message “Failed to connect to host *name* due to: Couldn't resolve host name” sometimes appeared. The workaround was to use IP address instead of host name. Hostname selection is fixed in this release by clearing the previous name string and by randomizing the deploy target within range of the host array.
- **Fix CoresPerSocket generation for OVF parser.**

  OVF Tool had a bug where non-processor XML elements could be exported with **CoresPerSocket** element. This release fixes the OVF parser to output the **CoresPerSocket** value only when the XML element is type Processor (CPU).

- **Fix issue with Boot order and StorageProfile sections in OVF.**

  When an OVF file specified both BootOrderSection and StorageGroupSection, resource pool import could fail with (Storage) “Profile not found” error. In some cases it worked with vSphere but not with VCD. This release is more lenient about varied ordering of boot and storage policies.
- **Fix (or feature) to add support for VMIOP with virtual GPU.**

  The VMIOP (virtual machine input output plugin) device is used for backing of virtual GPUs. This fix adds support for VMIOP with virtual GPU parameters for bidirectional conversions, including VMX and OVF file processing.

Compatibility Notices

When customers try to install vCenter Server from a browser on MacOS 10.15 Catalina, and later, a popup dialog appears saying “vcsa-deploy.bin cannot be opened because the developer cannot be verified” and installation fails with error “ovftool cannot be opened because the developer cannot be verified.” This is due to greater security in MacOS and OVF Tool not being notarized for Apple. See KB 79416 for workarounds.

Known Issues and Workarounds

This known issue was not fixed in time for this release:

- **OVF Tool rejects added XML metadata.**

  When a vendor places additional XML metadata into an OVA file, deployment can fail with an “Unsupported element” error. This will be fixed by being more forgiving about placement of added XML items.

These issues were reported in earlier releases:

- **ResourcePool import fails with OVF reference to empty storage policy.**

  When an OVF contains both StorageGroupSection and BootOrderSection, deployment with ResourcePool.ImportVApp fails, because even a valid storage policy is converted to profileId = "" (empty string). The workaround is to manually remove BootOrderSection from the OVF if StorageGroupSection is valid, or instead set EmptyProfileSpec for the disk.
- **OVF Tool interaction with Apple Silicon.**

  The MacOS 64-bit binary extracted from Zip runs on ARM processors (M1 and M2) when the Rosetta2 emulator is already installed. OVF Tool export might have difficulties with certain hardware specifics, but no such issues have been reported so far.
- **Network services library required.**

  OVF Tool calls the network services library **libnsl** but Red Hat Enterprise Linux 8 and Fedora 27 removed this library. Export fails and the message “export failed: unknown error” appears. The solution is to install **libnsl** and restart the Linux machine, as detailed in VMware KB 89515.

VMware OVF Tool 4.6.2 Release Notes

This document contains the following sections

- Introduction
- About the OVF Tool
- Before You Begin
- What's New
- Compatibility Notices
- Resolved Issues
- Known Issues and Workarounds

Introduction

OVF Tool 4.6.2 | 21 September 2023 | Build at developer.vmware.com/tools

Released with vSphere 8.0 Update 2 | Last document update 28 November 2023

Check for additions and updates to these release notes.

About the OVF Tool

The OVF Tool 4.6.2 is released with vSphere 8.0 U2. The previous OVF Tool 4.6.0 was released with vSphere 8.0 U1.

VMware OVF Tool is a command-line utility that allows import and export of OVF packages to and from virtual machines running on VMware virtualization platforms. OVF Tool gets called internally by many VMware products.

Before You Begin

You can download the OVF Tool for installation on Windows 64-bit or 32-bit, Linux 64-bit or 32-bit, and MacOS 64-bit. The OVF Tool landing page provides a link to the downloads for each release.

OVF Tool 4.6.2 supports the following operating systems:

- Windows 11 32-bit (x86) and 64-bit (x86_64)
- Windows 10 32-bit (x86) and 64-bit (x86_64)
- Windows 8.1 and Windows 7 SP1, 32-bit and 64-bit
- Windows Server 2022, 2019, 2016, and 2012 R2
- MacOS versions including 12 Monterey and 13 Ventura
- Red Hat Enterprise Linux (RHEL) recent releases
- Recent releases of CentOS and Fedora
- SUSE Linux Enterprise Server (SLES) recent releases
- Ubuntu Linux and variants, recent releases
- VMware Photon OS and Oracle Linux

Support for vCloud Director (VCD) 10.4.1 has been added. OVF Tool 4.6.0 was tested with VCD 10.2, 10.3, and 10.4.

What's New

These constituent open source components were upgraded: openssl to version 1.0.2zh.14260, curl to 8.1.2.9481, zlib to 1.2.13.6986, and c-ares DNS to 1.19.0.4101.

New: OVF support for SPBM storage profiles. When exporting a VM to OVF, vSphere extracts storage policy fields and adds them, for both disks and VM home, to the StorageGroupSection. When importing or deploying an OVF, storage profiles are preserved and OVF Tool checks to avoid duplicates. Previously when customers exported a VM to OVF twice, the second export did not import or deploy due to storage profile confusion, and this error message appeared: “Multiple sections of type StorageSection (Storage policy group reference) not allowed.” The workaround with older OVF Tool was to import or deploy with PowerCLI instead of the vSphere Client UI.

SPBM identifiers in the vSphere inventory. If an SPBM identifier is not present in the vSphere inventory, it is ignored during OVF deployment. For example, if the OVF contains BootOrderSection but this is not in the inventory, the OVF import or re-import may fail with error message such as “A specified parameter was not correct: Profile *ProfileId* not found.” The solution for such failures is adding to the vSphere inventory important SPBM identifiers that appear in error messages, or removing non-inventoried items such as BootOrderSection from the OVF.

For a summary of new features in the previous 4.6.0 release, see the *OVF Tool 4.6.0 Release Notes*.

Compatibility Notices

When customers try to install vCenter Server from a browser on MacOS 10.15 Catalina, MacOS 11.x Big Sur, MacOS 12 Monterey, or MacOS 13 Ventura, a popup dialog appears saying “vcsa-deploy.bin cannot be opened because the

developer cannot be verified” and installation fails with error “ovftool cannot be opened because the developer cannot be verified.” This is due to greater security in MacOS and OVF Tool not being notarized for Apple. See KB 79416 for workarounds.

Resolved Issues

These issues were fixed in OVF Tool 4.6.2:

- **Better resiliancy for massive deployments.**

  When thousands of VMs are deployed from OVF in a short time period, OVF Tool occasionally throws a ManagedObjectNotFound error. This is caused by a VM disappearing from the inventory during deployment. In this release, OVF Tool ignores the ManagedObjectNotFound exception and proceeds without exiting. If important, you can search logs afterwards to determine which VM deployments failed.
- **No waiting for non-existent task to complete.**

  During smoke testing, it was determined that ESXi hosts sometimes returned the wrong task in response to a GetRecentTask request. This was observed on deployments of embedded ESXi VMs. Possibly the wrong task was returned because the requested task had completed. In this release, OVF Tool checks the task name to be sure it's the right one. If not OVF Tool continues without calling WaitForTask and eventually marks the task Complete.
- **More convenient debugging with ConfigFile logging.**

  Added a new vmxFromConfigFile option to extract ConfigInfo from a live VM, convert it to VMX using ConvertConfigInfoToVmx, and save results in a specified file. For example:

  ovftool --X:vmxFromConfigFile=vmxConfigOut.log vi://admin@vsphere.local:test.example.vmware.com//vm/TestVM2
- **OVF Tool support for virtual TPM on hosted platforms.**

  For hosted platforms VMware Fusion and Workstation, OVF Tool's probe function was extended to detect the vtpm flag, which indicates that the VM supports virtual TPM (trusted platform module). OVF Tool was also changed to not add vtpm.present flag when the VM does not contain or cannot handle a virtual TPM. These changes prevent boot issues and improve security.

Known Issues and Workarounds

These issues were reported by customers or discovered during testing:

- **New: ResourcePool import fails with OVF reference to empty storage policy.**

  When an OVF contains both StorageGroupSection and BootOrderSection, deployment with ResourcePool.ImportVApp fails, because even a valid storage policy is converted to profileId = "" (empty string). The workaround is to manually remove BootOrderSection from the OVF, if StorageGroupSection is valid, or instead set EmptyProfileSpec for the disk.
- **OVF Tool interaction with Apple Silicon.**

  The MacOS 64-bit binary extracted from Zip runs on ARM processors (M1 and M2) when the Rosetta2 emulator is already installed. OVF Tool export might have difficulties with certain hardware specifics, but any such issues have not been fully characterized.
- **Network services library required.**

  OVF Tool calls the network services library libnsl but Red Hat Enterprise Linux 8 and Fedora 27 removed this library. Export fails and message “export failed: unknown error” appears. The solution is to install libnsl and restart the Linux machine, as detailed in VMware KB 89515.

VMware OVF Tool 4.6.0 Release Notes

This document contains the following sections

- Introduction
- About the OVF Tool
- Before You Begin
- What's New
- Compatibility Notices
- Resolved Issues
- Known Issues and Workarounds

Introduction

OVF Tool 4.6.0 | 18 April 2023 | Build 21259398 at developer.vmware.com/tools

Released with vSphere 8.0 Update 1 | Last document update 6 April 2023

Check for additions and updates to these release notes.

About the OVF Tool

The OVF Tool 4.6.0 is released with vSphere 8.0 U1 and vCloud Director. The previous OVF Tool 4.5.0 was released with vSphere 8.0.

VMware OVF Tool is a command-line utility that allows import and export of OVF packages to and from virtual machines running on VMware virtualization platforms. OVF Tool gets called internally by many VMware products.

Before You Begin

You can download the OVF Tool for installation on Windows 64-bit or 32-bit, Linux 64-bit or 32-bit, and Mac OS X 64-bit. The OVF Tool landing page provides a link to the downloads for each release.

OVF Tool 4.6 supports the following operating systems:

- Windows 11 32-bit (x86) and 64-bit (x86_64)
- Windows 10 32-bit (x86) and 64-bit (x86_64)
- Windows 8.1 and Windows 7 SP1, 32-bit and 64-bit
- Windows Server 2022, 2019, 2016, and 2012 R2
- MacOS versions including 11 Big Sur and 12 Monterey
- Red Hat Enterprise Linux (RHEL) recent releases
- Recent releases of CentOS and Fedora
- SUSE Linux Enterprise Server (SLES) recent releases
- Ubuntu Linux and variants, recent releases
- VMware Photon OS and Oracle Linux

Support for vCloud Director (VCD) 10.4.1 has been added. OVF Tool 4.6.0 was tested with VCD 10.2, 10.3, and 10.4.

What's New

No new features, but two issues found with vCloud Director interaction are fixed, as described in resolved issues below. For a summary of new features in the 4.5.0 release, see the *OVF Tool 4.5 Release Notes*.

Many constituent open source components were upgraded.

Compatibility Notices

When customers try to install vCenter Server 7.0.x from a browser on MacOS 10.15 Catalina, or Mac OS 11.x Big Sur, a popup dialog appears saying “vcsa-deploy.bin cannot be opened because the developer cannot be verified” and installation fails with error “ovftool cannot be opened because the developer cannot be verified.” This is due to greater security in MacOS and OVF Tool not being notarized for Apple. See KB 79416 for workarounds.

Resolved Issues

These issues were fixed in OVF Tool 4.6.0:

- **Case-sensitive scanning of HTTP headers sometimes failed to match.**

  When parsing vCloud Director headers, OVF Tool used case sensitive comparison to find relevant items. In some cases, keywords in HTTP header were capitalized unexpectedly, so comparison failed and key values were missed. The fix was to change keyword field search to be case insensitive.
- **Omitting password caused vCloud Director URL to throw HTTP 403 error.**

  When users typed name but not password in the source or destination vCloud Directory URL, an HTTP 403 error page resulted, so OVF Tool was not able to access the vApp for download. All other org/vdc operations worked until the point of fetching the vApp. This issue occurred only when username was supplied but password was not. When OVF Tool prompted for the password, the previously supplied username was lost, or in bearer mode, authorization was reset from bearer to basic authentication mode. In either case, vCloud Directory returend a 403 error. The fix was to prevent authorization mode change during HTTP transactions.
- **Error: No common hosts found.**

  When deploying to a vCenter Server and selecting a vSphere Cluster in the target URL, if the target network specified in network mapping has many hosts attached to it, OVF Tool mistakenly returned the error “No common hosts found” (see “Setting OVF Network Maps When Deploying to vSphere” in the *OVF Tool User Guide*). The workaround wass to specify, after cluster name, a host with access to both the desired network and datastore. A fix was incorporated in this release.

Known Issues and Workarounds

These issues were reported by customers or discovered during testing:

- **OVF Tool interaction with Apple Silicon.**

  The MacOS 64-bit binary extracted from Zip runs on ARM processors (M1 and M2) if the Rosetta2 emulator is already installed. OVF Tool export might have difficulties with certain hardware specifics, but these issues have not been fully characterized.
- **Network services library required.**

  OVF Tool calls the network services library libnsl but Red Hat Enterprise Linux 8 and Fedora 27 removed this library. Export fails and message “export failed: unknown error” appears. The solution is to install libnsl and restart the system, as detailed in VMware KB 89515.

VMware OVF Tool 4.5.4 Release Notes

This document contains the following sections

- Introduction
- About the OVF Tool
- Before You Begin
- What's New
- Compatibility Notices
- Resolved Issues
- Known Issues and Workarounds

Introduction

OVF Tool 4.5.4 for vCloud Director 10.4 | 27 September 2023 | developer.vmware.com/tools

Patch release following vSphere 8.0 GA | Last document update 27 September 2023

Check for additions and updates to these release notes.

About the OVF Tool

The OVF Tool 4.5.4 is a bug fix release to support vCloud Director 10.4. The previous OVF Tool 4.5.0 release was for vSphere 8.0 GA.

VMware OVF Tool is a command-line utility that allows import and export of OVF packages to and from virtual machines running on VMware virtualization platforms. OVF Tool gets called internally by many VMware products.

Before You Begin

You can download the OVF Tool for installation on Windows 64-bit or 32-bit, Linux 64-bit or 32-bit, and Mac OS X 64-bit. The OVF Tool landing page provides a link to the downloads for each release.

OVF Tool 4.5 supports the following operating systems:

- Windows 11 32-bit (x86) and 64-bit (x86_64)
- Windows 10 32-bit (x86) and 64-bit (x86_64)
- Windows 8.1 and Windows 7 SP1, 32-bit and 64-bit
- Windows Server 2022, 2019, 2016, and 2012 R2
- MacOS versions including 11 Big Sur and 12 Monterey
- Red Hat Enterprise Linux (RHEL) recent releases
- Recent releases of CentOS and Fedora
- SUSE Linux Enterprise Server (SLES) recent releases
- Ubuntu Linux and variants, recent releases
- VMware Photon OS and Oracle Linux

OVF Tool 4.5.0 added support for vCloud Director (VCD) 10.4, and was tested with VCD 10.2, 10.3, and 10.4.

What's New

No new features, but three issues found with vCloud Director, and one with VI targets, have been fixed, as described below. For a summary of new features in the 4.5.0 release, see the *OVF Tool 4.5 Release Notes*.

Compatibility Notices

When customers try to install vCenter Server 7.0.x from a browser on MacOS 10.15 Catalina, 11.x Big Sur, or later MacOS, a popup dialog appears saying “vcsa-deploy.bin cannot be opened because the developer cannot be verified” and installation fails with error “ovftool cannot be opened because the developer cannot be verified.” This is due to greater security in MacOS and OVF Tool not being notarized for Apple. See KB 79416 for workarounds.

New: In this release, OpenSSL libraries were updated to version 1.0.2zi.14602, Curl to version 8.2.1.9716, and c-ares DNS to version 1.19.1.4234.

Resolved Issues

These issues were fixed in OVF Tool 4.5.4:

- **New: Unable to import (not found) vApp Template into Catalog.**

When the vApp Template catalog had more than 25 pages, OVF Tool could not locate catalogs past 25 due to its one page restriction. In that case customers encountered the error “Catalog not found.” In this release OVF Tool adds support for multiple catalog pages, so the vApp Template can be located on any subsequent catalog page.

- **New: Case-sensitive scanning of HTTP headers sometimes failed to match.**

  When parsing vCloud Director headers, OVF Tool used case sensitive comparison to find relevant items. In some cases, keywords in HTTP header were capitalized unexpectedly, so comparison failed and key values were missed. The fix was to change keyword field search to be case insensitive.
- **New: Omitting password caused vCloud Director URL to throw HTTP 403 error.**

  When users typed name but not password in the source or destination vCloud Directory URL, an HTTP 403 error page resulted, so OVF Tool was not able to access the vApp for download. All other org/vdc operations worked until the point of fetching the vApp. This issue occurred only when username was supplied but password was not. When OVF Tool prompted for the password, the previously supplied username was lost, or in bearer mode, authorization was reset from bearer to basic authentication mode. In either case, vCloud Directory returend a 403 error. The fix was to prevent authorization mode change during HTTP transactions.
- **New: Host selection always picks first one from a list of hosts.**

  When multiple hosts are specified in a resource pool, OVF Tool always picked the first, and would invalidate other hosts in the list. In this release OVF Tool picks a random host in the list and other hosts remain valid. This change only effects VI targets that have not specified a specific host, when there are multiple hosts in a target cluster and OVF Tool must choose.

Known Issues and Workarounds

This issue was discovered during customer testing:

- **Network services library required.**

  OVF Tool calls the network services library libnsl but Red Hat Enterprise Linux 8 and Fedora 27 removed this library. Export fails and message “export failed: unknown error” appears. The solution is to install libnsl and restart the system, as detailed in VMware KB 89515.

This issue remains from previous OVF Tool releases:

- **DMG installer for MacOS not notarized.**

  VMware did not notarize the MacOS installer, so security override is required. An alternative is to unpack the Zip file and run ovftool out of the working directory.

VMware OVF Tool 4.5.0 Release Notes

This document contains the following sections

- Introduction
- About the OVF Tool
- Before You Begin
- What's New
- Compatibility Notices
- Resolved Issues
- Known Issues and Workarounds

Introduction

OVF Tool 4.5.0 for vSphere 8.0 | 11 October 2022 | Tool Resource on code.vmware.com

Table continued on next page

Continued from previous page

Initial Availability became GA on 8 November 2022 | Document updated 13 December 2022

Check for additions and updates to these release notes.

About the OVF Tool

The OVF Tool 4.5.0 release coincides with vSphere 8.0. The previous OVF Tool 4.4.3 release was for vSphere 7.0 Update 3.

VMware OVF Tool is a command-line utility that allows import and export of OVF packages to and from virtual machines running on VMware virtualization platforms. OVF Tool gets called internally by many VMware products.

Before You Begin

You can download the OVF Tool for installation on Windows 64-bit or 32-bit, Linux 64-bit or 32-bit, and Mac OS X 64-bit. The OVF Tool landing page provides a link to the downloads for each release.

OVF Tool 4.5 supports the following operating systems:

- Windows 11 32-bit (x86) and 64-bit (x86_64)
- Windows 10 32-bit (x86) and 64-bit (x86_64)
- Windows 8.1 and Windows 7 SP1, 32-bit and 64-bit
- Windows Server 2022, 2019, 2016, and 2012 R2
- MacOS versions including 11 Big Sur and 12 Monterey
- Red Hat Enterprise Linux (RHEL) recent releases
- Recent releases of CentOS and Fedora
- SUSE Linux Enterprise Server (SLES) recent releases
- Ubuntu Linux and variants, recent releases
- VMware Photon OS and Oracle Linux

See the *OVF Tool 4.4.x Release Notes* for previous system requirements and a summary of new features in each release.

What's New

In this release, Zip archives are provided for five platforms, as an alternative to MSI, DMG, and bundle installers. Windows and Linux have both 32-bit and 64-bit Zip archives, MacOS 64-bit only. Simply unpack the Zip into a directory and run ovftool from there.

A new Pull mode feature allows much faster deployments of OVF packages. The --pullUploadMode flag introduces a new mode for HTTP(S) sources where ESXi directly copies the files from source to itself. Restrictions: the ESXi host must have access and permission to view the source, ESXi might require the source server's security certificate (or a thumbprint) and source must be HTTP(S) based rather than filesystem based.

A new upload validation feature adds the option to check and validate the SHA1 digest of a file transferred to a VI Target against the SHA1 digest of the source file being transferred. File upload validation occurs when you specify the --verifyViTargetManifest flag. If you specify the verify flag, SHA1 takes effect for SHA calculations; of course this can happen only on hosts that support SHA1. This option is not compatible with the Pull mode feature above.

The *OVF Tool User Guide* describes various ways of adding a virtual TPM (vTPM) into an OVF file. Support for vTPM in the vSphere Client is expected in an upcoming release.

Support for vCloud Director (VCD) 10.4 has been added. OVF Tool 4.5 was tested with VCD 10.2, 10.3, and 10.4.

For this release, the c-ares DNS library was upgraded to version 1.17.2, the Open SSL library was upgraded to version 1.0.2zf, and the Curl library was upgraded to version 7.83.1.

Compatibility Notices

When customers try to install vCenter Server 7.0.x from a browser on MacOS 10.15 Catalina, or Mac OS 11.x Big Sur, a popup dialog appears saying “vcsa-deploy.bin cannot be opened because the developer cannot be verified” and installation fails with error “ovftool cannot be opened because the developer cannot be verified.” This is due to greater security in MacOS and OVF Tool not being notarized for Apple. See KB 79416 for workarounds.

Resolved Issues

These issues were fixed in OVF Tool 4.5:

- **Optimized VI target network lookups.**

  Some customer environments contained thousands of networks in their inventory, so OVF Tool took several minutes looping through to validate target networks. Network validation has been optimized for this release.
- **Chunking feature for image files such as NVRAM.**

  The chunking feature did not always work for image files, so if chunksize is mentioned, OVF Tool fixes the size of NVRAM and other image files in the rewritten OVF file.

Known Issues and Workarounds

These issues were reported by customers or discovered during testing:

- **New: Error: No common hosts found.**

  When deploying to a vCenter Server and selecting a vSphere Cluster in the target URL, if the target network specified in network mapping has many hosts attached to it, OVF Tool might return the error “No common hosts found” mistakenly. (See “Setting OVF Network Maps When Deploying to vSphere” in the *OVF Tool User Guide*.) A fix has been identified and will be available in a future release. For now, the workaround is to specify, after cluster name, a host with access to both the desired network and datastore. For example, replace the first line below with the second line. You can also specify a standalone host (one not in a cluster) as in line 3.

  vi://

  vCenterServerAddress/

  DatacenterName/

  host/

  ClusterName

  vi://*vCenterServerAddress*/*DatacenterName*/**host**/*ClusterName*/*HostName*

  vi://*vCenterServerAddress*/*DatacenterName*/**host**/*HostName*

- **Network services library required.**

  OVF Tool calls the network services library libnsl but Red Hat Enterprise Linux 8 and Fedora 27 removed this library. Export fails and message “export failed: unknown error” appears. The solution is to install libnsl and restart the system, as detailed in VMware KB 89515.

This issue remains from previous OVF Tool releases:

- **DMG installer for MacOS not notarized.**

  VMware did not notarize the MacOS installer, so security override is required. An alternative is to unpack the Zip file and run ovftool out of the working directory.

An Introduction: Getting Started with vSphere APIs and SDKs

After introductory chapters, this book presents code examples that show how to use the VMware vSphere® Management SDK and the vSphere Automation™ SDK. Code examples demonstrate connection and authentication, invocation of methods, and simple manipulation of remote objects (tags) stored on the server.

VMware provides APIs for managing different aspects of VMware vCenter Server® and the data center. This publication describes APIs in vSphere release 8.0 and later.

This book combines *An Introduction to vSphere APIs and SDKs* and *Getting Started with vSphere APIs and SDKs*. The former gave a high-level comparison of facilities for managing vSphere, the latter presented code samples in various programming languages.

The following table provides the update history of this book.

Revision	Description
4 April 2024	Combined Introduction and Getting Started manuals.
26 March 2024	New section about unified session for vSphere 8.0 Update 3 release.
24 October 2023	Add a PowerCLI chapter: PowerCLI Access to vSphere APIs
05 October 2023	Add Cygwin instructions to REST Access to vSphere APIs.
21 September 2023	New publication with vSphere 8.0 Update 2 release.

Intended Audience

This information is intended for anyone who wants to automate workflows using various vSphere APIs.

Overview of vSphere APIs and SDKs

VMware vSphere® is VMware's virtualization platform, which transforms data centers into aggregated computing infrastructures that include CPU, storage, and networking resources. vSphere manages these infrastructures as a unified operating environment, and provides you with the tools to administer the data centers that participate in that environment. VMware offers a number of powerful APIs and SDKs to help you manage your virtualization infrastructure.

VMware provides APIs for managing different aspects of VMware vCenter Server® and the data center. This publication describes APIs in vSphere release 8.0 and later.

The following table provides a quick comparison of vSphere APIs with respect to functional capabilities and language bindings. Each row of the table indicates protocol and language support for an API. Language support can be a full SDK, an add-on library, or in some cases partial library support for the API.

Table 1: vSphere APIs Compared

API	function	SOAP	JSON	Java	Python	Golang
vSphere Automation API	datacenter management advanced features		Yes	SDK	SDK	govmomi (partial)
VIM API	datacenter management basic features	Yes	Yes	SDK	pyvmomi	govmomi
vCenter Server Management API	vCenter Server Appliance admin		Yes	SDK	SDK	govmomi
Storage Policy API	storage affinity profiles	Yes		SDK	pyvmomi	govmomi

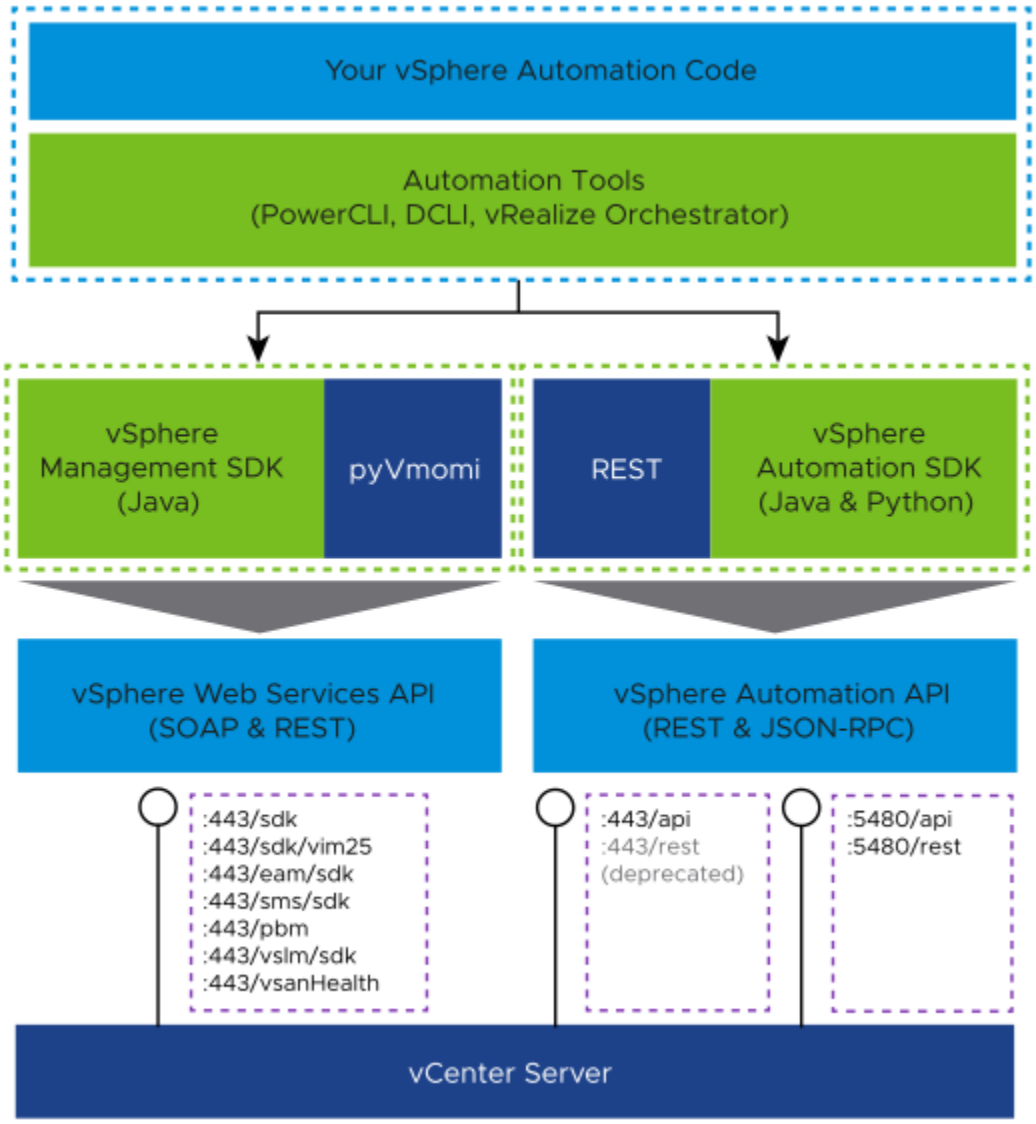
Table continued on next page

Continued from previous page

API	function	SOAP	JSON	Java	Python	Golang
Storage Monitoring	manage storage arrays	Yes		SDK	pyvmomi	govmomi
vSAN API	pooling ESXi attached storage	Yes		SDK	SDK + pyvmomi	SDK
Virtual Storage Lifecycle Management	manage standalone virtual disks	Yes		SDK		build from WSDL
ESX Agent Manager (EAM)	vSphere extensions & services	Yes		SDK	pyvmomi	govmomi
Virtual Disk Development Kit (VDDK)	operate on virtual disks, C language and Java	local	local	VADP		

For reference, the following diagram shows key service endpoints used by different vSphere APIs.

Figure 1: vCenter Server endpoints



Core vSphere APIs

VMware vSphere virtual infrastructure supports a user interface for automated management of remote vSphere assets. Many vSphere APIs address different functional areas. Two vSphere APIs support core functions of vSphere management.

The core functional areas can be viewed as configuration and management of hosts, virtual machines, and networks, as well as storage at an abstract level. The two core APIs that address these functional areas complement each other both in function and in design principles. Use the two APIs together to achieve complete functionality.

The vSphere Automation API applies REST design principles using JSON-based wire protocols, while the VIM API applies RPC principles using a SOAP protocol. In general, the Automation API offers a simpler design, while the VIM API offers better compatibility with older vSphere releases.

vSphere Automation API

The VMware vSphere Automation SDK™ is optimized for developing applications that use major vSphere features such as virtual machine management, content libraries and resource deployment, tagging, and managing internal and external security certificates.

The vSphere Automation API follows a resource-based REST architecture with native JavaScript Object Notation (JSON) requests and responses. Clients can use the following technologies to send requests to the vCenter Server Automation API endpoint:

- Java SDK
- Python SDK
- REST
- PowerCLI™

Other language bindings are available from open-source projects.

The vSphere Automation API currently includes core features such as these:

- Federated authentication with support for AD FS and Okta.
- The VMware vSphere Lifecycle Manager™ feature supports desired-state management of hosts in a cluster.
- The content library feature manages templating, storage, and deployment of virtual machines and vApps.
- Tagging of vSphere objects with metadata that facilitates filtering and sorting of objects when managing large data centers.
- Certificate management handles certificates issued by the VMware Certificate Authority (VMCA) and third-party or custom-made certificates to your environment.
- VMware vSphere Trust Authority™ is a foundational technology that enhances workload security.

VMware implements all major new vSphere features in the Automation API.

The Automation API exposes two protocols: JSON-RPC, which is used by the Automation SDKs, and a simplified JSON protocol used with REST clients. The SDKs are available for download in the vSphere Automation SDKs.

The Automation API service endpoint is

```
https://{domain}/api/{service}
```

where `{service}` is the name of a specific service interface.

For example, you can reach the Session service through the following URL:

```
https://<vcenter_server_ip_address_or_fqdn>/api/session
```

For more information about using the vSphere Automation API, see the following publications:

- *VMware vSphere Automation API REST Programming Guide*
- *VMware vSphere Automation API Java Programming Guide*
- *VMware vSphere Automation API Python Programming Guide*
- *VMware PowerCLI User's Guide*
- *vSphere Automation REST API Reference 8.0*
- *vSphere Automation API Java 8.0*
- *vSphere Automation API Python 8.0*

vSphere Web Services (VIM) API

The vSphere Web Services (VIM) API gives programmatic access to core virtualized resources that support robust, fault-tolerant virtualized applications comprising compute, networking, and storage resources. Resource types include virtual

machines, ESXi hosts, clusters, datastores, networks, and system abstractions such as events, alarms, authorization, and plug-in extensions.

The VIM API is a mature and comprehensive management API. The API works against both ESXi and vCenter Server systems, providing access to a hierarchy of managed objects that you use to configure, monitor, and manage data center resources.

Developers have several ways to access the VIM API:
- The vSphere Management SDK contains Java bindings and samples that use SOAP access to the server.
- The pyvmomi SDK is a user-friendly Python package containing bindings and utilities for working with the APIs.
- The govmomi SDK is a comprehensive set of bindings, utilities, examples, and a CLI for Go developers.
- The vSphere Management SDK provides language-neutral WSDL code for the SOAP protocol, from which you can generate bindings for your choice of language.
- The vSphere Management SDK provides a language-neutral OpenAPI specification to which you can write code that uses a REST-like JSON protocol.

The vSphere Web Services API includes these core features:
- Data center inventory management, using a hierarchy of folders and resource objects
- A powerful server-side data retrieval feature, the Property Collector
- Host system configuration and management
- Datastore configuration and management
- Network configuration and management
- Virtual machine configuration and management, including guest OS operations
- Virtual machine encryption
- Cluster and resource pool configuration and management
- Management of events and alarms

The Web Services API uses a SOAP-XML RPC protocol. It is bundled in the vSphere Management SDK. The Management SDK provides RPC stubs for clients with Java language bindings.

The Web Services server endpoint is
`https://{domain}/sdk`

The VIM JSON protocol service endpoint is
`http://{domain}/sdk/vim25/8.0.1.0`

For more information about using the VIM API, see the following documentation:
- *vSphere Web Services SDK Developer Setup Guide*
- *vSphere Web Services SDK Programming Guide*

vSphere Functional Extension APIs

VMware supplements the core vSphere APIs with several extensions that provide specialized capabilities for managing specific areas, such as management of different types of storage or the vCenter Server Appliance™ itself.

VMware vSAN

VMware vSAN™ is a distributed layer of software that aggregates local or direct-attached capacity devices of a host cluster and creates a single storage pool shared across all hosts in the vSAN cluster.

vSAN runs natively as a part of the VMware ESXi™ hypervisor. It provides an object-oriented, distributed storage area network platform that can use either SSD or spinning disks in an ESXi host as physical media. vSAN uses dedicated VMkernel network interfaces for node-to-node communication.

vSAN supports features that require shared storage, such as HA, vMotion, and DRS, but vSAN eliminates the need for external shared storage and simplifies storage configuration and virtual machine provisioning activities.

VMware vSAN uses a software-defined approach that creates shared storage for virtual machines. It virtualizes the local physical storage resources of ESXi hosts and turns them into pools of storage that can be divided and assigned to virtual machines and applications according to their quality-of-service requirements.

The API can be used for solution integration. You can use the vSAN API to configure and monitor vSAN disk clusters and related services on ESXi hosts and vCenter Server instances. The vSAN API also handles virtual disk functions such as mounting, partitioning, secure erasing, and snapshots.

Functional features of vSAN include:
- Cluster configuration and query methods.
- Stretched cluster configuration and query methods.
- Performance, health, and diagnostics.
- Virtual disk formatting and space usage queries.
- Software upgrade handling.
- Data Protection Service (DPS) is an optional data backup and recovery feature for VMware vCloud Air that enables self-service, policy-based protection of business-critical data by backing up vApps and their associated virtual machines within dedicated or VMware vCloud® Air™ Virtual Private Cloud service types.
- Cloud Native Storage (CNS) is a VMware vSphere with VMware Tanzu™ feature that makes Kubernetes aware of how to provision storage on vSphere on-demand, in a fully automated, scalable fashion. CNS also provides visibility for the administrator into container volumes through the CNS UI within vCenter.

The vSAN API uses a SOAP-XML RPC protocol, patterned after the vSphere Web Services (VIM) API. It is bundled in the vSphere Management SDK.

The vSAN service endpoint is
`https://{domain}/vsan`

For more information about using the vSAN API, see the following publications:
- *vSAN Management SDK for Java*
- *vSAN Management SDK for Python*
- *vSAN Management SDK for .NET*
- *vSAN Management SDK for Perl*
- *vSAN Management SDK for Ruby*

Virtual Storage Lifecycle Management

Virtual Storage Lifecycle Management (VSLM) is a Web service that runs on vCenter Server as part of storage policy services (SPS). You use this API to connect to the VSLM endpoint and retrieve information about first class disks (FCD). SPS also includes storage policy based management (SPBM) and storage management service (SMS).

First class disk, also called improved virtual disk (IVD), refers to disks not associated with a virtual machine. Previously, disk operations on unattached virtual disks required you to create a dummy VM and attach the virtual disks before you could work with the disks.

For example, snapshots work on a per-VM basis, and to snapshot a single virtual disk rather than all virtual disks attached to a VM required configuration overhead. The FCD feature offers full lifecycle management of virtual storage objects, independent of any VM. VSLM APIs are able to create, delete, snapshot, backup, restore, and do other lifecycle tasks on VMDK objects without requiring VM attachment.

An FCD is identified by UUID and human readable name. The UUID is globally unique and the primary identifier. The UUID remains valid even if a snapshot is taken or its FCD relocated by XvMotion.

Use cases for FCD include App Volumes, just-in-time virtual desktops for VDI, OpenStack Cinder, Docker persistent volumes, Kubernetes nodes, and VMware Cloud Native Storage.

The VSLM API uses a SOAP-XML RPC protocol, patterned after the vSphere Web Services (VIM) API. It is bundled in the vSphere Management SDK.

The VSLM service endpoint is
`https://{domain}/vslm`

For more information about using the VSLM API, see the following API reference documentation:
- *Virtual Storage Lifecycle Management API 8.0*

Storage Monitoring Service

The Storage Monitoring Service (SMS) provides methods to retrieve information about available storage topology, capabilities, and state.

The vSphere API for Storage Awareness (VASA) permits storage arrays to integrate with vCenter for management functionality. VASA providers expose features of the physical storage devices, such as storage health status, configuration info, and storage capacity.

SMS establishes and maintains connections with VASA providers. SMS retrieves information about storage availability from the providers, and clients can use the SMS API to perform the following operations.
- Identify VASA providers.
- Retrieve information about storage arrays.
- Identify vSphere inventory entities (hosts and datastores) which are associated with external storage entities on the storage arrays.

The SMS API uses a SOAP-XML RPC protocol, similar to the vSphere Web Services (VIM) API. It is bundled in the vSphere Management SDK.

The SMS service endpoint is
`https://{domain}/sms`

For more information about using the Storage Monitoring Service, see the following API reference documentation:
- *vCenter Storage Monitoring Service API 8.0*

Storage Policy API

The Storage Policy (SPBM) API, available in the vSphere Management SDK, is an extension of the Web Services (VIM) API. Storage policies simplify the task of matching available storage to virtual machines.

A vSphere storage profile defines storage policy information that describes storage requirements for virtual machines and storage capabilities of storage providers. You use the VMware Storage Policy API to manage the association between virtual machines and datastores.

With appropriate policies, SPBM governs virtual machine placement into appropriate datastore, accounting for RAID levels, pre-zeroed allocated disk, and space efficient thin provisioned disk. Storage policies may also request specific data services for virtual disks, such as caching, replication, and deduplication.

The Storage Policy API supports the following capabilities:
- Access to vSphere storage profiles and to entities associated with profiles
- Compliance checking of profiles and entities
- Placement compatibility checking

The Storage Policy API uses a SOAP-XML RPC protocol, similar to the vSphere Web Services (VIM) API. It is bundled in the vSphere Management SDK. The Management SDK provides RPC stubs for clients with Java language bindings.

The Storage Policy service endpoint is
`https://{domain}/pbm`

For more information about using the Storage Policy API, see the following API reference documentation:
- *VMware Storage Policy API 8.0*

ESX Agent Manager

You can add functions to vSphere by developing software applications that you register as vCenter Server extensions. A vSphere Solution is an extension that registers with vCenter Server and implements some or all of the extension features of the vSphere API.

You can deploy a solution as an Open Virtualization Format (OVF) package, with optional VMware vSphere Installation Bundles (VIB). You can also install solutions by using an installer, such as Windows Installer (MSI) or RPM Package Manager. Most of the extension functions in the vSphere API are independent of the technology that you use to deploy a solution. If you deploy a solution by using OVF, you can use the vCenter Extension vService to simplify the registration of the solution with vCenter Server.

You can create two types of vSphere solutions:

- A vService is a service that a solution provides to specific applications that run inside virtual machines and vApps. A solution can provide several types of vServices. Virtual machines or vApps can have dependencies on several types of vServices.
- A vSphere ESX agent is a virtual machine and an optional vSphere Installation Bundle (VIB) that extends the functions of an ESXi host to provide additional services that a vSphere solution requires.

The vService Manager and ESX Agent Manager are pre-built solutions that are part of the vCenter Server Extensions functionality within the vCenter Server.

vSphere ESX Agent Manager is an intermediary between vCenter Server and a solution with two key aspects:

- provisioning agent virtual machines and VIB modules for the solution onto a scope in vCenter Server
- monitoring changes to the agent virtual machines and scope in vCenter Server, and reporting them back to the solution

vSphere ESX Agent Manager exposes a SOAP-based API similar to the VIM API, that solutions use to register and monitor agents. The SDK includes a Model-View-Controller framework and library functions for developing vSphere extensions that interface with the vSphere Web Services (VIM) API. It is bundled in the vSphere Management SDK.

The EAM service endpoint is

`https://{domain}/eam/sdk`

For more information about using the ESX Agent Manager, see the following publication:

- *Developing and Deploying vSphere Solutions, vServices, and ESX Agents*

vCenter Server Management API

You can use the vCenter Server Management API to automate management tasks for a vCenter Server Appliance. Rather than manage the data center infrastructure, this API manages the server instance itself.

Features of the vCenter Server Management API include:
- Installation and Setup
- Configuration
- Patching and Upgrade

- Backup and Restore

The vCenter Server Management API uses a REST JSON RPC protocol, associated with the Automation API. It is available for download in the vSphere Automation SDKs.

For most operations, the vCenter Server Management API uses the service endpoint
`https://{domain}/api/{service}`

where `{service}` is the name of a specific service interface.

During setup, the Management API uses the service endpoint
`https://{domain}:5480/api/{service}`
until networking is configured and the service is available on port 443.

For more information about using the vCenter Server Management API, see the following publications:
- *VMware vSphere Automation API REST Programming Guide*
- *VMware vSphere Automation API Java Programming Guide*
- *VMware vSphere Automation API Python Programming Guide*
- *vSphere Automation REST API Reference 8.0*
- *vSphere Automation API Java 8.0*
- *vSphere Automation API Python 8.0*

vCenter Single Sign-On

The VMware vCenter Single Sign-On API simplifies secure communications with the vCenter Single Sign-On service.

The vCenter Single Sign-On server provides a Security Token Service (STS). A token uses the Security Assertion Markup Language (SAML), which is an XML encoding of authentication data.

vCenter Single Sign-On authentication can use the following identity store technologies:

- Windows Active Directory
- OpenLDAP (Lightweight Directory Access Protocol)
- Local user accounts (vCenter Single Sign-On server resident on the vCenter Server machine)
- vCenter Single Sign-On user accounts

This API defines a set of request operations that correspond to the WS-Trust 1.4 bindings:
- Issue – Obtains a token from a vCenter Single Sign-On server.
- Renew – Renews an existing token.
- Validate – Validates an existing token.
- Challenge – Part of a negotiation with a vCenter Single Sign-On server to obtain a token.

The vCenter Single Sign-On SDK includes Java bindings for the vCenter Single Sign-On WSDL. The SDK also contains sample code that demonstrates client-side support for the WSSecurityPolicy standard. Security policies specify the elements that provide SOAP message security. To secure SOAP messages, a client inserts digital signatures, certificates, and SAML tokens into the SOAP headers for vCenter Single Sign-On requests. The Java sample includes a JAX-WS implementation of SOAP header methods that support the vCenter Single Sign-On security policies.

The vCenter Single Sign-On SDK is bundled in the vSphere Management SDK.

The Single Sign-On service endpoint is
`https://{domain}/STS/STSService`

For more information about using the vCenter Single Sign-On API, see the following documentation:

- *vCenter Single Sign-On Programming Guide*

Virtual Disk API

The Virtual Disk API provides methods for backup and recovery, cloning, and other operations on virtual disks.

The Virtual Disk API can connect to local or network virtual disks, either for ESXi or hosted applications.

The Virtual Disk API supports the following functions:
- Create, open, close, and delete virtual disks
- Read and write both data and metadata
- Grow, shrink, and defragment virtual disks
- Manage virtual disk snapshot chains
- Quiesce virtual machine access and flush buffers
- Clone virtual disks

The Virtuas Disk API is provided as a C language library, running in-process as part of user-defined utility software.

The Virtual Disk API is available as part of the Virtual Disk Development Kit (VDDK).

For more information about the Virtual Disk API, see the *Virtual Disk Development Kit Programming Guide*.

vSphere Add-On SDKs

Add-on API layers are available for some popular vSphere APIs. Open-source communities can contribute new language bindings or usability improvements.

pyVmomi

pyVmomi is an open-source Python binding for several vSphere SOAP-based APIs.

The pyVmomi package provides a convenience layer on top of the core VIM API and the following related extension APIs that use the SOAP/XML protocol:
- ESX Agent Manager
- Storage Policy API
- Storage Monitoring Service (SMS)

pyVmomi is notable for:
- Simplified approach to connection, authentication, and retrieval of the ServiceContent object
- Module to handle client-server synchronization and asynchronous task tracking
- Convenient retrieval of managed object properties
- Facility to interface with vCenter Single Sign-On

For more information about using the pyVmomi, see the following documentation:
- http://vmware.github.io/pyvmomi-community-samples/#getting-started

govmomi

govmomi is a Go language library for interacting with the VMware Web Services API for ESXi and vCenter Server.

The code in the govmomi package is a wrapper for the code that is generated from the vSphere API description. It primarily provides convenience functions for working with the vSphere API:
- Generated types, managed objects, and methods for the complete VIM API
- Client structure that abstracts a session and its immutable state
- Session Manager abstraction that allows a user to login and logout

- Wrappers for a selection of managed objects
- Generated coverage for some other SOAP-based APIs, such as SMS, SPBM, and vSAN
- Partial coverage for the vSphere Automation API

For more information about using govmomi, see the following documentation:
- https://pkg.go.dev/github.com/vmware/govmomi

Getting Started with vSphere APIs and SDKs

This chapter contains a preliminary section about required resources, and then sections providing a summary of authentication options.

Subsequent sections describe how to set up a test environment for various SDKs so you can run sample programs that:

1. Open a connection to the server and create an authenticated session.
2. Create a 'Hello' folder in a data center.
3. Create a tag category.
4. Create a tag within the category.
5. Attach the tag to the folder.

These operations are implemented in several languages, including a shell script that uses **`curl`** requests to accomplish these steps.

Although these code examples create several objects, they do not delete anything. You can verify the results by using the VMware vSphere Client™, then delete the sample objects manually if you want to run another example.

> **NOTE**
> The code examples in this book use basic authentication and omit server certificate verification for simplicity. In a production environment you should use stricter security practices.

Preliminary Resources for vSphere Programming

This topic explains the vSphere resources you need to access when you run the examples provided in subsequent chapters.

To run the API examples shown in this book, you need access to a test installation of vSphere and a development workstation or virtual machine, including:
- vCenter Server
- Data center storage space to create virtual machines
- A user role with permissions on vCenter Server that allow you to create folders and tag objects in the vSphere inventory.
- A developer workstation, with space to download & install SDKs

The example code creates an inventory object folder and a tag for the folder. You can run these examples in your on-premises data center.

If you do not have a vSphere test environment readily available, you can easily create a test environment with VMware Cloud™ on AWS. Refer to the following documentation:

Getting Started With VMware Cloud on AWS

Alternatively, you may be able to install vCenter Server as a virtual machine running on VMware Workstation™ or VMware Fusion®. This is not a supported feature, but some users have used the Open Virtualization Format Tool to set up test

installations using the vCenter Server Appliance™ running on Workstation or Fusion. If you choose this method, the following links are helpful:

https://williamlam.com/2022/10/how-to-deploy-the-vcenter-server-appliance-vcsa-8-0-to-vmware-fusion-or-workstation.html

https://developer.broadcom.com/tools/open-virtualization-format-ovf-tool/latest/

Authentication with vSphere APIs

This chapter provides an overview of authentication options for vSphere users. You can use either basic or token-based authentication to access vSphere APIs. Because it is the simplest method, this guide uses basic authentication to access vSphere APIs in the following chapters.

You can use both the vSphere Management SDK and the vSphere Automation API to manage your vSphere environment. With both SDKs, you gain access to a wider range of supported automation features. Use the:

- vSphere Management SDK for host management, virtual machine provisioning, data center monitoring, virtual storage management, and more.
- vSphere Automation SDK for configuration, authentication, certificate management, tagging, life cycle management, content libraries, Kubernetes cluster management, and more.

Choosing Your Identity Source

To administer vSphere users and groups, you must first choose your identity source. You can either:

Use the vCenter Server built-in identity source

vSphere comes with a built-in identity provider, called vCenter Single Sign-On. By default, vCenter Server uses the `vsphere.local` domain as the identity source (but you can change this domain during installation). A vCenter Single Sign-On administrator can manage users and groups in the `vsphere.local` domain from the vSphere Client. You can configure the vCenter Server built-in identity provider to use Active Directory (AD) as its identity source using LDAPS, OpenLDAP, and Integrated Windows Authentication (IWA). Such configurations allow customers to log into vCenter Server using their Active Directory or LDAP accounts.

Use an external identity provider as the identity source

In vSphere 7.0 and later, you can outsource identity management to an external identity provider through federated authentication. You replace vCenter Server as the identity provider and the administration of users and groups occurs on the external identity source. When you configure vSphere to use an external identity provider, the external identity provider interacts with the identity sources on behalf of vCenter Server.

You can federate vCenter Server to the following external identity providers:

- Active Directory Federated Service (ADFS), starting in vSphere 7.0
- Okta, starting in vSphere 8.0 Update 1
- Microsoft Entra AD, starting in vSphere 8.0 Update 2

For more information on vSphere identity sources and identity management, see the topic "Authentication Mechanisms" in the *`vSphere Automation REST API Programming Guide`*.

Obtaining an Identifier and Unified Session

Before using the API services, principals must first obtain session identifiers for the vSphere Management API and the vSphere Automation API. Valid session identifiers guarantee identity of the principals and establish trust with the API endpoints.

To obtain a session identifier for the vSphere APIs, you can use different authentication mechanisms, as covered below. Once obtained, the session identifier is sent with every HTTPS request, or added to the security context for authentication.

Before vSphere 8.0 Update 3, separate sessions were required for vSphere Management API and for vSphere Automation API. As of vSphere 8.0 U3, unified sessions are supported. Now, developers can re-use a vSphere Management (VIM) session for vSphere Automation (vAPI) calls, and vice-versa. See Unified Management and Automation Session for details and code examples.

Choosing Your Authentication Method

To obtain a session identifier, you can choose among several authentication mechanisms depending on your needs and requirements. Simply stated, these mechanisms can be divided in two major groups: basic authentication and token-based authentication.

Basic Authentication

With basic authentication, you supply your vCenter Server or external provider user name and password when connecting to the vSphere Management or vSphere Automation endpoints.

- If you use the built-in vCenter Server identity source, you connect with your local domain credentials.
- With an external identity source, use your identity provider credentials to connect to the vSphere API endpoints. This workflow is possible thanks to the OAuth 2.0 Password grant type.

For more information, see these topics:

- "Authenticate to vCenter Server with vCenter Single Sign-On Credentials" in the *VMware vSphere Automation REST API Programming Guide*.
- "Authenticate by Using the Password Grant Type" in the *VMware vSphere Automation REST API Programming Guide*.
- "Create a vSphere Automation Session with User Credentials" in the *VMware vSphere Automation SDKs Programming Guide*.

NOTE
VMware encourages you to employ token-based authentication, because it is more secure and flexible.

Token-Based Authentication

With token-based authentication, principals issue a token request to the vCenter Single Sign-On built-in service. In response, vCenter Single Sign-On issues a Security Assertion Markup Language (SAML) token that principals use to obtain session identifiers for the vSphere Management API and the vSphere Automation API.

Obtaining a SAML token

You can obtain a SAML token by using either the `Issue` service of the vSphere Management API or the `AuthenticationToken` service of the vSphere Automation API. For more information, see these topics:

- "Authenticate to an AD FS-Federated vCenter Server by Using the Authorization Code Grant Type" in the *VMware vSphere Automation REST API Programming Guide*.
- "Acquiring a SAML Token from a vCenter Single Sign-On Server" in the *vCenter Single Sign-On Programming Guide*
- "Retrieve a SAML Token" in the *VMware vSphere Automation SDKs Programming Guide*

IMPORTANT
If you are using an external identity provider as the identity source, you must first obtain a JSON Web Token (JWT) from the identity provider and then exchange it for a SAML token. For more information, see topic "Authenticate to an AD FS-Federated vCenter Server by Using the Authorization Code Grant Type" in the *VMware vSphere Automation REST API Programming Guide*.

Creating a Session with the SAML Token

After you receive the SAML token, you use it to obtain a session identifier from the vSphere API endpoints.

- For the vSphere Management API, use the `LoginByToken` service (in `SessionManager`).
- For the vSphere Automation API, use the `CISSession` service.

Once you have obtained a session identifier, you can use it for authentication in your subsequent calls or in your security context.

> **NOTE**
> Before vSphere 8.0 U3, you had to create separate authentication sessions for the vSphere Management API and the vSphere Automation API. That is, you needed a separate session identifier for each of the APIs. With unified session, this is no longer necessary.

For more information, see chapter "Authentication Mechanisms" in the *VMware vSphere Automation REST API Programming Guide* and "LoginByToken Example" in the *vCenter Single Sign-On Programming Guide*.

Unified Management and Automation Session

As of vSphere 8.0 Update 3, developers can re-use a vSphere Management (VIM) session for making vSphere Automation (vAPI) calls, or re-use a vAPI session for making VIM calls. Examples follow.

Here is Python code to establish a vSphere Automation (vAPI) session to then invoke a vSphere Management (VIM) call:

```
import requests

from pyVim.connect import SmartConnect

from vmware.vapi.vsphere.client import create_vsphere_client


vc_host = '10.168.203.112'

user = 'Administrator@vsphere.local'

password = 'NKCF_n4PagP0jmz_'


# vAPI setup

session = requests.session()

session.verify = False


client = create_vsphere_client(vc_host, user, password, session=session)


# Acquire session_id

session_id = client.get_session_id()


# Reuse session_id in SmartConnect

si = SmartConnect(host=vc_host,

                  port=443,

                  disableSslCertValidation=True,

                  sessionId=session_id)
```

```
content = si.RetrieveContent()
dc_moref = content.rootFolder.CreateDatacenter("TEST")
print(dc_moref)
```

Here is Python code to establish a vSphere Management (VIM) session to then invoke a vSphere Automation (vAPI) call:

```
import requests
from pyVim.connect import SmartConnect
from vmware.vapi.vsphere.client import create_vsphere_client

vc_host = '10.168.203.112'
user = 'Administrator@vsphere.local'
password = 'NKCF_n4PagP0jmz_'

# Login with pyvmomi
si = SmartConnect(host=vc_host,
                  port=443,
                  user=user,
                  pwd=password,
                  disableSslCertValidation=True)

# TODO GetSessionId documentation
session_id = si._GetStub().GetSessionId()

# vAPI setup
session = requests.session()
session.verify = False

client = create_vsphere_client(vc_host, session=session, session_id=session_id)
print(client.vcenter.compute.Policies.list())
```

Here are Curl commands to establish a VIM session via JSON, make a vAPI call, and then call the REST tag API. The second and third are get requests and do not apply anything. The second command returns a list of folders. The third returns a list of tags for folder `group-d3`.

```
$ curl "https://$VC/sdk/vim25/8.0.2.0/SessionManager/SessionManager/Login"
  -H 'Content-Type: application/json'
  -d '{"userName": "Administrator@vsphere.local", "password": "$PSWD"}' -ki
$ curl https://$VC/api/vcenter/folder -H "vmware-api-session-id: $VIM_SESSION" -k
$ curl https://$VC/sdk/vim25/8.0.2.0/Folder/group-d3/tag
  -H "vmware-api-session-id: $VIM_SESSION" -k
```

Here are Curl commands to create a REST session and use it to call the VIM tag API:

```
$ curl -X POST https://$VC/api/session -u "administrator@vsphere.local:$PSWD" -k
$ curl "https://$VC/sdk/vim25/8.0.2.0/Folder/group-d1001/tag"
  -H "vmware-api-session-id: $VAPI_SESSION" -k
```

The following caveats and limitations apply to unified VIM/vAPI sessions:

- Session timeout can be configured with property `config.vmacore.soap.sessionTimeout` in file `/etc/vmware-vpx/vpx.conf` on vCenter Server, or using the vSphere Client UI.
- Default session lifespan can be configured with property `vpx.sso.sts.tokenLifetime`, also in `vpx.conf`, or using the vSphere Client UI.
- The maximum number of inflight sessions can be configured with property `config.vmacore.soap.maxSessionCount`, also in `vpx.conf`, or using the vSphere Client UI.
- Session IDs from either endpoint work in the other endpoint. Clients that want the benefit of single session should adjust to accept variable size Session ID tokens.

Python Access to vSphere APIs

For the Python developer, this chapter shows how to install the Python SDKs and run example programs against vSphere APIs.

VMware vSphere offers two important programming interfaces: (1) Web Services API in the vSphere Management SDK, which is SOAP based and the integration point for `pyvmomi`. (2) vSphere Automation API, which is primarily REST based but has additional bindings for Python and Java. SOAP interfaces were first used decades ago to implement the vSphere Client UI. The Automation API is newer with additional features and some duplication. Code samples in this chapter use them together.

Prerequisite: Python 3.8 or higher. Recent Linux distributions and MacOS include higher Python versions. The command name is probably **`python3`** but can be aliased.

Installing the Python SDKs for vSphere

Python access to vSphere APIs is most convenient with the `pyvmomi` library. To install `pyvmomi`:

1. If not already installed, get the Python installer **`pip`**. On Ubuntu systems, the package is named `python-pip`. On MacOS there are several alternatives; one is to download and run **`get-pip.py`**. In any case, add **`pip`**'s location to your PATH.
2. Update **`pip`** and setup tools, then install (or upgrade) `pyvmomi` with the **`pip`** command.

   ```
   pip install --upgrade pip
   pip install --upgrade setuptools
   ```

```
pip install --upgrade pyvmomi
```

3. Install the vSphere Automation SDK for Python from its Github archive. The expression `git+https` means to use secure HTTP for downloading the Github install.
   ```
   pip install --upgrade git+https://github.com/vmware/vsphere-automation-sdk-python.git
   ```

   You could also download the vSphere Automation SDK for Python (as a ZIP file) from its https://developer.broadcom.com/sdks landing page. Install instructions, including for a secure air-gapped environment, are available in the Quick Start Guide section of README.md on the http://github.com/vmware/vsphere-automation-sdk-python web page.
4. Now you are ready to run the code samples below. Copy and paste the code into a file and run the **`python`** or **`python3`** command on the file. You can run one at a time, or all three copy-and-pastes together at once.

Basic Authentication Using pyvmomi

The `pyvmomi` SDK simplifies code for connecting to the VIM endpoint by using a `SmartConnect` function that accepts basic user credentials, opens a session with the server, and returns the `ServiceInstance` object. `ServiceInstance` contains references to specific managed objects needed to use VIM API features.

The `vsphere_client` object contains interface stubs for all Automation API endpoints. This code excerpt shows a function that connects to both the VIM endpoint and the Automation API endpoint and returns both the `vsphere_client` object and the `ServiceInstance` object.

```
hostname = 'vc1.example.com'

username = 'administrator@vsphere.local'

password = 'TooSecret2C!'


from pyVim.connect import SmartConnect

from vmware.vapi.vsphere.client import create_vsphere_client, VsphereClient

import requests, urllib3

import sys

from pyVmomi import vim


def connect(host: str, user: str, pwd: str, insecure: bool) -> tuple[VsphereClient,
vim.ServiceInstance]:

    session = requests.session()

    if insecure:

        session.verify = False

        urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

    print('Creating an Automation API session.')

    vsphere_client = create_vsphere_client(host, user, pwd, session=session)

    print('Creating a VIM/SOAP session.')
```

```
    si = SmartConnect(host=host,
                      user=user,
                      pwd=pwd,
                      disableSslCertValidation=insecure)
    return vsphere_client, si


(vsphere_client, service_instance) = connect(hostname,
                                             username,
                                             password,
                                             insecure=True)
```

At this point, your session is established and the session ID is stored in a cookie that will be passed along with future requests.

Creating a Hello Folder with the VIM API Using pyvmomi

The `ServiceContent` object contains a reference to the root folder of the managed object hierarchy. You can invoke the `CreateFolder()` method on the `rootFolder` object to create a subfolder named `Hello Folder`.

```
...
print('Using VIM to create Hello Folder.')
try:
  hello_folder = service_instance.content.rootFolder.CreateFolder('Hello Folder')
except vim.fault.DuplicateName as oops:
  print(oops)
  print('Please delete folder from Inventory and try again.\n')
  sys.exit(1)
print('Hello Folder MOref: ', hello_folder, type(hello_folder), '\n')


# Convert object IDs from VIM to VAPI.
def vim_moref_to_moid(moref):
  return moref._GetMoId()
...
```

Tagging the Folder with Automation API for Python

To tag a folder, start by creating a tag category, then create a tag in that category, and finally attach the tag to the folder.

```
...
# Interface factory object:
```

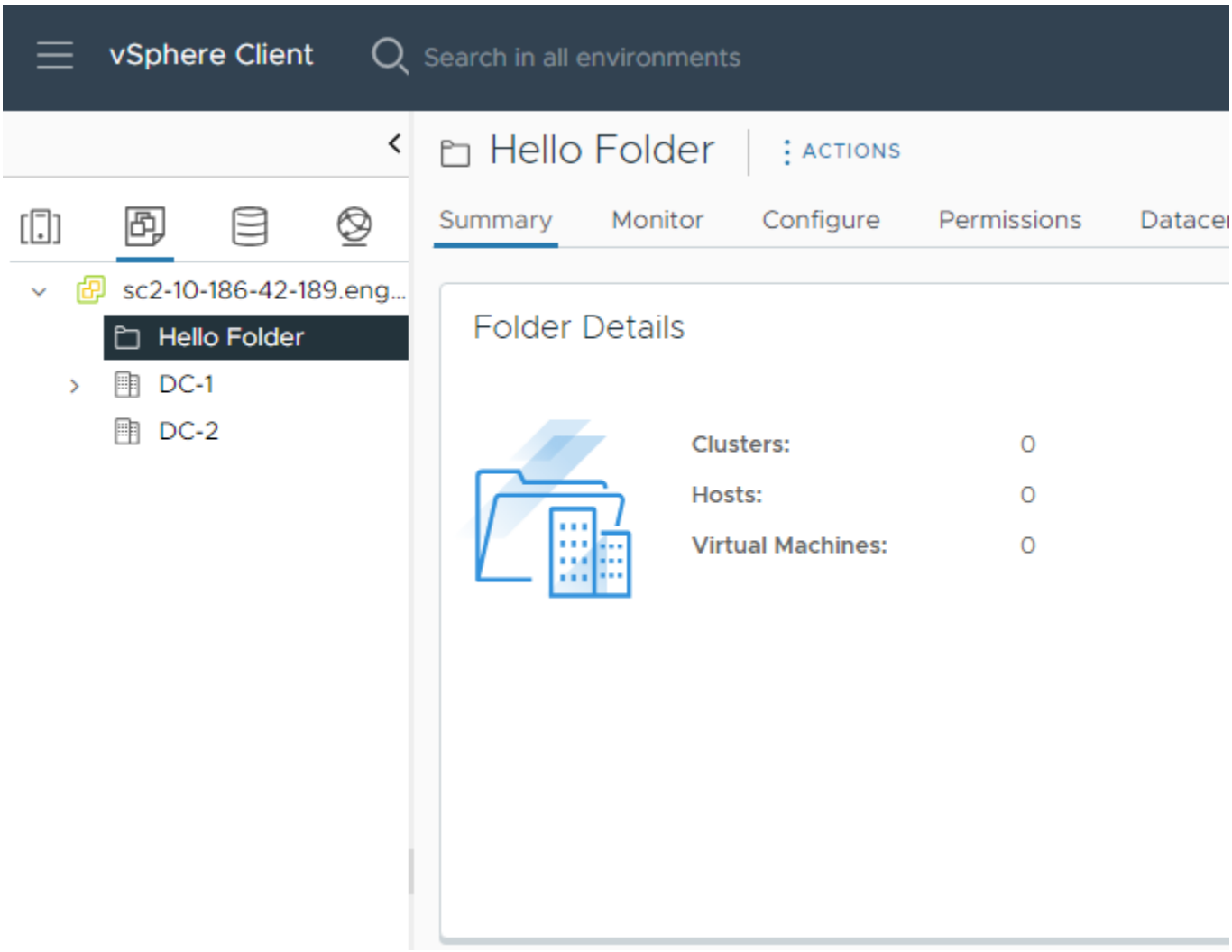
```
stub_config = vsphere_client._stub_config

print('Automation API creating a tag category.')
from com.vmware.vapi.std_client import DynamicID
from com.vmware.cis.tagging_client import Category, CategoryModel, Tag, TagAssociation
import com.vmware.vapi.std.errors_client
category_id = None
tag_id = None
category_stub = Category(stub_config)
tc_create_spec = category_stub.CreateSpec(name='hello_category',
   description = 'example of tagging category from pyvmomi',
   cardinality = CategoryModel.Cardinality.MULTIPLE,
   associable_types = set(['Folder']))
try:
  category_id = category_stub.create(tc_create_spec)
except com.vmware.vapi.std.errors_client.AlreadyExists as oops:
  print(oops)
  print('Please delete category from Tags & Custom Attributes and try again.\n')
  sys.exit(1)
print('New category created: ', category_id)

print('VAPI creating a tag in new category.')
tag_stub = Tag(stub_config)
tag_create_spec = tag_stub.CreateSpec(name='hello_tag',
description= 'example of a tag',
category_id=category_id)
try:
  tag_id = tag_stub.create(tag_create_spec)
except errors_client.NotFound as oops:
  print('Exception: \n', oops, '\n')
  sys.exit(1)
print('New tag created: ', tag_id)
```

```
print('VAPI creating association of tag with Hello Folder.')
dynamic_id = DynamicID(type='Folder',
 id=vim_moref_to_moid(hello_folder))
tag_association_stub = TagAssociation(stub_config)
tag_association_stub.attach(tag_id=tag_id,
 object_id=dynamic_id)
print('Tag attached to folder.')
```

The following image shows the Hello Folder and Category as it looks in the vSphere Client UI.



What To Do Next

You can download community-sourced `pyvmomi` examples three different ways:

- **`git clone https://github.com/vmware/pyvmomi-community-samples`**
- the Zip file: https://github.com/vmware/pyvmomi-community-samples/zipball/master
- the tarball: https://github.com/vmware/pyvmomi-community-samples/tarball/master

Java Access to vSphere APIs

For the Java developer, this chapter shows how to install the Java SDKs and run example programs against vSphere APIs.

Installing the vSphere Automation SDK and Web Services SDK for Java

The vSphere Web Services SDK, along with several other VMware SDKs, is contained in the vSphere Management SDK, available for download at https://developer.broadcom.com/sdks/vsphere-management-sdk/latest/.

- Use Java SE with JDK 11 for production environments. vCenter Server supports HTTPS connections with Java 1.8, and later, but not with Java 1.7.
- Download and install the vSphere Management SDK from the developer site. Click the **SDKs** link to reach a landing page that has links to all vSphere SDKs.
  After you download and expand the vSphere Management SDK package, the Web Services SDK is in the subdirectory `SDK/vsphere-ws`
- Download and install the Automation SDK from the developer site.
- Set the `JAVAHOME` environment variable to the root path of the Java Runtime Environment (JRE), such as `C:\Program Files\Java\jdk11.0.16_221`
- If needed, build the samples for the vSphere Web Services SDK.
  This Hello Folder example uses `wssamples.jar` and `vim25.jar` to communicate with the vSphere Web Services API.
- Install these external dependencies:
  - `slf4j-api-2.0.9.jar`
  - `jackson-databind-2.15.2.jar` and its dependencies
  - `httpcomponents-httpcore-4.4.13.jar`
  - `org-apache-commons-logging.jar`
- Set your runtime classpath to include the following:
  - `org-apache-commons-logging.jar`
  - `target/classes`
  - `vapi-runtime-2.40.0.jar`
  - `vim25.jar`
  - `wssamples.jar`
  - `vapi-authentication-2.40.0.jar`
  - `slf4j-api-2.0.9.jar`
  - `jackson-databind-2.15.2.jar`
  - `jackson-core-2.15.2.jar`
  - `jackson-annotations-2.15.2.jar`
  - `httpclient/httpclient-4.5.6.jar`
  - `httpcomponents-httpcore-4.4.13.jar`
  - `org-apache-commons-logging.jar`
  - `vcenter-bindings-4.1.0.jar`

Creating and Tagging a Folder Using Java

The vSphere Automation (VAPI) SDK and the vSphere Web Services (VIM) SDK can be used together to supplement each API with features present in the other API. This example shows:

1. Authentication with both VAPI and VIM.
2. Using VIM to create a folder as a child of the root folder in the vSphere inventory.
3. Using VAPI to create a tag category.
4. Using VAPI to create a tag in the tag category.
5. Using VAPI to associate the tag with the folder.
6. Logging out of both sessions.

NOTE
This code is for example purposes only. The example skips SSL verification and uses only basic authentication. In a production environment, both these practices are unsafe.

```
package example;


import java.security.KeyStore;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.Map;

import java.util.HashSet;

import java.util.Set;


import com.vmware.vapi.bindings.StubConfiguration;

import com.vmware.vapi.protocol.HttpConfiguration;

import com.vmware.vapi.protocol.HttpConfiguration.KeyStoreConfig;

import com.vmware.vapi.protocol.HttpConfiguration.SslConfiguration;


import vmware.samples.common.authentication.VapiAuthenticationHelper;

import vmware.samples.common.authentication.VimAuthenticationHelper;

import vmware.samples.common.SslUtil;


import org.slf4j.Logger;

import org.slf4j.LoggerFactory;


import com.vmware.vapi.bindings.StubConfiguration;

import com.vmware.vapi.protocol.HttpConfiguration;

import com.vmware.vapi.protocol.HttpConfiguration.KeyStoreConfig;
```

```
import com.vmware.vapi.protocol.HttpConfiguration.SslConfiguration;

import vmware.samples.common.authentication.VapiAuthenticationHelper;
import vmware.samples.common.authentication.VimAuthenticationHelper;
import vmware.samples.common.SslUtil;
import com.vmware.connection.ConnectedVimServiceBase;
import com.vmware.vim25.ManagedObjectReference;
import com.vmware.vim25.ServiceContent;
import com.vmware.vim25.VimPortType;

import com.vmware.cis.tagging.Category;
import com.vmware.cis.tagging.CategoryModel.Cardinality;
import com.vmware.cis.tagging.CategoryTypes;
import com.vmware.cis.tagging.Tag;
import com.vmware.cis.tagging.TagAssociation;
import com.vmware.cis.tagging.TagTypes;
import com.vmware.vapi.std.DynamicID;


public class Example() extends ConnectedVimServiceBase {
  public Example() {
    this.serverName = "vc1.example.com";
    this.userName   = "administrator@vsphere.local";
    this.password   = "TooSecret2C!";
  }

  // Server URL and credentials for authentication:
  protected String serverName;
  protected String userName;
  protected String password;

  /**
```

```
 * Builds the SSL configuration to be applied for the connection to the
 * server
 *
 * Note: Below code circumvents SSL trust if "skip-server-verification" is
 * specified. Circumventing SSL trust is unsafe and should not be used
 * in production software. It is ONLY FOR THE PURPOSE OF DEVELOPMENT
 * ENVIRONMENTS.
 */
protected SslConfiguration buildSslConfiguration() throws Exception {
    SslConfiguration sslConfig;
    /*
     * Below code enables all VIM & vAPI connections to the server
     * without validating the server certificates..
     *
     * Note: Below code is to be used ONLY IN DEVELOPMENT ENVIRONMENTS.
     * Circumventing SSL trust is unsafe and should not be used in
     * production software.
     */
    SslUtil.trustAllHttpsCertificates();
    sslConfig = new SslConfiguration.Builder()
                    .disableCertificateValidation()
                    .disableHostnameVerification()
                    .getConfig();
    return sslConfig;
}

protected HttpConfiguration buildHttpConfiguration() throws Exception {
  HttpConfiguration httpConfig =
          new HttpConfiguration.Builder()
          .setSslConfiguration(buildSslConfiguration())
          .getConfig();
  return httpConfig;
```

```
}

/**
 * Creates dual sessions with server for VIM & VAPI, using username/password.
 * @throws Exception
 */
protected VapiAuthenticationHelper vapiAuthHelper;
protected VimAuthenticationHelper vimAuthHelper;
protected StubConfiguration sessionStubConfig;

protected void login() throws Exception {
  this.vapiAuthHelper = new VapiAuthenticationHelper();
  this.vimAuthHelper = new VimAuthenticationHelper();
  HttpConfiguration httpConfig = buildHttpConfiguration();
  this.sessionStubConfig =
          vapiAuthHelper.loginByUsernameAndPassword(
              this.server, this.username, this.password, httpConfig);
  this.vimAuthHelper.loginByUsernameAndPassword(
              this.server, this.username, this.password);
}

protected void logout() throws Exception {
  this.vapiAuthHelper.logout();
  this.vimAuthHelper.logout();
}

protected ManagedObjectReference createHelloFolder() throws Exception {
  System.out.println("Use VIM API to create a Hello Folder.");
  ServiceContent content = this.vimAuthHelper.getServiceContent();
  ManagedObjectReference folderMoRef = content.getRootFolder();
  VimPortType vimPort = this.vimAuthHelper.getVimPort();
  ManagedObjectReference hello =
```

```
      vimPort.createFolder(folderMoRef, "Hello Folder");
    System.out.println("Sucessfully created Hello Folder.");
    return hello;
  }

  protected String createTagCategory() throws Exception {
    System.out.println("VAPI creating a tag category.");
    Category categoryStub =
       this.vapiAuthHelper.getStubFactory().createStub(Category.class,
                                                sessionStubConfig);
    CategoryTypes.CreateSpec createSpec = new CategoryTypes.CreateSpec();
    createSpec.setName("hello_category");
    createSpec.setDescription("example of tagging category from Java");
    createSpec.setCardinality(Cardinality.SINGLE);
    Set<String> set = new HashSet<>();
    set.add("Folder");
    createSpec.setAssociableTypes(set);
    String newCategoryId = categoryStub.create(createSpec);
    System.out.println("Created \"hello_category\".");
    return newCategoryId;
  }

  protected String createTag(String cat) throws Exception {
    System.out.println("VAPI creating a tag.");
    Tag tagStub =
       this.vapiAuthHelper.getStubFactory().createStub(Tag.class,
                                                sessionStubConfig);
    TagTypes.CreateSpec createSpec = new TagTypes.CreateSpec();
    createSpec.setName("hello_tag");
    createSpec.setDescription("example of a tag from Automation API");
    createSpec.setCategoryId(cat);
    String newTagId = tagStub.create(createSpec);
```

```
    System.out.println("Created \"hello_tag\".");
    return newTagId;
  }

  protected void associateTag(String tag, DynamicID folder) throws Exception {
    System.out.println("Using Java to associate a tag to a folder.");
    TagAssociation assocStub =
       this.vapiAuthHelper.getStubFactory().createStub(TagAssociation.class,
                                                sessionStubConfig);
    assocStub.attach(tag, folder);
    System.out.println("Associated \"hello_tag\" with \"Hello Folder\".");
  }

  public static void main(String[] args) throws Exception {
    Example example = new Example();
    example.login();
    System.out.println("Logged in to VIM and to VAPI.");
    ManagedObjectReference helloFolder = example.createHelloFolder();
    String cat = example.createTagCategory();
    String tag = example.createTag(cat);
    DynamicID folder =
                new DynamicID(helloFolder.getType(),
                              helloFolder.getValue());
    example.associateTag(tag, folder);
    example.logout();
  }
}
```

Golang Access to vSphere APIs

For the Golang developer, this chapter shows how to install the govmomi SDK and run example programs against vSphere APIs.

Installing the govmomi SDK for vSphere

To access vSphere APIs using the Go language, the `govmomi` library is most convenient. To install `govmomi`, open a terminal window and use these commands:

- mkdir govmomi
- cd govmomi
- go mod init example/govmomi
- go get -u github.com/vmware/govmomi

Basic Authentication with the VIM API Using govmomi

The `govmomi` SDK simplifies the connection code with the `govmomi.NewClient` function. The function accepts a service URL containing basic user credentials, opens a session with the server, and returns a `client` object containing the `ServiceContent` object. `ServiceContent` contains references to specific managed objects needed to use VIM API features.

```
package main


import (
        "context"
        "fmt"
        "net/url"


        "github.com/vmware/govmomi"
        "github.com/vmware/govmomi/vim25/methods"
        "github.com/vmware/govmomi/vim25/types"
        "github.com/vmware/govmomi/vapi/rest"
        "github.com/vmware/govmomi/vapi/tags"
)


// Connection parameters.
var hostname = "vc1.example.com"
var username = "administrator@vsphere.local"
var password = "TooSecret2C!"


func main() {
        fmt.Println("Creating a VIM/SOAP session.")
        vcURL := "https://" + username + ":" + password + "@" + hostname + "/sdk"
        u, err := url.Parse(vcURL)
        if err != nil {
```

```
        fmt.Printf("Error parsing url %s\n", vcURL)
        return
}
ctx, cancel := context.WithCancel(context.Background())
defer cancel()
vimClient, err := govmomi.NewClient(ctx, u, true)
if err != nil {
        fmt.Printf("Error logging in: %s\n", err.Error())
        return
}
fmt.Println("Login successful")
...
```

At this point, your session is established and the session ID is stored in the `client` object. The session ID will be passed to the server with future requests.

Creating a Hello Folder with the VIM API Using govmomi

The `ServiceContent` object contains a reference to the root folder of the managed object hierarchy. You can invoke the `CreateFolder()` method on the remote `rootFolder` object to create a subfolder named `Hello Folder`.

```
...
fmt.Println("Using VIM to create Hello Folder.")
req := types.CreateFolder{
                This: vimClient.Client.ServiceContent.RootFolder,
                Name: "Hello Folder",
}
res, err := methods.CreateFolder(ctx, vimClient.Client, &req)
if err != nil {
    fmt.Println(err)
    return
}
fmt.Println(res.Returnval.Type, "created")
}
helloFolder := res.Returnval
fmt.Println(helloFolder)
...
```

Basic Authentication with the Automation API Using govmomi

To create an authenticated session with the Automation API, you can use code like the following.

```
...
fmt.Println("Creating an Automation API session.")
vapiClient := rest.NewClient(vimClient.Client)
err = vapiClient.Login(ctx, url.UserPassword(username, password))
if err != nil {
    fmt.Println(err)
    return
}
fmt.Println("VAPI session created.")
...
```

Tagging the Hello World Folder with the Automation API Using govmomi

To tag a folder, start by creating a tag category, then create a tag in that category, and finally associate the tag with the folder.

```
...
        fmt.Println("VAPI creating a tag category.")
        m := tags.NewManager(vapiClient)
        catId, err := m.CreateCategory(ctx, &tags.Category{
            AssociableTypes: []string{"Folder"},
            Cardinality:     "SINGLE",
            Description:     "example of tagging category from govmomi",
            Name:            "hello_category",
        })
        if err != nil {
            fmt.Println(err)
            return
        }
        fmt.Println("New category created: ", catId)

        fmt.Println("VAPI creating a tag in new category.")
        tagId, err := m.CreateTag(ctx, &tags.Tag{
                      CategoryID:  catId,
```

```
                Description: "example of a tag from govmomi",
                Name:        "hello_tag",
        })
        if err != nil {
            fmt.Println(err)
            return
        }
        fmt.Println("New tag created: ", tagId)

        fmt.Println("VAPI creating association of tag with Hello Folder.")
        err = m.AttachTag(ctx, tagId, helloFolder)
        if err != nil {
            fmt.Println(err)
            return
        }
        fmt.Println("Tag attached to folder: ", tagId, helloFolder)
}
```

REST Access to vSphere APIs

This chapter shows how to run example programs by using the `vSphere Automation REST API` and the `Virtual Infrastructure JSON API`.

The example code used here is in the form of a `bash` shell script that invokes `curl` commands and captures the results in environment variables that can be used by subsequent `curl` commands. The example code creates a 'hello' folder and applies a 'hello' tag. The example is divided into sections for clarity, but you can copy and paste the sections into a single plain text file named `example.sh` and execute it with the command `bash example.sh` in the directory where you saved the file.

If you have access to a Unix or Linux development machine, you might need to use your Linux distribution's package manager to install the `jq` utility, which is used to parse JSON response bodies. If you have access to a Windows development machine, you can run the bash example by first installing the Cygwin utility, which provides a Unix-like shell, then installing the `jq` package that is available for Cygwin.

Installing Cygwin for Windows

To use this bash shell script example on Windows, install Cygwin and the jq packages:

1. Download and run the Cygwin setup executable.
2. Answer the installation configuration questions.
3. Select **Up to Date** from the **View** pull-down menu and click through to install the Cygwin base packages.
4. Run the Cygwin setup executable again.
5. Select **Full** from the **View** pull-down menu and search for `jq`.

6. For the `jq` and `libjq1` packages, change the value in the **New** column from `Skip` to the latest release, then click through to install the `jq` dependencies.

Basic Authentication to the VIM API

Starting with vSphere 8.0 Update 1, you can use the `Virtual Infrastructure JSON API` as a substitute for the SOAP-based `vSphere Management API`. The `Virtual Infrastructure JSON API` is an HTTP-based wire protocol that maps directly to the `vSphere Management API`. The `Virtual Infrastructure JSON API` follows a resource-based REST architecture with JavaScript Object Notation (JSON) requests and responses. You use the main CRUD (Create, Retrieve, Update, Delete) functions over HTTP to manage object representations.

1. To authenticate with the `Virtual Infrastructure JSON API`, you must first retrieve the `SessionManager` managed object ID by using the `Service Instance Get Content` operation.

```
SESSION_MANAGER_MOID=$(curl -k https://$VC/sdk/vim25/8.0.1.0/ServiceInstance/
ServiceInstance/content \

                        | jq .sessionManager.value \

                        | xargs -t)
```

2. Then, you use the `SessionManager Login` service to retrieve a session identifier for the `Virtual Infrastructure JSON API`.

```
VC='vc1.example.com'

UNAME='administrator@vsphere.local'

PWORD='TooSecret2C!'


curl -v -k "https://$VC/sdk/vim25/8.0.1.0/SessionManager/$SESSION_MANAGER_MOID/Login"
\

    -H 'Content-Type: application/json' \

    -d '{"userName": <myUsername>, "password": <myPassword>}'

BODY='{"userName": "'"$UNAME"'", "password": "'"$PWORD"'"}'

VIM_SESSION_ID=$(curl -k -i "https://$VC/sdk/vim25/8.0.1.0/SessionManager/
$SESSION_MANAGER_MOID/Login" \

                        -H 'Content-Type: application/json' \

                        -d "$BODY" \

                        | tr -d '\r' \

                        | awk '/vmware-api-session-id/ {print $2}' )
```

   In the response headers, you receive the `vmware-api-session-id:<mySessionIdentifier>` key-value pair. Use it in the header of subsequent API requests for authentication.

Creating a Hello Folder with the VIM API Through REST

The `ServiceContent` object contains a reference to the root folder of the managed object hierarchy. You can retrieve that reference by using the `ServiceInstance Get Content` operation. Then you invoke the `CreateFolder()` method on the `rootFolder` object to create a subfolder named `Hello Folder`.

```
echo 'Using REST to create Hello Folder.'

BODY='{"name": "Hello Folder"}'

AUTH_H="vmware-api-session-id: $SESSION_ID"

HELLO_FOLDER_MOID=$(curl -kv \

   "https://$VC/sdk/vim25/8.0.1.0/Folder/$ROOT_FOLDER_MOID/CreateFolder" \

                        -H "$AUTH_H" -H "Content-Type: application/json" \

                        -d "$BODY" \

                        | jq .object.value \

                        | xargs -t)

echo HELLO_FOLDER_MOID=${HELLO_FOLDER_MOID}
```

Basic Authentication to the vSphere Automation API

To obtain a session identifier, you call the `Create Session` service with the base-64 encoded value of your vCenter Single Sign-On user name and password separated by a colon (`username:password`) in the authorization header.

```
auth=$(printf '%s:%s' "$UNAME" "$PWORD" | base64)

VAPI_SESSION_ID=$(curl -k -i -X POST -H "Authorization: Basic $auth" "https://$VC/api/
session" \

                         | tr -d '\r' \

                         | awk '/vmware-api-session-id/ {print $2}' )
```

In the response, you receive the `vmware-api-session-id:<mySessionIdentifier>` key-value pair. Use it in the header of subsequent API calls for authentication.

Tagging the Hello World Folder with the Automation API Through REST

To tag a folder, start by creating a tag category, then create a tag in that category, and finally associate the tag with the folder.

1. Create a tag category.

```
echo 'Using REST to create tag category.'

BODY='{"name": "hello_category",

      "description": "A tag category for custom folders",

      "cardinality": "SINGLE",

      "associable_types": ["Folder"]}'

CAT_ID=$(curl -k \

             -X POST \

             -H "vmware-api-session-id: $VAPI_SESSION_ID" \

             -H "Content-Type: application/json" \

             -d "$BODY" \
```

```
              "https://$VC/api/cis/tagging/category" \
              | xargs -t)
```

The response contains the details of the newly created tag category, including its UUID.

2. Create a tag in that category. Use the tag category ID you created in the previous step.

```
echo 'Using REST to create tag.'
BODY='{"name": "hello_tag",
      "description": "example of a tag from Automation API",
      "category_id": "'${CAT_ID}'"}'
TAG_ID=$(curl -k \
              -X POST \
              -H "vmware-api-session-id: $VAPI_SESSION_ID" \
              -H "Content-Type: application/json" \
              -d "$BODY" \
              "https://$VC/api/cis/tagging/tag" \
              | xargs -t)
```

The response contains the details of the newly created tag, including its UUID.

3. Associate the tag with the *Hello World* folder. You must have the *Hello World* folder ID and the tag ID you created in the previous step.

```
echo 'Using REST to tag Hello Folder.'
BODY='{ "object_id": {"id": "'${HELLO_FOLDER_MOID}'", "type": "Folder"} }'
curl -k \
    -X POST \
    -H "vmware-api-session-id: $VAPI_SESSION_ID" \
    -H "Content-Type: application/json" \
    -d "$BODY" \
     "https://$VC/api/cis/tagging/tag-association/${TAG_ID}?action=attach"
```

On success, the operation returns HTTP Code 204 - No Content.

PowerCLI Access to vSphere APIs

This chapter shows how to run example scripts by using the PowerCLI SDK cmdlets that bind directly to the vSphere APIs.

You can use the following mechanisms to access the vSphere APIs with PowerCLI:

- For the `vSphere Management API`: use the **`Get-View`** cmdlet that allows you to manipulate VIM objects.
- For the `vSphere Automation API`: use the `vSphere Automation SDK for PowerShell` with its **`Initialize-`** and **`Invoke-`** cmdlets.

Using Get-View

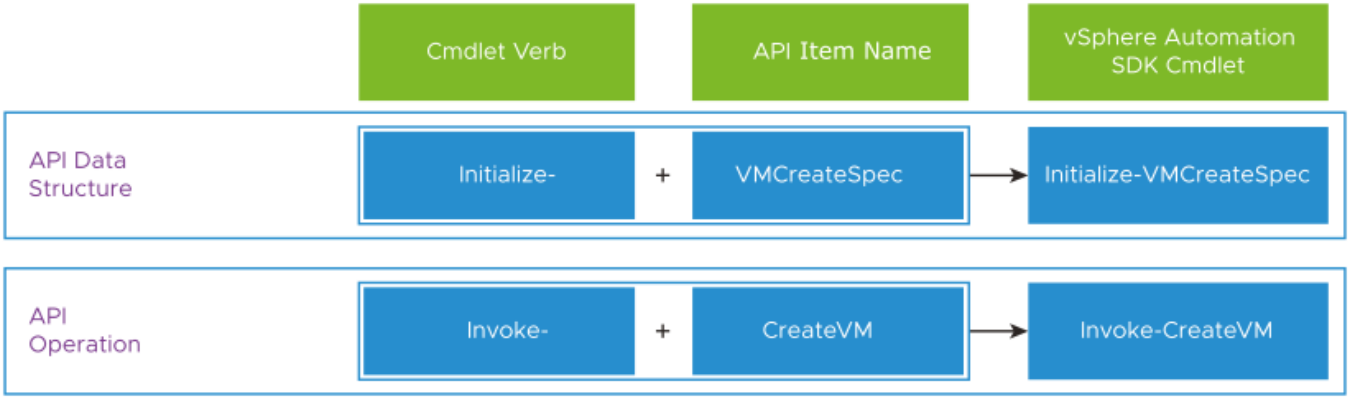
Use the Get-View cmdlet to view objects and call methods on the `vSphere Management API`. The Get-View cmdlet offers high performance by binding directly to the vSphere Management API.

Using the vSphere Automation SDK for PowerShell

Starting from VMware PowerCLI 12.4.0, you can use the `vSphere Automation SDK for PowerShell` to communicate with the `vSphere Automation API`.

You can construct `vSphere Automation SDK` cmdlets by using the `Initialize-` and `Invoke-` cmdlet verbs in combination with the names of the vSphere Automation API data structures or operations.

Figure 2: Constructing vSphere Automation SDK Cmdlets



- Use `Initialize-` cmdlets to prepare the data structures, if any, for your API calls. These cmdlets function on the client side and do not communicate with the `vSphere Automation API` server.
- Use `Invoke-` cmdlets to execute the operations on the server side.

Installing Microsoft PowerShell and VMware PowerCLI

To access vSphere APIs through cmdlets, you must have Microsoft PowerShell and VMware PowerCLI installed on your machine.

1. Verify that PowerShell is available on your system. Windows users have PowerShell pre-installed, while Linux and macOS users must install it manually. For more information, see the PowerCLI Installation Guide.
2. To install PowerCLI, open PowerShell and run the command:
   `Install-Module` `-Name`*`VMware.PowerCLI`*

Basic Authentication to the VIM API and the vSphere Automation API

With PowerCLI, you connect with a single command to the `vSphere Management API` and the `vSphere Automation API`.

1. Run the **`Connect-VIServer`** cmdlet with your credentials and you create authentication sessions to both vSphere APIs automatically.
   `Connect-VIServer` `-Server vc1.example.com -User 'MyUser' -Password 'MyPassword'`

Creating a Hello Folder with the VIM API Using PowerCLI

Because the `CreateFolder` method is a part of the `vSphere Management API`, you must use the **`Get-View`** cmdlet.

1. Create a variable with the `ServiceInstance` object view.

```
$serviceInstance = Get-View ServiceInstance
```

2. Create a variable with the `RootFolder` object view.
```
$rootFolderView = Get-View -Id $serviceInstance.Content.RootFolder
```
3. To create your folder, invoke the `CreateFolder` method on the `RootFolder` view.
```
$FolderMOID = $rootFolderView.CreateFolder('Hello Folder')
```

Tagging the Hello World Folder with the Automation API Through PowerCLI

The Tagging APIs are included in the `vSphere Automation API` as a part of the `CIS REST APIs` package (CIS stands for common infrastructure services). Therefore, you can use the `vSphere Automation SDK for PowerShell` to access the Tagging APIs with PowerCLI.

To tag a folder, start by creating a tag category, then create a tag in that category, and finally associate the tag with the folder. Each step in the tag creation and application process requires two PowerCLI verbs, both the `Initialize` and the `Invoke`.

1. Create a tag category.
```
$TaggingCategoryCreateSpec = Initialize-TaggingCategoryCreateSpec -Name
"helloCategory" -Description "A tag category for custom folders" -Cardinality
"SINGLE" -AssociableTypes "Folder"
```
```
$tagCategoryID = Invoke-CreateCategory -TaggingCategoryCreateSpec
$TaggingCategoryCreateSpec
```
   The system returns the ID of the newly created tag category.
2. Create a tag in that category. You use the tag category ID from the previous step.
```
$TaggingTagCreateSpec = Initialize-TaggingTagCreateSpec -Name "helloTag" -Description
"example of a tag from PowerCLI" -CategoryId $tagCategoryID
```
```
$tagID = Invoke-CreateTag -TaggingTagCreateSpec $TaggingTagCreateSpec
```
   The system returns the ID of the newly created tag.
3. Associate the tag with the *`Hello World`* folder. You use the *`Hello World`* folder ID and the tag ID that you created in the previous step.
```
$FolderID = Initialize-StdDynamicID -Type "Folder" -Id $FolderMOID.value
```
```
$TaggingTagAssociationAttachRequestBody = Initialize-
TaggingTagAssociationAttachRequestBody -ObjectId $FolderID
```
```
Invoke-AttachTagIdTagAssociation -TagId $tagID
-TaggingTagAssociationAttachRequestBody $TaggingTagAssociationAttachRequestBody
```

You can now see the *`Hello World`* folder with the tag and category in the vSphere Client.

Use Cases for vSphere APIs

Virtual Disk Development Kit

This SDK is most frequently used to implement backup and restore applications. This section is a walk-though of code to restore a virtual machine.

The Virtual Disk Development Kit (VDDK) is a set of libraries and utilities that enable software developers to create applications that can access and manipulate virtual disks used for storage by products such as vSphere, Workstation, and Fusion. VDDK is a valuable tool for developers of storage and backup applications.

VDDK provides APIs for backup and recovery, cloning, and other operations on virtual disks. With VDDK, developers can create applications that perform backup and restore operations for virtual machines, as well as applications that create clones or snapshots of virtual disks. VDDK also includes utilities for converting virtual disks between different formats and for creating virtual disks from scratch.

Safekeeping – a VDDK Use Case

Based on VDDK, Safekeeping is a "proof of concept" application to demonstrate backup and restore of vSphere VMs, vApps, and first-class disk. You can find its source code on Github, as linked from the developer.broadcom.com SDK landing page for VDDK. This section focuses on `RestoreVm.java`, a source code module that recovers a VM previously saved by the `BackupVm.java` module.

```
safekeeping backup vm:myTestVM

safekeeping restore vm:myTestVM
```

After package and interface imports, class `RestoreVm` extends method `AbstractRestoreFcoWithDisk`, which in turn extends method `AbstractRestoreFco` for restoring a first class object (FCO) but with disk, as in a vSphere VM. (A vApp is an FCO that includes VMs but not disk.) `RestoreVm` must be passed a VIM connection, user-specified command options, and a log file.

The `checkOptions` method parses passed-in options, including (if specified) VM name, parent vApp, data center, resource pool, datastore, and storage profile. If the destination VM folder cannot be restored to, `RestoreVM` uses a different location.

Snapshot is a typical operation for both backup and restore. The `createSnapshot` method takes a vSphere snapshot so the VM can continue running while being restored, and also to quiesce the virtual disk while it is being replaced. If a snapshot is unnecessary because the VM does not exist, snapshot is skipped.

If the VM is being restored because it no longer runs, the `destroy` method deletes the old VM so it does not conflict with the VM to be restored.

The `disksMapping` method is one of the most complicated in `RestoreVM`. It needs to know the number of disks to restore, the storage profile of disks, and the location of disk data in the backup. This information comes from the `managedInfo` object, combined with the generated `profile` object.

Once virtual disks are correctly mapped, they are restored by the `disksRestore` method. This could take a lot of time, so the `final` loop blocks until disks are restored.

If a failed VM was not deleted, the `manageExistingFco` method deals with naming conflicts between the pre-existing VM and the VM to be restored. The VM might be part of a vApp, which allows duplicate names. If the pre-existing VM is powered on, the backed-up VM cannot be restored. With duplicate names, the user can be given a chance to rename VM to be restored.

The `networksMapping` method determines what networks the backed-up VM used for each virtual NIC, and attempts to reformulate them using existing networks.

Finally the `powerOn` method starts up the restored VM so it can be configured. A VM can be powered on before the Guest OS is ready to run.

The `reconfig` method configures the VM based on profiles of its virtual disks and networks determined above. Configuration may be interrupted by a fault. Similar `reconfigDisk` and `reconfigNetwork` methods take care of disk backing and network backing for the VM to be restored. The `removeSnapshot` methods cleans up the snapshot previously created.

Finally the `restore` method, the main restore function, puts this all together. It retrieves the restore profile and calls snapshot, start-access, `disksRestore`, and end-access in an if statement. This is where the C++ functions in VixDiskLib get called (start and end access prevent vMotion during restore).

```
if (createSnapshot(rar) && startVddkAccess(rar, jvddk) && disksRestore(rar, jvddk)
                        && endVddkAccess(rar, jvddk)) {
   final String st = rar.getFcoToString() + " Restore Success - start cleaning";
}
```

The restored VM could either be newly created with the `restoreByCreate` method, or recovered with import of virtual disk data by the `restoreByImport` method. It is the `restoreMetadata` method that decides which. The `restoreManagedInfo` method takes care of items managed by vCenter Server, such as folder groupings and resource pools.

vSphere Web Services SDK Developer's Setup Guide

This book, the *vSphere Web Services SDK Developer's Setup Guide,* provides information about setting up your development environment to use the VMware® vSphere Web Services SDK.

VMware provides several different APIs and SDKs for various applications and goals. This book provides information about using the vSphere Web Services SDK for developers who are interested in creating client applications for managing VMware® vSphere components available on VMware ESXi and VMware vCenter Server systems.

To view the current version of this book as well as all VMware API and SDK documentation, go to http://www.vmware.com/support/pubs/sdk_pubs.html

Revision History

This guide is revised with each release of the product or when necessary. A revised version can contain minor or major changes. The Revision History summarizes the significant changes in each version of this guide.

Table 2: Revision History

Revision	Description
21 SEP 2023	vSphere 8.0 Update 2 release. • Minor changes for search engine optimization.
11 OCT 2022	vSphere 8.0 release. • .NET content removed.
05 OCT 2021	• Update to JDK 11 in the vSphere Management SDK. • Update certificate procedure.
12 APR 2021	VMware has rebranded the My VMware portal as VMware Customer Connect. We have updated this document accordingly.
09 Mar 2021	vSphere 7.0 Update 2 release, with Java 11 work-around.
06 Oct 2020	vSphere 7.0 Update 1 release.
02 Apr 2020	vSphere 7.0 release: Java 1.7 no longer supported.
17 Apr 2018	Minor updates for vsphere 6.7.
05 Nov 2016	Update and restructure procedures for modifying service endpoints with vSphere 6.5. Simplify process of building C# DLLs and samples.
12 Mar 2015	vSphere 6.0 release. Changed destination directory for C# DLLs.
20 Feb 2014	Corrected instructions for modifying reverse proxy configuration.
04 Dec 2013	Clarified location of Web Services SDK.
01 Oct 2013	Revised C# setup instructions to reflect simplified procedure in 5.5 release and use new tools. Corrected path to Java sample. Updated Java version. Fixed minor typographical errors in code.
09 Sep 2012	Removed Axis support for 5.1 release. Made more corrections and expansions to C# instructions.
17 Nov 2011	Corrected setup instructions for C# stubs.
24 Aug 2011	Updated for vSphere 5.0 (included information on using JAX-WS bindings).
13 Jul 2010	Minor updates for vSphere 4.1 (new WSDL file configuration, example syntax).
07 May 2009	Revised release of the *vSphere Web Services SDK Developer's Setup Guide* for vSphere Web Services SDK 4.0. Server-certificate setup information is located in the reference section. Changed directory name for WSDLFILE environment variable.
29 Nov 2007	Initial release of *vSphere Web Services SDK Developer's Setup Guide* for VMware Infrastructure SDK 2.5.

Intended Audience

This book is intended for anyone who wants to develop applications using the VMware vSphere Web Services SDK. vSphere Web Services SDK developers typically include software developers creating client applications using Java and targeting VMware vSphere.

What is the vSphere Web Services SDK

The VMware vSphere® Web Services SDK helps you develop client applications for the VMware vSphere API. Learn how to install it in this publication.

The vSphere Web Services SDK includes all the components necessary to work with the VMware vSphere API, including WSDL files, sample code, and libraries. With the vSphere Web Services SDK, developers can create client applications to manage, monitor, and maintain VMware vSphere components, as deployed on VMware® VMware vSphere® ESX®, ESXi™, and VMware® vCenter™ Server systems.

This *vSphere Web Services SDK Developer's Setup Guide* explains how to set up the development environment to create new applications with Java. This guide also includes information about running the sample applications included with the vSphere Web Services SDK.

What you need to know for the vSphere Web Services SDK

To develop applications for the vSphere Web Services SDK requires a programming language with a SOAP toolkit. You must also understand the following concepts:

- Web services technology provides operations, also known as methods in the context of client applications. Using the vSphere Web Services SDK and your choice of programming language, you can create client applications that invoke these operations to perform the full range of server-side management and monitoring tasks.
- The Web services API is defined in Web Services Description Language (WSDL) files. The WSDL files are used by client-side Web-services utilities to create proxy code (stubs) that client applications use to interact with the server.
- Client applications invoke operations by calling proxy interface methods. The client proxy encodes an operation invocation into a SOAP message and sends it to the server. Simple Object Access Protocol (SOAP) is a programming-language neutral XML format. SOAP message translation is transparent to the developer.
- Communications between client and server occur over HTTP or HTTPS. HTTPS is a secure form of HTTP that uses SSL to encrypt client-server communications. The default is HTTPS, but you can configure the VMware vSphere Web server to support HTTP. (See Endpoint Configuration for HTTP.)

You should also know about basic ESXi and vCenter Server operations. See the VMware vSphere Documentation page on the VMware Web site.

Programming Language Support in the vSphere Web Services SDK

You can develop for the vSphere API in any language from which you can generate client-side stubs from Web-services WSDL files. This section describes SDK support for Java clients.

See What files are in the vSphere Web Services SDK package for additional packaging details and for some caveats about the Java samples and for specific version requirements for the JDK, the Java API for XML Web Services libraries, and the JAXB libraries.

Language/Tool Context	Java
Development environment or framework	JDK 11 (also known as J2SE 1.11).
Web-services-client application development toolset, also known as a SOAP toolkit	JAX-WS 2.3.3 and later (Java/Jakarta XML Web Services). JAXB 2.3.2 and later (Java/Jakarta XML Binding).

SOAP toolkits are readily available for Java. Developers, scripters, and administrators can also use Microsoft PowerShell or Perl with the vSphere Web Services API through toolkits that VMware provides. For more information, see http://communities.vmware.com/community/developer.

Types of Applications That You Can Build Using This SDK

You can use the vSphere Web Services SDK to develop system administration, provisioning, and monitoring applications for VMware vSphere systems.

The VMware vSphere Client application and VMware vSphere Web Access are two examples of client applications that were developed using vSphere API. The vSphere Client is a traditional Windows client application. Web Access is a browser plug-in that is available through the Web server port on ESXi and vCenter Server systems.

With the vSphere Web Services SDK, you can create your own client applications that automate many administration, provisioning, or monitoring tasks associated with virtual infrastructure management and operations. The following examples are operational tasks that you can automate using the vSphere Web Services API:

- Create, configure, power cycle, or suspend virtual machines explicitly or by using profiles or templates to facilitate faster provisioning.
- Create, configure, and manage virtual devices, such as virtual CD-DVD drives, virtual network interface cards, virtual switches, and other components.
- Connect, power cycle, and disconnect ESXi host systems.
- Capture the state of a virtual machine to a snapshot and restore the state of a virtual machine from a snapshot, such as in a backup application.
- Gather statistics about host system and virtual machine performance.
- Manage events generated by the server, such as those created by alarms set for specific thresholds.
- Move virtual machines between hosts automatically.
- Manage load balancing and failover through the distributed resource scheduler (VMware DRS) and high availability (VMware HA) subsystems. VMware DRS and VMware HA require vCenter Server.

This list is not comprehensive. Also, some of the operations pertain to the service as a whole, not specific hosts or virtual machines. For example, load balancing can be a service-wide operation rather than a per-host or per-virtual machine operation.

How You Can Download the vSphere Web Services SDK

The vSphere Web Services SDK is part of the vSphere Management SDK. This section tells how to access the Web Services SDK files.

You can download the vSphere Management SDK at https://code.vmware.com. Click the **SDKs** link to reach a landing page that has links to all vSphere SDKs. After you download and expand the vSphere Management SDK package, the Web Services SDK is in the subdirectory `SDK/vsphere-ws`. You also need the `SDK/ssoclient` subdirectory for client authentication.

Sample Authentication with Single Sign-On Configurations

You can choose to authenticate your client with vCenter Server Single Sign-On. This topic explains how to do Single Sign-On authentication with the SDK samples..

At this time, the two possible configurations are:

- vCenter Server with an embedded Platform Services Controller, which includes vCenter Single Sign-On Server, Lookup Service Server, and other features. This configuration combines all the platform services in the same installation with vCenter Server.
- vCenter Server with an external Platform Services Controller, where the platform services are installed on a different host or in a different virtual machine from vCenter Server.

You will have to provide the vCenter Single Sign-On URL explicitly in order to run samples with the second configuration. With the first configuration, there is no need to provide the vCenter Single Sign-On URL (since the vCenter Single Sign-On service is embedded in the management node) and our SDK will continue to work as before.

The SDK samples have always had the option to explicitly specify the vCenter Single Sign-On URL (whether the vCenter Single Sign-On service is running inside or outside the management node). This is useful in cases where the vCenter Single Sign-On service is deployed outside the vSphere management node like the second configuration above.

When you log in to a vCenter Single Sign-On Server, you must be in a domain that has been added as a vCenter Single Sign-On identity source. If that domain is not the default domain, you must include the domain name as part of your user name, such as, administrator@vsphere.local. To learn more about configuring the vCenter Single Sign-On Server, see vSphere Security.

What files are in the vSphere Web Services SDK package

The vSphere Web Services SDK is one of several SDKs bundled inside the vSphere Management SDK. The Web Services SDK contains the following files:

- WSDL files that define the API available on a VMware vSphere server (ESXi or vCenter Server) Web service.
- Precompiled client-side libraries (`vim.jar`, `vim25.jar`) available for test purposes that were generated from the WSDL. The vSphere Web Services API is packaged in the `vim25.jar` file and is available in the `SDK\vsphere-ws\wsdl\vim25` subdirectory.
- Sample code demonstrating common use cases associated with managing virtual infrastructure. The sample code includes compiled and ready-to-run Java class files and source code files.
- Batch files and shell scripts (`build.bat` and `build.sh`) that automate the build process for Java client applications.
- Batch files and shell scripts (`run.bat` and `run.sh`) that facilitate running the Java samples from the Windows command prompt.
- The *`vSphere API Reference`*, which provides language-neutral descriptive information about the VMware vSphere API and the object model, such as object type definitions, properties, and method signatures.

Complete information about setting up the environment, and about generating, compiling, and running applications is included in Set Up to Develop in Java with the vSphere Web Services SDK.

SDK Version Support and Limitations of the ESXi API

vSphere supports various versions, both with vCenter Server and with ESXi. The ESXi service contains some limitations not present in vCenter Server.

You can use the VMware vSphere Web Services SDK with many previous versions of VMware vSphere servers and its predecessor, VMware Infrastructure, including:

- ESXi 8.0, 7.0, 6.7, 6.5, 6.0
- vCenter Server 8.0, 7.0, 6.7, 6.5, 6.0

All versions are supported by the WSDL declarations in the `vimService.wsdl` top-level file in the `vim25` directory.

The VMware vSphere API is a Web service that runs on VMware vSphere ESXi and vCenter Server. The API exposed is the same in both products. However, vCenter Server provides the following capabilities which are not available through an ESXi Web service:

- Collecting historical performance data
- Optimizing resources, including managing distributed resources
- Enabling migration from one host system to another by using VMware vMotion
- Providing distributed resource management, including recovery, across all host systems under its control

If you attempt to invoke an operation on an ESXi system that is supported only on vCenter Server, the server returns a fault saying “not implemented” or “not supported.” For example, the ExtensionManager API is available only on VirtualCenter Server. Attempting to register an extension to an ESXi system returns a fault, “not supported.”

Set Up to Develop in Java with the vSphere Web Services SDK

To develop a Web Services client, you need a development environment. This chapter explains how to set up an environment to develop in Java.

Java Development Requirements for the Web Services SDK

To develop Java client applications using the VMware vSphere Web Services SDK, you need Java SE and a Java Web services development toolset.

For best results, use the Java platform Standard Edition (SE) version 11. Java SE includes both the runtime environment (JRE) and the development kit (JDK).

The Java Web services development toolset must be a SOAP implementation that can be deployed to a Tomcat server. For example, you can use the client-side libraries in JAX-WS. The JAX-WS libraries are included with the vSphere Management SDK.

You can use other client-side tools and libraries, such as IBM WebSphere and several open source implementations, such as Open JDK, with the vSphere Web Services SDK. However, only the JAX-WS client libraries were tested with this guide.

The samples archive, in `samples.jar`, includes all vSphere Web Services SDK samples. The samples include client-side stub classes generated using the JAX-WS libraries.

> **NOTE**
> Use JDK 11 for production environments. vCenter Server supports HTTPS connections with Java 1.8 and later, but not with Java 1.7. If you need to use Java 1.7 for development work, see Endpoint Configuration for HTTP.

Set Up for Java Development for the Web Services SDK

Specific setup instructions depend on whether your development workstation already meets some or all of the requirements, which client-side Web service library you plan to use, and whether you plan to use the provided samples. Specific setup instructions also depend on whether your target server uses the HTTPS protocol or HTTP.

Software Downloads for the Web Services SDK

You can obtain the software you need for Java client development from the following Web sites:

- The JDK is available from http://www.oracle.com. The vSphere API supports connections from JDK 11.
- Obtain the VMware vSphere Web Services SDK from https://developercenter.vmware.com. It is included in the vSphere Management SDK package.

Set Up for Development Using JAX-WS with the Web Services SDK

The Web Services SDK provides precompiled samples that use JAX-WS libraries. The precompiled files include `vim25.jar` and `samples.jar`.

The precompiled sample programs use the `vim25.jar` client side library to access the Web Services API. The following instructions describe how to set up your environment to run sample programs. These instructions assume that the target server uses HTTPS, which is the default server configuration.

1. If the JDK is not installed, create directories for the JDK and for the vSphere Web Services SDK package.

Do not use spaces in the directory names, to avoid issues with some of the included SDK batch and script files.

2. Install Java SE with JDK 11.
3. Unpack the components into subdirectories created in If the JDK is not installed, create directories for the JDK and for the vSphere Web Services SDK package. , using the provided installer if appropriate.

   The JDK uses an installation wizard. The SDK ZIP file unpacks into the directory you specify.
   - Unpack with **Use folder names** selected, to maintain the organizational structure.
   - On UNIX development systems, use the **`unzip`** command with the `-a` modifier, to ensure proper line-endings in the shell scripts. For example:
     ```
     unzip -a VMware-vSphere-SDK-8.0.0-14379537.zip
     ```

4. Import server-certificates and use the Java keytool utility to create a **`vmware.keystore`**. See Import Server Certificates into the Java Keystore for the Web Services SDK for details.

   As an alternative, pass the `--ignorecert` argument at runtime to ignore server-certificate verification for any of the sample Java applications.

5. Create the JAVAHOME environment variable.

   The JAVAHOME environment variable must be set to the root path of the Java Runtime Environment (JRE), such as `C:\Program Files\Java\jdk11.0.16_221`. The root directory of your Java installation contains `bin\javac` and other binary files needed to build the stubs and the samples.

6. If you are unable to use the `run.bat` or `run.sh` script to run Java samples, add the precompiled sample files, `vim25.jar` and `samples.jar`, to your system CLASSPATH environment variable.

   To test your setup, run the Java version of SimpleClient, as described in Running the Web Services SDK SimpleClient Sample Application to Validate Setup.

Batch Files and Shell Scripts for Building and Running Samples in the Web Services SDK

The vSphere Web Services SDK includes several batch files for Windows and shell scripts for Linux that facilitate building and running the sample applications.

Some of the batch files are used by other batch files. For example, `build.bat` calls the `lcp.bat` and `clean.bat` scripts. If you modify the batch files for any reason, be aware of the dependencies among them.

Filename	Description	Usage note
`build.bat` `build.sh`	Checks for environment variable JAVAHOME and sets PATH, using the JAVAHOME variable. Cleans up existing Java files (by calling `clean.bat` or **`clean.sh`**). `build.bat` sets the local classpath (by calling `lcp.bat`). Creates the `vim25.jar`, and `samples.jar` files.	Use this script to generate client stubs and rebuild all sample applications. Use the -w flag to recompile without regenerating stubs.
`lcp.bat`	Sets the local classpath on the workstation. Called by `build.bat` and by `run.bat`.	Optional. Use to set local classpath.
`run.bat` `run.sh`	Batch file that enables running any of the sample applications. Sets the Java trustStore property to the local trust store and invokes the Java runtime with the name of the application passed as a parameter.	Use this script to run any Java sample applications.

Table continued on next page

Continued from previous page

Filename	Description	Usage note
`clean.bat`	Removes any existing artifacts before building the samples, deleting Java class files in the samples packages and `samples.jar` file. Called by build script.	Optional. Deletes all generated source code files.

Import Server Certificates into the Java Keystore for the Web Services SDK

Import server certificates if you plan to use the HTTPS protocol and if you do not plan to use the **`--ignorecert`** command-line argument.

To use HTTP, rather than HTTPS, and avoid the use of certificates entirely, follow the procedure detailed in Endpoint Configuration for HTTP. However, using HTTPS provides better security for production environments.
The JAVAHOME environment variable must be set and added to the PATH environment variable. The certificate for each target server must be located in the `vmware-certs` subdirectory, as described in this procedure. For detailed information about server certificates, see Obtaining Server Certificates.

1. Open the Windows command prompt or Linux shell command.
2. Create the directory for the Java certificate store.

   Create the directory only. The actual keystore file, `vmware.keystore`, is created during the process of importing the certificates.

Operating System	Path
Windows	`C:\VMware-Certs\vmware.keystore`
Linux	`~/vmware-certs/vmware.keystore`

3. Navigate to the directory.

   For example, on Windows use the following directory:

   ```
   cd vmware-certs\vmware
   ```

4. Use the Java **`keytool`** utility to import a certificate.

   The syntax is as follows:
   `keytool -import -trustcacerts -alias server-`*`name`* `-file certificate-`*`filename`* `-keystore keystore-`*`name`*

   For example:

   ```
   C:\VMware-Certs>keytool -import -trustcacerts -alias root -file root.cer -keystore
   keystore.jks
   ```

   A prompt requesting a password for the keystore appears:

   ```
   Enter keystore password:
   ```

5. Create a password for the keystore by entering it at the prompt.

   The keystore utility displays the certificate information at the console. For example:

   ```
   Owner: OID.1.2.840.113549.1.9.2="1183400896,564d7761726520496e632e",

   CN=sdkpubslab-01.vmware.com, EMAILADDRESS=ssl-certificates@vmware.com,

   OU=VMware ESX Server Certificate, O="VMware, Inc.", L=Palo Alto,
   ```

```
ST=California, C=US Issuer:

OID.1.2.840.113549.1.9.2="1183400896,564d7761726520496e632e",

CN=sdkpubslab-01.vmware.com, EMAILADDRESS=ssl-certificates@vmware.com,

OU=VMware ESX Server Certificate, O="VMware, Inc.", L=Palo Alto,

ST=California, C=US Serial number: 0 Valid from: Mon Jul 02 11:28:17 PDT 2007 until:
Mon Aug 31 11:28:17 PDT 2026

Certificate fingerprints:

MD5: . . .61:35:C0:C4

SHA1: 4C:...78:B2
```

At the end of the certificate information, a prompt displays a request for confirmation that the certificate is trusted:

```
Trust this certificate? [no]:
```

6. Type yes and press Enter to respond to the prompt and import the certificate into the **`vmware.keystore`** keystore.

   The console displays this message:

   ```
   Certificate was added to keystore
   ```

7. Repeat Step 4 through Step 6 for each target server.

Generating Stubs and Compiling Classes for the Web Services SDK

The vSphere Web Services SDK includes a set of Java archive files for the sample programs. The sample `.jar` files were created using the JAX-WS Web services libraries.

Precompiled JAX-WS Samples in the Web Services SDK

The JAX-WS samples include the `vim25.jar` and `samples.jar` files that were created using the JAX-WS libraries included with the JDK. These files are located in the `%WS_SDK_HOME%\java\JAXWS\lib` directory.

To use these precompiled libraries, see the instructions in Running the Web Services SDK SimpleClient Sample Application to Validate Setup. If you modify the precompiled code, see the build instructions in Use the Included Build Scripts in the Web Services SDK.

Use the Included Build Scripts in the Web Services SDK

If you modify the sample files, or need to rebuild them for any reason, you can use the `build.bat` or `build.sh` script included with the SDK. These scripts performs all necessary tasks to build the SDK files.

You must install the JDK and set the `JAVAHOME` environment variable.

This task shows how to use the build scripts included with the SDK.

1. Open a command prompt.
2. Navigate to the subdirectory containing the `build.bat` and `build.sh` files.

```
cd %WS_SDK_HOME%\java\JAXWS\
```

3. Run the `build.bat` (or `build.sh`) script by entering its name at the command prompt.

   ```
   build
   ```

   The console displays output, starting with **`Generating stubs from wsdl`**. In a few minutes, the process finishes. The word **`Done`** appears at the command prompt, as shown in the following example. The **`Generating stubs from wsdl`** message appears twice, because this build file generates client stubs using both sets of WSDL declarations, found in the `\vim` and `\vim25` subdirectories.

   Successful Stub Generation and Compilation Using the build.bat Script

   ```
   Generating stubs from wsdl

   Compiling stubs.

   ...

   Done.

   C:\devprojects\visdk21\SDK\vsphere-ws\java\JAXWS>
   ```

When the process finishes, the appropriate sample `.jar` files show the current date and time.

To compile without re-generating the stubs from the WSDL, use the
`-w`
flag with the build script, as follows:

```
build -w
```

You can run any of the sample applications by following the instructions in Running the Web Services SDK SimpleClient Sample Application to Validate Setup

Running the Web Services SDK SimpleClient Sample Application to Validate Setup

You can test your setup and connectivity by running one of the sample applications, such as SimpleClient. SimpleClient is a Java class that connects to the server and obtains a listing of the top-level inventory entities, their properties, and references. You can run any of the samples using the `run.bat` (or `run.sh`) script.

If you are using stubs generated by JAX-WS, these scripts require the JAVAHOME environment variable to be set.

Run a Sample Application Using the Provided Scripts in the Web Services SDK

You can use the `run.bat` or `run.sh` script to run any of the Java samples. The **`SimpleClient`** sample is a good choice to verify that your installation is correct. The path to the source file for SimpleClient is:

```
%WS_SDK_HOME%\java\JAXWS\samples\com\vmware\general\SimpleClient.java
```

When you run the script, specify the Java class for the sample application along with the
`--url`
,
`--username`
, and
`--password`
switches on the command line. Include the complete package name in the Java class specification. The following statement shows the general format for using the `run.bat` script to run the **`SimpleClient`** sample application from the Java samples subdirectory for JAX-WS:

```
run.bat com.vmware.general.SimpleClient --url https://yourFQDNservername/sdk
      --username username --password password [--ignorecert ignorecert]
```

The following example shows sample output from the SimpleClient sample program.

Sample Output of a Successful Run of SimpleClient, Using Precompiled Java Sample

```
Object Type : Folder

Reference Value : ha-folder-vm

   Property Name : name

   Property Value : vm

Object Type : HostSystem

Reference Value : ha-host

   Property Name : name

   Property Value : sdkpubslab-02.eng.vmware.com

Object Type : ResourcePool

Reference Value : ha-root-pool

   Property Name : name

   Property Value : Resources

Object Type : Folder

Reference Value : ha-folder-host

   Property Name : name

   Property Value : host

Object Type : ComputeResource

Reference Value : ha-compute-res

   Property Name : name

   Property Value : sdkpubslab-02.eng.vmware.com

Object Type : VirtualMachine

Reference Value : 16

   Property Name : name

   Property Value : Windows_2K3_VM

...

Object Type : Datacenter

Reference Value : ha-datacenter

   Property Name : name

   Property Value : ha-datacenter

Object Type : Folder
```

```
Reference Value : ha-folder-root

   Property Name : name

   Property Value : ha-folder-root
```

To run the precompiled SimpleClient from the command prompt, do the following procedure.

1. Open a Windows command prompt or shell prompt on Linux.
2. Navigate to the Java samples subdirectory.

   ```
   cd %WS_SDK_HOME%\java\JAXWS\samples
   ```

3. Invoke the Java runtime, providing the full package name of the SimpleClient, server URN, credentials, and Java keyStore location, or the
   `--ignorecert`
   argument. The complete syntax is as follows:

   ```
   java -Djavax.net.ssl.trustStore=keystore-path-or-%KEYSTORE%-environment-variable

            package-hierarchy-classname --url server-url --username username

            --password password [--ignorecert ignorecert]
   ```

   For example:

   ```
   java -Djavax.net.ssl.trustStore=%VMKEYSTORE% com.vmware.general.SimpleClient

            --url https://example.com/sdk --username pubs --password ***

            --ignorecert ignorecert
   ```

   NOTE
   If error messages occur due to system heap or other memory problems, you can give the Java VM more memory, as follows:

   ```
   java -Djavax.net.ssl.trustStore=%VMKEYSTORE% -Xms512M -Xmx1024M

           com.vmware.general.SimpleClient https://sdkpubslab-02.eng.vmware.com/
   sdk

           --username username

           --password password --ignorecert ignorecert
   ```

Endpoint Configuration for HTTP

For development environments, you can simplify the connection process by configuring an HTTP connection rather than an HTTPS connection to the Web service. If your client also connects to a vSphere Automation API endpoint, you can use a similar procedure to configure the vSphere Automation API connection.

The procedure to modify the service configuration differs, depending on the service type. Choose one of the following options that applies to your situation:

- HTTP Configuration for Web Services API Endpoint
- HTTP Configuration for vSphere Automation API Endpoint

Modifying Service Endpoint Configurations to Support HTTP

ESXi, vCenter Server, and vSphere Automation API endpoints run by default on port 443, as secure Web services that can be accessed using SSL or TLS over HTTP (HTTPS). However, for a development environment, you might want to simplify the connection process from a client application by configuring the target services to support HTTP.

> **NOTE**
> The HTTP protocol without SSL or TLS is insecure. Your user credentials could be intercepted by wire capture software.

The API service endpoints are handled by a reverse-proxy service, which has a configuration file that can be modified to specify support for HTTP as an accepted protocol. If you configure the service for HTTP, you do not need to import the server certificates on the client development workstation. Modifying the service configuration to support HTTP access is recommended for test or development environments only, not for production deployments. The default protocol, HTTPS, provides better security for production deployments.

HTTP Configuration for Web Services API Endpoint

You can modify the Web Services API endpoint configuration to accept HTTP connections, in addition to or instead of HTTPS connections.

HTTP Configuration for Web Services API Endpoint on ESXi or vCenter Server for Windows

This procedure applies to version 5.5 or later ESXi or vCenter Server for Windows.

You can modify ESXi configuration from a shell window over an SSH connection, using the following procedure. If you do not have SSH enabled, use the appropriate vSphere CLI command to obtain the configuration file from the server, modify the file to support HTTP, and move the file back to the ESXi system. For more information about the vSphere CLI command syntax, see the *vSphere CLI Installation and Reference Guide*.

1. Log in to a shell window or File Explorer with **root** or **administrator** privileges.
2. Change directories to the location of the endpoint configuration file.

   The location differs, depending on the platform.
   - For ESXi: `/etc/vmware/rhttpproxy`
   - For vCenter Server for Windows: `C:\Program Data\VMware\vCenterServer\cfg\vmware-rhttpproxy\endpoints.conf.d`

3. Copy the `endpoints.conf` file to a temporary directory for editing.
   - On ESXi: `# cp endpoints.conf /tmp/endpoints.conf`
   - On vCenter Server for Windows, use File Explorer to copy the file.

4. Change the permissions on the temporary `endpoints.conf` file to allow editing.
   - On ESXi: `# chmod +w /tmp/endpoints.conf`
   - On vCenter Server for Windows, right click the file in File Explorer and select **Properties** to change file permissions.

5. Use a text editor to open the temporary file.
   - On ESXi: `# vi /tmp/endpoints.conf`
   - On vCenter Server for Windows, choose any text editor, such as Notepad, from the **Start** menu.

6. Navigate to the line that specifies the endpoints for SDK connections, which begins with `/sdk`.

   The line looks similar to this:

```
/sdk    local    8085              redirect     allow
```

7. To enable HTTP connections, change the word **`redirect`** to **`allow`**.

   When configured to allow both HTTP and HTTPS connections, the `/sdk` line looks similar to this:

   ```
   /sdk    local    8085              allow     allow
   ```

8. (Optional) If you prefer to completely disable HTTPS, change the last word to **`reject`** instead of **`allow`**.

   When configured to allow only HTTP connections, the `/sdk` line looks similar to this:

   ```
   /sdk    local    8085              allow     reject
   ```

9. (Optional) Change the setting for the Managed Object Browser as well.

   When configured to allow both HTTP and HTTPS connections, the `/mob` line looks similar to this:

   ```
   /mob    local    8085              allow     allow
   ```

10. Save your settings and close the file.
11. Change the permissions on the temporary file to disable editing.
    - On ESXi: `# chmod -w /tmp/endpoints.conf`
    - On vCenter Server for Windows, right click the file in File Explorer and select **Properties** to change file permissions.
12. Copy the original `endpoints.conf` file to a backup file.
    - On ESXi: `# cp endpoints.conf endpoints.conf.old`
    - On vCenter Server for Windows, use File Explorer to copy the file.
13. Copy the temporary file `endpoints.conf` file back, replacing the original `endpoints.conf` file.
    - On ESXi: `# cp /tmp/endpoints.conf endpoints.conf`
    - On vCenter Server for Windows, use File Explorer to copy the file.
14. Signal the reverse proxy service to update its configuration by entering the following command:
    - On ESXi:
      `/etc/init.d/rhttpproxy restart`
    - On vCenter Server for Windows:
      From the Windows menu, choose **Control Panel › Administrative Tools › Services**, right click the rhttpproxy service, and choose **Restart**.

An `endpoints.conf` File Modified To Support HTTP connections to the SDK and the MOB

```
/                   local          8309                          redirect       allow
/sdk                local          8307                          allow          allow
/client/clients.xml local          8309                          allow          allow
/ui                 local          8308                          redirect       allow
/vpxa               local          8089                          reject         allow
/mob                namedpipe      /var/run/vmware/proxy-mob     allow          allow
/wsman              local          8889                          redirect       allow
```

```
/sdkTunnel            namedpipetunnel /var/run/vmware/proxy-sdk-tunnel allow
 reject
/ha-nfc               local           12001                            allow          allow
/nfc                  local           12000                            allow          allow
/folder               local           8309                             redirect       allow
/host                 local           8309                             redirect       allow
/tmp                  local           8309                             redirect       allow
/screen               local           8309                             redirect       allow
/guestFile            local           8309                             redirect       allow
/cgi-bin              local           8309                             redirect       allow
```

HTTP Configuration for Web Services API Endpoint on vCenter Server Appliance

You can modify the Web proxy service to support HTTP on vCenter Server Appliance.

You can modify from a shell window over an SSH connection, using the following procedure. If you do not have SSH enabled, see *vCenter Server Appliance Configuration* or *VMware vCenter Server Appliance Progrmmaing Guide*. This procedure applies to version 6.0 or later vCenter Server Appliance.

1. Log in to a shell window with **root** privileges.
2. Change directories to the location of the endpoint configuration file.
   - For vCenter Server Appliance: cd /etc/vmware-rhttpproxy/endpoints.conf.d
3. Copy the `endpoints.conf` file to a temporary directory for editing.
   - On vCenter Server Appliance: `# cp vpxd-rhttpproxy-endpoint.conf /tmp/vpxd-rhttpproxy-endpoint.conf`
4. Change the permissions on the temporary vpxd-rhttpproxy-endpoint.conf file to allow editing.
   - On vCenter Server Appliance: `# chmod +w /tmp/vpxd-rhttpproxy-endpoint.conf`
5. Use a text editor to open the temporary file.
   - On vCenter Server Appliance: `# vi /tmp/vpxd-rhttpproxy-endpoint.conf`
6. Navigate to the line that specifies the endpoints for SDK connections, which begins with `/sdk`.

   The line looks similar to this:

   ```
   /sdk    local    8085              redirect     allow
   ```
7. To enable HTTP connections, change the word **redirect** to **allow**.

   When configured to allow both HTTP and HTTPS connections, the `/sdk` line looks similar to this:

   ```
   /sdk    local    8085              allow     allow
   ```
8. (Optional) If you prefer to completely disable HTTPS, change the last word to **reject** instead of **allow**.

   When configured to allow only HTTP connections, the `/sdk` line looks similar to this:

   ```
   /sdk    local    8085              allow     reject
   ```

9. (Optional) Change the setting for the Managed Object Browser as well.

   When configured to allow both HTTP and HTTPS connections, the `/mob` line looks similar to this:

   ```
   /mob    local    8085              allow     allow
   ```

10. Save your settings and close the file.
11. Change the permissions on the temporary file to disable editing.
    - On vCenter Server Appliance: `# chmod -w /tmp/vpxd-rhttpproxy-endpoint.conf`
12. Copy the original `endpoints.conf` file to a backup file.
    - On vCenter Server Appliance: `# cp vpxd-rhttpproxy-endpoint.conf vpxd-rhttpproxy-endpoint.conf.old`
13. Copy the temporary file `endpoints.conf` file back, replacing the original `endpoints.conf` file.
    - On vCenter Server Appliance: `# cp /tmp/vpxd-rhttpproxy-endpoint.conf vpxd-rhttpproxy-endpoint.conf`
14. Signal the reverse proxy service to update its configuration by entering the following command:
    - On vCenter Server Appliance:
      /etc/init.d/vmware-rhttpproxy restart

    For an example of the contents of a vpxd-rhttpproxy-endpoint.conf file modified to support HTTP connections, see HTTP Configuration for Web Services API Endpoint.

HTTP Configuration for vSphere Automation API Endpoint

You can also modify the vAPI endpoint configuration for vCenter Server to accept HTTP connections in addition to or instead of HTTPS connections. This is useful for clients that interact both with the vSphere Web Services API endpoint and the vSphere Automation API endpoint in a development environment. Use the following procedure.

1. Log in to a shell window or File Explorer with**root** or**administrator** privileges.
2. Change directories to the location of the endpoint configuration file.

   The location differs, depending on the platform.
   - For vCenter Server Appliance: `/etc/vmware-rhttpproxy/endpoints.conf.d`
   - For vCenter Server for Windows: `C:\Program Data\VMware\vCenterServer\cfg\vmware-rhttpproxy\endpoints.conf.d`
3. Copy the vapi-`endpoint.conf` file to a temporary directory for editing.
   - On vCenter Server Appliance: `# cp vapi-endpoint.conf /tmp/vapi-endpoint.conf`
   - On vCenter Server for Windows, use File Explorer to copy the file.
4. Change the permissions on the temporary file to allow editing.
   - On vCenter Server Appliance: `# chmod +w /tmp/vapi-endpoint.conf`
   - On vCenter Server for Windows, right click the file in File Explorer and select **Properties** to change file permissions.
5. Use a text editor to open the temporary file.
   - On vCenter Server Appliance: `# vi /tmp/vapi-endpoint.conf`
   - On vCenter Server for Windows, choose any text editor, such as Notepad, from the **Start** menu.
6. Navigate to the line that specifies the endpoint for REST connections, which begins with `/rest`.

The line looks similar to this:

```
/rest    local    12346              redirect     allow
```

7. To enable HTTP connections for REST clients, change the word **`redirect`** to **`allow`**.

   When configured to allow both HTTP and HTTPS connections, the `/rest` line looks similar to this:

   ```
   /rest    local    12346              allow     allow
   ```

8. (Optional) If you prefer to completely disable HTTPS, change the last word to **`reject`** instead of **`allow`**.

   When configured to allow only HTTP connections, the `/rest` line looks similar to this:

   ```
   /rest    local    12346              allow     reject
   ```

9. Navigate to the line that specifies the endpoint for clients that use language bindings rather than REST, which begins with `/api`.

   The line looks similar to this:

   ```
   /api    local    12346              redirect     allow
   ```

10. To enable HTTP connections for clients using language bindings rather than REST, change the word **`redirect`** to **`allow`**.

    When configured to allow both HTTP and HTTPS connections, the `/api` line looks similar to this:

    ```
    /api    local    12346              allow     allow
    ```

11. Save your settings and close the file.
12. Change the permissions on the temporary file to disable editing.
    - On vCenter Server Appliance: **#** `chmod -w /tmp/vapi-endpoint.conf`
    - On vCenter Server for Windows, right click the file in File Explorer and select **Properties** to change file permissions.
13. Copy the original vapi-`endpoint.conf` file to a backup file.
    - On vCenter Server Appliance: **#** `cp vapi-endpoint.conf vapi-endpoint.conf.old`
    - On vCenter Server for Windows, use File Explorer to copy the file.
14. Copy the temporary file back, replacing the original vapi-`endpoint.conf` file.
    - On vCenter Server Appliance: **#** `cp /tmp/vapi-endpoint.conf vapi-endpoint.conf`
    - On vCenter Server for Windows, use File Explorer to copy the file.
15. Signal the reverse proxy service to update its configuration by entering the following command:
    - On vCenter Server Appliance:
      /etc/init.d/vmware-rhttpproxy restart
    - On vCenter Server for Windows:
      From the Windows menu, choose **Control Panel › Administrative Tools › Services**, right click the rhttpproxy service, and choose **Restart**.

A vapi-endpoint.conf File Modified To Support HTTP connections for API and REST

```
/vapiendpoint/health            local          12346       allow          allow
```

```
/vapiendpoint/resourcebundle    local          12346       allow          allow
/rest                           local          12346       allow          allow
/site/api                       local          12346       redirect       allow
/site/rest                      local          12346       redirect       allow
/api/                           local          12346       allow          allow
```

vSphere Web Services SDK Programming Guide

The *vSphere Web Services SDK Programming Guide* provides information about developing applications using the VMware® vSphere Web Services SDK.

VMware provides different APIs and SDKs for various applications and goals. The vSphere Web Services SDK targets developers who create client applications for managing VMware® vSphere components available on ESXi hosts and vCenter Servers.

For more VMware API and SDK documentation, go to https://developer.broadcom.com.

Revision History

This book is revised with each release of the product or when necessary. The following table summarizes significant changes in each version of this book.

Table 3: Revision History

Revision Date	Description
18 JUN 2024	vSphere 8.0 Update 3 • Updates to Client Applications chapters, Web Services and JSON. • Rebranded website links from VMware to Broadcom. • Corrections to PropertyCollector chapter, especially concerning WaitForUpdatesEx (OCT 2023).
21 SEP 2023	vSphere 8.0 Update 2 • Removed `/etc/syslog.conf` from `/host` HTTP access. This configuration has been moved to the ConfigStore. • Changed some section titles for search engine optimization. • Corrected reference to PropertyCollector as a singleton, which is not strictly correct. • Added clarifications regarding PropertyCollector updates.
18 APR 2023	vSphere 8.0 Update 1 • Added Client Applications Using the JSON Protocol.
05 DEC 2022	Minor cleanup.
06 OCT 2022	vSphere 8.0 • Removed C# and .NET material after deprecation. • Removed more system configuration parameters that have been moved to ConfigStore. These settings can no longer be edited directly with a web browser. • Updates for accuracy including Security chapter. Alt text for graphics.
05 OCT 2021	vSphere 7.0 Update 3 • Added raw cloud-init configuration to Guest Operations chapter. • Added PTP time service capability to Hosts chapter.
19 MAY 2021	• Added information about configuring advanced host options. • Added information about configuring direct access to physical devices on the host. • A number of system configuration parameters moved to ConfigStore. These settings can no longer be edited directly with a web browser.
12 APR 2021	VMware has rebranded the My VMware portal as VMware Customer Connect. We have updated this document accordingly.

Table continued on next page

Continued from previous page

Revision Date	Description
9 MAR 2021	vSphere 7.0 Update 2 • Added a simpler use case in the Property Collector chapter. Also made minor corrections and clarifications. • Rerganized part of the Tasks chapter to break up the very long Listview example program into sections. Also made minor corrections and clarifications.
06 OCT 2020	vSphere 7.0 Update 1 • In Inventory Traversal and Property Selection, add best practice to destroy PropertyCollector views after use. • In Task Monitoring Example Using the Listview Object, cross-reference information about chunking of results from PropertyCollector. • At VMware, we value inclusion. To foster this principle within our customer, partner, and internal community, we are replacing some of the terminology in our content. We have updated this guide to remove instances of non-inclusive language.
02APR2020	vSphere 7.0 Added scalable shares feature. Reworked and expanded fixed shares and other resource anagement material. Added Virtual Machine Guest Operations chapter, including guest customization for instant clone virtual machines.
20AUG2019	vSphere 6.7 Update 3 Bug fixes. Added section on crypto key query for VM encryption.
16OCT2018	vSphere 6.7 Update 1 Added section on Hyperconverged Infrastructure (HCI) clusters.
17APR2018	vSphere 6.7 - GA Added Instant Clone material to VM Management chapter. Minor updates elsewhere.
12APR2017	Removed appendix with list of permissions. Was not maintained.
15NOV2016	vSphere 6.5 - GA Added chapter about vSphere virtual machine encryption. Removed description of deprecated CIM Storage Management API. Updated information about session cookie management.
04SEP2015	Updated information about migrating VMs with vMotion across data centers.
12MAR2015	vSphere 6.0 - Rewrote “Exporting a Virtual Application” section in Virtual Applications chapter.
19SEP2013	vSphere 5.5 – Added a C# example of using LoginByToken; clarified limitation for HA clusters.
10SEP2012	vSphere 5.1 – Added information about using the SessionManager.LoginByToken method; added information about distributed virtual switches.
24AUG2011	vSphere 5.0 - Revised performance manager chapter. Added information about: unset properties, using vCenter to access host data, and using the QueryConfigOption to add devices; emphasized

Table continued on next page

Continued from previous page

Revision Date	Description
	ListView instead of TaskManager; clarified limits and limitations of Linked Virtual Machines; updated samples in chapters 3,5,14, and 16; replaced information about Axis bindings with JAX-WS; and updated paths to samples supplied with SDK.
13JUL2010	Restructured manual and added chapters about host, storage, and networking. Revised property collector chapter and added appendix about HTTP access.
07MAY2009	vSphere Web Services SDK 4.0 Programming Guide.

Intended Audience

This book is intended for anyone who needs to develop applications using the vSphere Web Services SDK. Developers typically create client applications using Java or C# (in the Microsoft .NET environment) targeting VMware vSphere. An understanding of Web Services technology and some programming background in one of the stub languages (C# or Java) is required.

The vSphere Web Services SDK

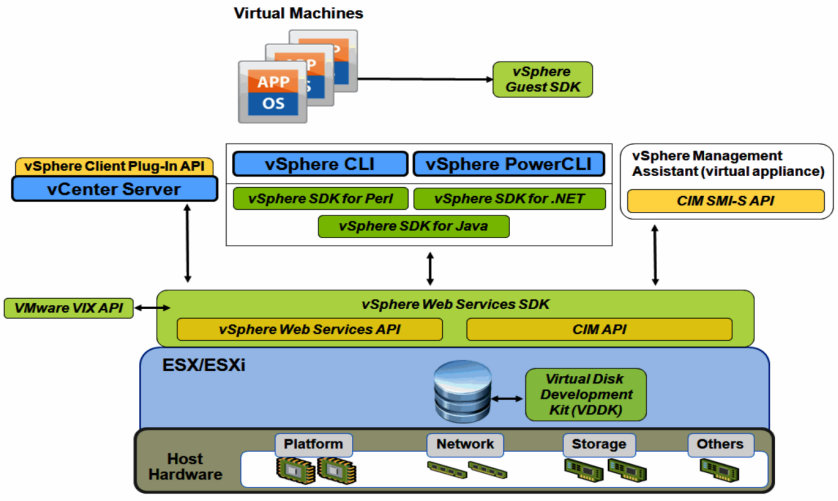
VMware vSphere supports robust, fault-tolerant virtualized applications, networking, and storage. vSphere offers many optional components such as High Availability and vMotion. The vSphere Web Services SDK gives developers programmatic access to most vSphere components.

vSphere Web Services SDK

The vSphere Web Services SDK is the most comprehensive of the available management APIs. The SDK works with both ESXi and vCenter Server systems.

The Web Services SDK is language neutral. It includes stubs and examples for Java and a comprehensive documentation set including an *API Reference* generated from the source.

Figure 3: vSphere APIs and CLIs



SDK Developer Setup for the Web Services SDK

Before you can start developing applications with the vSphere Web Services SDK, you must download the software and set up your system. The *vSphere Web Services SDK Developers Setup Guide* has complete instructions for Java client development and discusses a simplified security configuration for development environments.

SDK Samples for the Web Services SDK

The SDK includes a set of Java samples that illustrate many of the SDK features.

The samples include a set of utility applications that are used by the sample code. Example code fragments in this guide are in part based on the Java sample applications, but show code that does not require the utility applications to run.

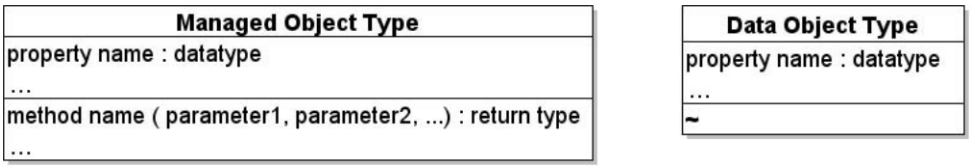
See Sample Program Overview for lists of samples and a brief explanation of what each sample does.

UML Diagrams Used in the Web Services SDK Programming Guide

This guide uses UML (unified modeling language) diagrams to illustrate the API objects and their relationships.

The guide includes class diagrams and instance diagrams. Figure: Legend for UML Class Diagrams shows the UML notation used for managed objects and data objects. The diagrams use a tilde (~) if an object has no properties or methods. Ellipses (...) means some properties or methods are omitted.

Figure 4: Legend for UML Class Diagrams



vSphere Web Services Programming Model

The vSphere Web Services API is implemented as a language-neutral Web service. The API is based on a remote procedure call mechanism that client applications use to access services and components on ESXi and vCenter Server systems.

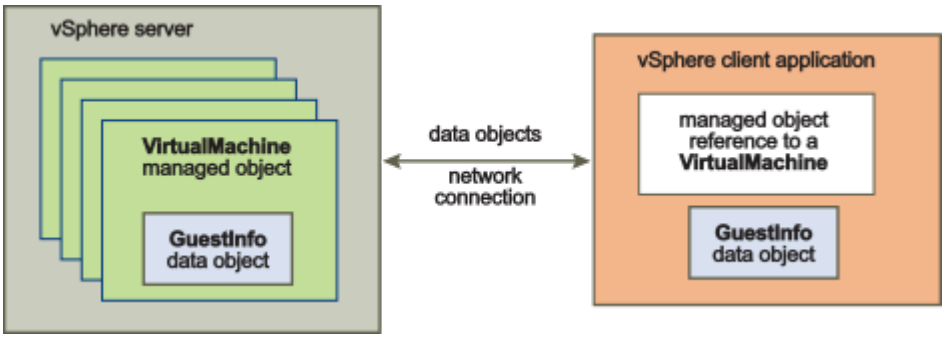
vSphere Client-Server Architecture

VMware vSphere client applications participate in a distributed architecture that uses an asynchronous communications model. This architecture is based on server-side managed objects, client-side managed object references, and data objects.

- Managed objects exist on a vSphere server (ESXi or vCenter Server systems). They represent vSphere services and components. Services include **`PropertyCollector`**, **`SearchIndex`**, **`PerformanceManager`**, and **`ViewManager`**. Components include inventory objects such as **`VirtualMachine`**, **`Datastore`**, and **`Folder`**.
- Managed object references are client application references to server-side managed objects. Your client application uses **`ManagedObjectReference`** objects when it invokes operations on a server. A **`ManagedObjectReference`** is guaranteed to be unique and persistent during an object's lifetime. The reference persists after an object has moved within the inventory, across sessions, and across server restarts. If you remove an object, for example, a virtual machine, from the inventory, and then put it back, the reference changes.
- Data objects contain information about managed objects. Your client application sends data objects to and receives data objects from a vSphere server. Examples are the different specification and capability objects such as **`VirtualMachineConfigSpec`** and **`HostCapability`**.

Figure: vSphere Server and Client shows a vSphere server and client application. The client has a managed object reference to a virtual machine on the server, and a copy of the **`GuestInfo`** data object for the virtual machine. A client must maintain its copy of a data object because, depending on the type of client request, a vSphere server might send property data for a data object as a set of name-value pairs associated with a managed object reference. See the description of the **`ObjectContent`** data object in the *`vSphere API Reference`*.

Figure 5: vSphere Server and Client



The VMware vSphere application model uses an asynchronous client-server communication model in most cases. Methods are nonblocking and return a reference to a **`Task`** managed object. For more information about this managed object, see Tasks and Scheduled Tasks.

Web Services API as a Web Service

The vSphere Web Services API is a language-neutral Web service that runs on ESXi hosts and vCenter Server systems.

The API complies with the Web Services Interoperability Organization (WS-I) Basic Profile 1.0. The WS-I Basic Profile 1.0 includes support for the following standards:

- XML Schema 1.0
- SOAP 1.1
- WSDL 1.1

For information about the WS-I Basic Profile 1.0, go to the Web Services Interoperability Organization (WS-I) Web site at http://ws-i.org.

Web services employ **operations**, which are the same as methods in other programming languages. The vSphere API Web Service provides access to most operations necessary for monitoring and managing vSphere components, such as compute resources, virtual machines, networks, storage, and so on.

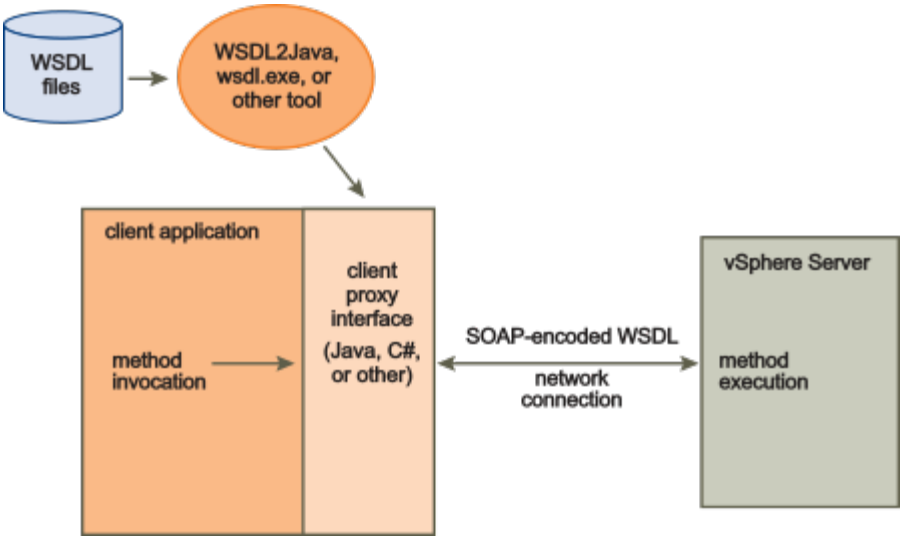
WSDL Files and the Client-Side Proxy Interface

The vSphere Web Services SDK provides a set of WSDL (Web Services Description Language) files that define the vSphere Web Services API. Web-services development tools such as JAX-WS **`wsimport`** use these WSDL files to generate client-side proxy code (stubs).

The client proxy provides a language-specific interface proxy. The proxy facilitates remote method invocation, organization of object data, and other aspects of distributed, object-oriented, applications programming. Your client application calls proxy interface methods. The client proxy uses SOAP (Simple Object Access Protocol) or JSON (JavaScript Object Notation) to exchange WSDL messages with a vSphere server.

Figure: Client-Server Communication Through a Client Proxy Interface is a representation of a client application that uses the client proxy interface to call a method. The client proxy interface is based on the WSDL definitions.

Figure 6: Client-Server Communication Through a Client Proxy Interface



To use the VMware client proxy interface, you must import the vSphere Web Services API client libraries in to your client application using the following Java statement.

```
import com.vmware.vim25.*;
```

> **IMPORTANT**
> The vSphere Web Services SDK includes Java client-side proxy code (stubs) generated using the JAX-WS toolkit. If the versions of Java and JAX-WS on your development platform are the same as those used to generate the stubs shipped in the SDK, you do not have to generate stubs from the WSDL yourself. See the *Developer's Setup Guide* for information about how to configure a vSphere Web Services development environment.

Network Access to the vSphere Web Service

Your client application can use the vSphere Web Services API to communicate with vSphere servers over HTTPS (HTTP over an encrypted Secure Sockets Layer connection) at port 443. HTTPS is the default protocol.

You can configure the server to support HTTP. Use HTTP access only for test or development environments, not for production. For information about how to configure the server to support HTTP access, see the *vSphere Web Services SDK Developer's Setup Guide*.

Language-Specific Classes and Methods

The SOAP tools generate language-specific classes and methods that match the WSDL definitions. The tools also produce objects and methods that are not in the WSDL files.

- Generated objects. The additional objects provide access to the vSphere Web Service to establish the client-server connection (**`VimService`**, **`AppUtil`**) and declare the methods defined for the vSphere API (**`VimPortType`**, **`VimService`**).
- Generated methods. The additional methods are accessor (getter) and mutator (setter) methods for properties. For Java, the method names are constructed by adding **`get`** and **`set`** prefixes to a property name, and changing the first character of the property name to upper case.

The following table identifies client proxy definitions for the vSphere Web Services SDK WSDL.

Element Access	Java
Access to vSphere Web service (HTTPS/HTTP)	**`VimService`** class
Access to vSphere API methods	**`VimPortType`** class
Access to vSphere API properties	**`get`** *`PropertyName`* and **`set`***`PropertyName`* methods defined for data objects
vSphere API data objects	Data objects in the vSphere API (see the *`vSphere API Reference`*) defined as objects in the proxy interface

The following code fragments show getter and setter method definitions for the **`AfterStartupTaskScheduler.minute`** property.

Java

```
public int getMinute() {
     return minute; }
public void setMinute(int minute) {
     this.minute = minute; }
```

You can extrapolate the getter and setter methods that are available in the client proxy interface from the *`vSphere API Reference`*. For example, the **`ScsiLun`** data object has a **`displayName`** property. For the Java API, you can use a **`setDisplayName`** method to assign a string value to the property, and obtain the string value by using the **`getDisplayName`** method. The vSphere Web Services SDK includes Java sample code that illustrates how to use the proxy interfaces. See Client Applications for the Web Services API.

Mapping XML Data Types to Java or JSON Data Types

In this guide, the UML class and object diagrams use the primitive data type names such as string and integer, without the XML Schema definition namespace prefix (**`xsd:`**). The *`vSphere API Reference`* contains the complete data type name, such as **`xsd:string`**. The data types map to the primitive data types of the programming language used for the client application.

The following table lists some of the more common XML primitive data type mappings.

XML Schema	Java	Open API
`xsd:base64binary`	**`byte[]`**	**`string`** **`(format: byte)`**
`xsd:boolean`	**`boolean`**	**`boolean`**
`xsd:byte`	**`byte`**	**`integer`** **`(min: -128, max: 127)`**
`xsd:dateTime`	**`java.util.Calendar`**	**`string`** **`(format: date-time)`**
`xsd:decimal`	**`java.math.BigDecimal`**	
`xsd:double`	**`double`**	**`number`** **`(format: double)`**

Table continued on next page

Continued from previous page

XML Schema	Java	Open API

`xsd:float`	`float`	`number`
		
		`(format: float)`

`xsd:int`	`int`	`integer`
		
		`(format: int32)`

`xsd:string`	`java.lang.String`	`string`

Access to Managed Objects

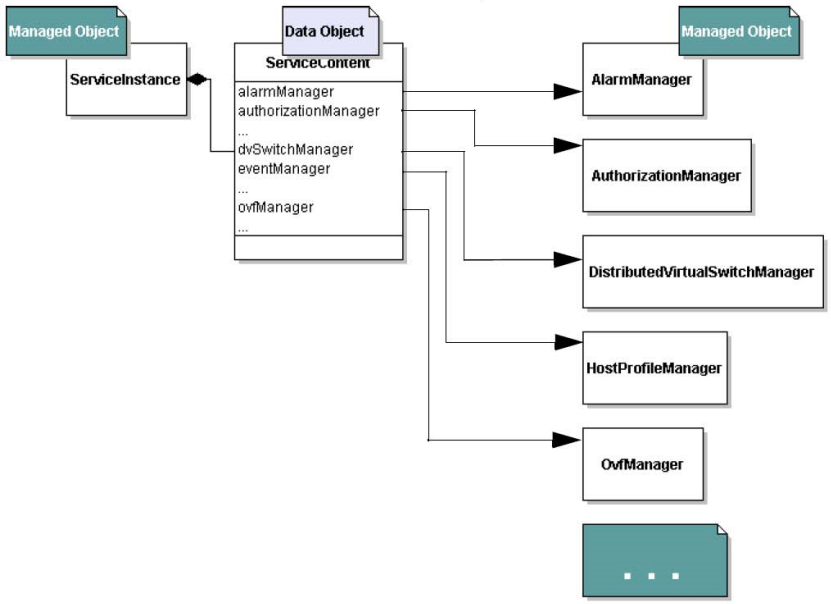
Your client application obtains access to managed objects through the **`ServiceInstance`** managed object and its associated **`ServiceContent`** data object. The **`ServiceContent`** data object contains managed object references to services and manager entities, and to the root folder of the inventory.

The **`ServiceInstance`** managed object is the root object of the inventory on both ESXi and vCenter Server systems. The server creates the **`ServiceInstance`**, and creates the manager entities that provide services in the virtual environment. Examples of manager entities are **`LicenseManager`**, **`PerformanceManager`**, and **`ViewManager`**.

The **`ServiceInstance`** is the primary point of access to the server inventory. Your client application starts by connecting to a server and creating a reference to the **`ServiceInstance`**. After you have connected to the server, you can call the **`ServiceInstance.RetrieveServiceContent`** method to a **`ServiceContent`** data object. **`ServiceContent`** provides access to the vSphere managed object services. See Overview of a Java Sample Application for the Web Services SDK for an example of connecting to a server and using the **`ServiceInstance`** reference to retrieve the **`ServiceContent`** object.

Figure: ManagedObjectReference Data Object shows the object model for the **`ServiceInstance`** and **`ServiceContent`** objects. The figure shows some of the **`ServiceContent`** managed object references and the target objects of the references. Each managed object reference identifies a specific managed object on the server with its type and a value. (The **`value`** property is an opaque string.)

Figure 7: ManagedObjectReference Data Object



Access to vSphere Server Data

To obtain information about the virtual infrastructure, you retrieve managed object properties. Managed object properties can be simple data types, such as integer or string data, or they can be complex types such as data objects that contain sets of properties.

Obtaining Information from a Server

With a reference to a managed object, you can obtain information about the state of the server-side inventory objects and populate client-side data objects based on the values. You can use one of the following approaches:

- Use an accessor (getter) method. The client proxy interface provides accessor methods for each data object property. You can use these accessor methods to obtain the values of the object. See Language-Specific Classes and Methods for information about client proxy interface accessor methods.
- Use a **`PropertyCollector`** to navigate to a selected point on the server and obtain values from specific properties. See Property Collector for more information about **`PropertyCollector`**.
- Use the **`SearchIndex`** managed object to obtain a managed object reference to the managed entity of interest. The **`SearchIndex`** can return managed object references to specific managed entities—**`ComputeResource`**, **`Datacenter`**, **`Folder`**, **`HostSystem`**, **`ResourcePool`**, **`VirtualMachine`**—given an inventory path, IP address, or DNS name.

  IMPORTANT
  You can use API methods to operate on managed objects in the vSphere inventory. A method that updates properties in one managed object may also update properties in other managed objects. The Server performs asynchronous updates to the inventory. There is no guarantee that the inventory will be completely updated when the method returns to the caller. Use the **`PropertyCollector`** method **`WaitForUpdatesEx`** to obtain property changes.

Working with Data Structures

Properties contain information about the server-side objects at a given point in time. The value of a property can be of one of the following types:

- Simple data types, such as a string, boolean, or integer (or other numeric) data type. For example, the **`ManagedEntity`** managed object has a **`name`** property that takes a string value.
- Arrays of simple data types or data objects. For example, a **`HostSystem`** managed object contains an array of managed object references (a type of data object) to virtual machines hosted by that physical machine. As another example, the **`SessionManager`** managed object has a **`sessionList`** property that is an array of **`UserSession`** data objects.
- Enumerated types (enumeration, enum) of predefined values. The values can be a collection of simple data types or data objects. For example, a virtual machine's power state can be one of three possible string values—**`poweredOn`**, **`poweredOff`**, or **`suspended`**.
  The type of a property is often a string, but the property actually expects one of the values an enumeration encapsulates. For example, when you set **`VirtualMachineConfigSpec.guestid`** you can specify one of the elements of the **`VirtualMachineGuestOSIdentifier`** as a string.
- Complex (or composite) data types. For example, the **`HostProfileConfigInfo`** object contains data objects, an array of data objects, and an array of strings.

Accessing Property Values

To use the composite data structures and arrays that contain Server data:

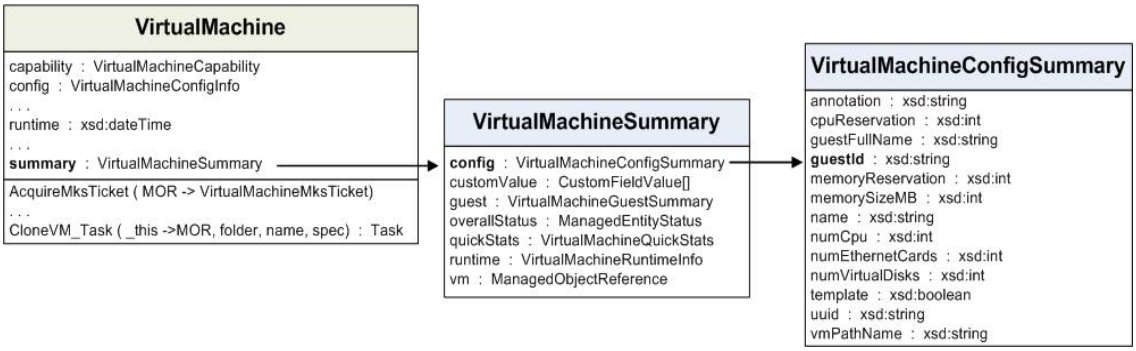
- Use dot notation to access nested properties in composite data structures.
- Cast unconstrained property values (**`xsd:anyType`**) to array types.
- Use keys or index values as appropriate to access array values.

Nested Properties and Property Paths in Composite Data Structures

vSphere Data objects can include properties that are defined as composite data types, such as data objects. The embedded data objects can also contain properties that are data objects. Properties can nest to several levels.

For example, the following figure shows a UML class diagram of the **`VirtualMachine`** managed object, which has a **`runtime`** property that is defined as an **`xsd:dateTime`** data type. **`VirtualMachine`** also has a **`summary`** property that is a **`VirtualMachineSummary`** data object. The **`VirtualMachineSummary`** data object contains a **`config`** property that is a **`VirtualMachineConfigSummary`** data object.

Figure 8: VirtualMachine Managed Object and Nested Properties



To refer to a nested property, use dot notation to separate the object names in the sequence that defines the path to the property. Your code must handle the type referenced at the end of the sequence.

For example, you can compare the property referenced by the path **summary.config.guestId** (a string value) to the property referenced in the path **summary.config** (the complete **VirtualMachineSummary** data object).

The following table shows examples of property references and the corresponding data types for some of the properties of the **VirtualMachine** managed object shown in Figure: VirtualMachine Managed Object and Nested Properties.

Reference	Data Type
summary	**VirtualMachineSummary** data object
summary.config	**VirtualMachineConfigSummary** data object
summary.config.guestID	string

xsd:anyType Arrays

The vSphere Web Services API uses **xsd:anyType** unconstrained type declarations. A vSphere client must map values of **xsd:anyType** to explicit data types. An **xsd:anyType** value can represent a single data value or it can represent an array. The WSDL for the vSphere Web Services API defines array types for all of the data values that a vSphere client can send or receive as arrays. The array types use the prefix "ArrayOf". An example of an array type is **ArrayOfString** for string values.

When a client sends data to a vSphere Server, the client must use explicit datatypes. For example, a client can define a **MethodAction** for a **ScheduledTask**. The vSphere Web Services API defines arguments to the action (the **MethodActionArgument.value** property) as type **xsd:anyType**. If the action takes an array argument, the client must set the corresponding **MethodAction.argument[].value** to the appropriate ArrayOf... type.

When a client receives **xsd:anyType** data from a vSphere server, it must cast the data to an explicit type. For example, the **PropertyCollector** method **RetrievePropertiesEx** returns a set of **ObjectContent** data objects. The **ObjectContent.propSet** property is a list of **DynamicProperty** objects that contains the requested property values. Each **DynamicProperty** object contains a name-value pair. The value property (DynamicProperty.val) is of type **xsd:anyType**. It can represent a single object or an array of objects.

When the returned value is a single object such as an **Event**, **ManagedObjectReference**, or **String**, you can cast it directly to a variable of the appropriate type. However, when the value is an array of objects you cannot cast the **anyType** value directly to an array variable.

When the **PropertyCollector** returns array data, it sends it as an **xsd:anyType** value. The language-specific bindings contain definitions for array objects such as **ArrayOfEvent**, **ArrayOfManagedObjectReference**, and **ArrayOfString**, and corresponding "get" methods. To extract the actual array from a property of type **xsd:anyType**, cast DynamicProperty.val to the appropriate array type and use the matching get method – for example, **getEvent()**, **getManagedObjectReference()**, or **getString()**.

The following sections provide some examples of how to cast returned values for a few of the array types.

Cast an xsd:anyType Array to an Event Array

When the **PropertyCollector** returns array data representing an array of **Event** objects, the array has the type **xsd:anyType**. To use the data in your application, you must first cast it to a specialized type for **Event** arrays.

This example code uses the JAX-WS-generated Java bindings for the VMware vSphere Web Services SDK WSDL.

1. Use the **DynamicProperty.getVal()** method to retrieve the anyType property value.
2. Cast the **anyType** value to a value of type ArrayOfEvent.
3. Use the corresponding get method to assign the result of the cast operation to a list variable.

```
/*
* Handling arrays of Event objects.
* Cast the return value to ArrayOfEvent and use getEvent().
*/
List[] eventList = ((ArrayOfEvent) dynamicProp.getVal()).getEvent();
```

Cast an `xsd:anyType` Array to an Array of Managed Object References

When the **`PropertyCollector`** returns array data representing an array of managed objects, the array has the type **`xsd:anyType`**. To use the data in your application, you must first cast it to a specialized type for arrays of managed object references.

This example code uses the JAX-WS-generated Java bindings for the VMware vSphere Web Services SDK WSDL.

1. Use the **`DynamicProperty.getVal()`** method to retrieve the `anyType` property value.
2. Cast the **`anyType`** value to a value of type **`ArrayOfManagedObjectReference`**.
3. Use the corresponding get method to assign the result of the cast operation to a list variable.

```
/*
* Handling arrays of ManagedObjectReference objects.
* Cast the return value to ArrayOfManagedObjectReference and use
getManagedObjectReference().
*/
List[] morList =
     ((ArrayOfManagedObjectReference)dynamicProp.getVal()).getManagedObjectReference();
```

Cast an `xsd:anyType` Array to an Array of String

When the **`PropertyCollector`** returns array data representing an array of strings, the array has the type **`xsd:anyType`**. To use the data in your application, you must first cast it to a specialized type for arrays of strings.

This example code uses the JAX-WS-generated Java bindings for the VMware vSphere Web Services SDK WSDL.

1. Use the **`DynamicProperty.getVal()`** method to retrieve the anyType property value.
2. Cast the **`anyType`** value to a value of type **`ArrayOfString`**.
3. Use the corresponding get method to assign the result of the cast operation to a list variable.

```
/*
* Handling arrays of strings.
* Cast the return value to ArrayOfString and use getString().
*/
List[] stringList = ((ArrayOfString) dynamicProp.getVal()).getString();
```

Indexed Array and Key-Based Array Properties

The VMware vSphere data structures include array properties, which can be indexed arrays or key-based arrays.

- **Indexed arrays** are accessed by using an index integer. Indexed arrays are used for arrays of data types whose positions in the array do not change. For example, the **`roleList`** property of the **`AuthorizationManager`** managed object is an array of authorization roles. Adding a new role to the array does not change the position of existing elements in the array.
- **Key-based arrays** are used for information whose position is subject to change. A key-based array (same basic concept as a Perl hash or a Python dictionary) uses a unique, unchanging value as a key to access an element's value.
  Typically, the key is a string, but integers can also be used. For example, **`Event`** arrays use integers as keys.

  The vSphere management object model uses key-based arrays to track managed object references. The contents of a key-based array property are accessed by the value of either the key property or, in the case of a managed object reference, its value property. The value of these fields is unique across all of the components of an array.

  Nested properties can also refer to entries in a key-based array. For example, **`a.b.c["xyz"]`** refers to the property **`c`** that has the key value of **`xyz`**.

Unset Optional Properties

Many of the Data Objects in the vSphere Web Services SDK have optional properties that may be set by your client application or by a Server process or event. If you retrieve a data object that has an optional property that is unset, the Server will not return a value for the optional property. If you call an accessor function to retrieve the property value, the value returned by the function depends on the programming language that you are using.

For example, if you are programming in Java, the value you will receive for an unset property is `null`.

Figure: Data Object - HostFirewallInfo Properties shows part of the Properties table for the `HostFirewallInfo` data object in the *`vSphere Web Services SDK API Reference`*. When you look at properties in the *`vSphere Web Services SDK API Reference`*, you can see that optional properties are marked with a red asterisk.

In this example, that the `defaultPolicy` property is always returned, but the `ruleset` property will be returned as a `null` value if it has not been set.

Figure 9: Data Object - HostFirewallInfo Properties

Properties

NAME	TYPE	DESCRIPTION
defaultPolicy	HostFirewallDefaultPolicy	Default firewall policy.
ruleset*	HostFirewallRuleset[]	List of configured rulesets.
Properties inherited from DynamicData		
dynamicProperty, dynamicType		

* Need not be set

Since Data Objects are part of many different constructs, there is no standard scenario for when an optional property should be set, what will happen if an optional property is left unset, or what you should do if a null value is returned.

Escape Character in Name and Path Properties

The percent sign (%) is used as an escape character to embed special characters in strings. For example, %2f (or %2F) is interpreted as the slash (/) character. To include a percent sign as a literal in a string, use %%.The path to the inventory starts from the root folder (**`ServiceContent.rootFolder`** property), denoted by the slash character.

Character	Description	Representation in URL
%	Percent sign	%25
/	Slash	%2F, %2f
\	Backslash	%5C, %5c
-	Dash	%2D, %2d
.	Dot	%2E, %2e
“	Double quotation mark	%2B, %2b

Client Applications for the Web Services API

Any client application written for the Web Services API must perform certain basic functions. These include making a connection to the server, authenticating and creating a session, and closing the connection.

vCenter Server Connections

Every vCenter Server client application must connect to the server and pass user account credentials to authenticate to the server. After the connection has been established, the client application can use vSphere services to access the virtual environment.

vSphere uses SSL certificates, HTTP tokens, and vCenter Single Sign-On tokens to authenticate a client and support a persistent connection between the client and vCenter Server. The following table provides an overview of these elements.

Security Element	Description
SSL certificates	vSphere Servers use standard X.509 version 3 (X.509v3) certificates to encrypt session information sent over Secure Socket Layer (SSL) protocol connections. In a production environment, client applications verify the vSphere Server certificate during the connection sequence. The examples in this chapter and the examples in the vSphere Web Services SDK accept all certificates.
HTTP tokens	A vSphere Server uses an HTTP token to identify a client session. The Server provides the HTTP token in its response to a client connection request. Subsequent messages between the client and the Server include the HTTP token in the HTTP header.
Client authentication vCenter Single Sign On token	vSphere supports vCenter Single Sign-On. A vCenter client can obtain a vCenter Single Sign-On token from a vCenter Single Sign-On Server and use that token to login to a vCenter Server.
Client authentication username/password	Username/password authentication for client-server connections. A client can present user credentials either directly to vCenter Server to establish a session, or to the vCenter Single Sign-On Service in exchange for a SAML token.

Establishing a Single Sign-On Session with a vCenter Server

vSphere uses single sign-on to provide a single point of authentication for clients. vSphere includes the vCenter Single Sign-On Server. To use vCenter Single Sign-On, your client obtains a SAML token (Security Assertion Markup Language) from the vCenter Single Sign-On Server and passes the token to the vCenter Server in the login request. The token represents the client and contains claims that support client authentication. Components in the vSphere environment perform operations based on the original authentication. For information about obtaining a vCenter Single Sign-On token from the vCenter Single Sign-On Server, see *vCenter Single Sign On Programming Guide*.

To use single sign on, your client calls the **`LoginByToken`** method. Your client must send a SAML token to the vCenter Server by embedding the token in the SOAP header for the **`LoginByToken`** request. During the login sequence, your

client must save and restore the HTTP session cookie. The vCenter Single Sign-On SDK contains sample code that demonstrates how to use the **`LoginByToken`** method.

The following sections describe examples of using the **`LoginByToken`** method to establish a vCenter Single Sign On session with a vCenter Server.

LoginByToken to vCenter Server By Using Java

The following example is based on the `LoginByTokenSample.java` file contained in the vCenter Single Sign On SDK. The SDK contains Java code that supports HTTP and SOAP header manipulation.

vCenter Server Single Sign-On Session Using Java

After you obtain a SAML token from the vCenter Single Sign-On Server, you can use the vSphere Web Services API method **`LoginByToken`** to establish a vCenter Single Sign-On session with a vCenter Server. At the beginning of the session, your client is responsible for the following tasks:

- Insert the vCenter Single Sign-On token and a timestamp into the SOAP header of the **`LoginByToken`** message.
- Maintain the vCenter session cookie. During the login sequence, the Server produces an HTTP session cookie to support the persistent connection. Your client must save this cookie and re-introduce it at the appropriate times.
- If at a later time your client invokes the **`LoginByToken`** method, or other login method, the Server issues a new session cookie in response. You must have a cookie handler in place to save the cookie for subsequent requests.

The example program uses these general steps:

1. Call the **`RetrieveServiceContent`** method to establish an HTTP connection with the vCenter Server and get the Session Manager managed object reference.
2. Call the **`LoginByToken`** method to authenticate the vCenter session. To send the token to the vCenter Server, the client uses a handler to embed the token and a time stamp in the SOAP header for the message. The client uses an HTTP header handler method to extract the session cookie from the vCenter Server response.
3. Restore the session cookie for future requests. To identify the session started with the **`LoginByToken`** method, the client uses a handler to embed the session cookie in the HTTP header.

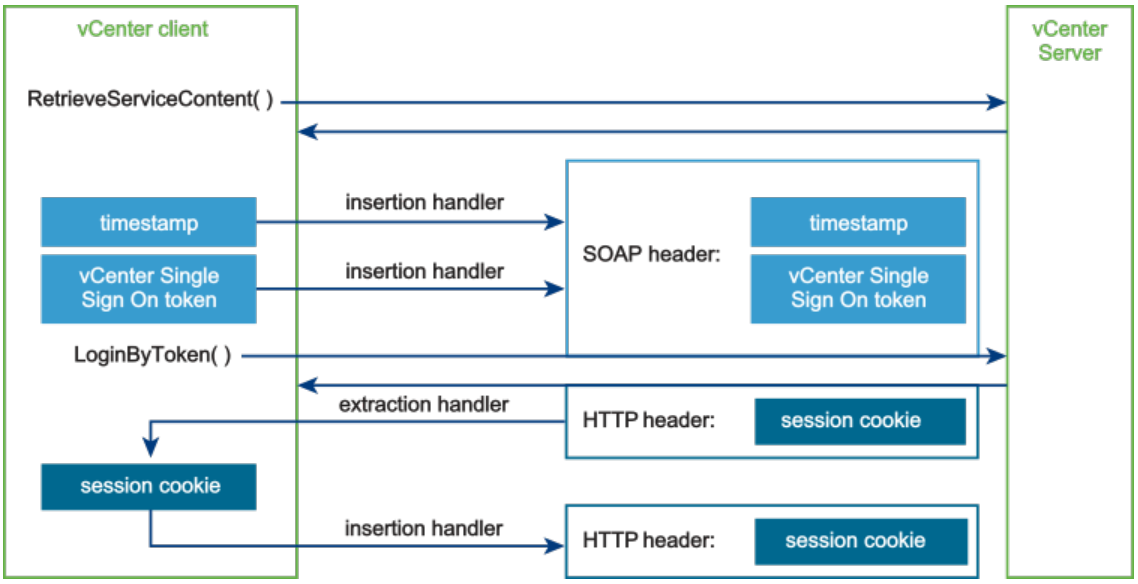
HTTP and SOAP Header Handlers in Java

To use a vCenter Single Sign On token to login to a vCenter Server, the example uses header handlers to manipulates the HTTP and SOAP header elements of the login request. After establishing a handler, subsequent requests automatically invoke the handler.

- Insertion handlers put the vCenter Single Sign On token and a timestamp into the SOAP header into the HTTP header of the login request.
- An extraction handler obtains the HTTP session cookie provided by the vCenter Server. After setting up the handler, a call to the **`LoginByToken`** method will invoke the handler to extract the cookie from the Server response.

The following figure shows the use of handlers to manipulate header elements when establishing a vCenter Single Sign On session with a vCenter Server.

Figure 10: Starting a vCenter Session



IMPORTANT
Every call to the vCenter Server will invoke any message handlers that have been established. The overhead involved in using the SOAP and HTTP message handlers is not necessary after the session has been established. The example saves the default message handler before setting up the SOAP and HTTP handlers. After establishing the session, the example will reset the handler chain and restore the default handler.
The example code also uses multiple calls to the **`VimPortType.getVimPort method`** to manage the request context. The **`getVimPort`** method clears the HTTP request context. After each call to the **`getVimPort`** method, the client resets the request context endpoint address to the vCenter Server URL. After the client has obtained the session cookie, it will restore the cookie in subsequent requests.

Sample Code for a Java Client to the Web Services SDK

The code examples in the following sections show how to use the **`LoginByToken`** method with a holder-of-key security token. The code examples are based on the sample code contained in the vCenter Single Sign On SDK. The files are located in the Java samples directory (`SDK/ssoclient/java/JAXWS/samples`):

- LoginByToken sample:
  `samples/com/vmware/vsphere/samples/LoginByTokenSample.java`
- Header cookie handlers:
  `samples/com/vmware/vsphere/soaphandlers/HeaderCookieHandler.java`

  `samples/com/vmware/vsphere/soaphandlers/HeaderCookieExtractionHandler.java`
- SOAP header handlers. These are the same handlers that are used in the vCenter Single Sign-On example in *`vCenter Single Sign On Programming Guide`*. The SOAP handler files are contained in the vCenter Single Sign-On SDK and are located in the SSO client **`soaphandlers`** directory:
  `SDK/ssoclient/java/JAXWS/samples/com/vmware/sso/client/soaphandlers`

Creating the HTTP Connection in Java

The code fragment in this section establishes an HTTP session with the vCenter Server and saves the HTTP session cookie.

The following sequence describes these steps and shows the corresponding objects and methods.

1	Use the **`getHandlerResolver`** method to save the default message handler. To use the HTTP and SOAP message handlers, you must first save the default message handler so that you can restore it after login. The HTTP and SOAP message handlers impose overhead that is unnecessary after login.	`VimService.getHandlerResolver( )`
2	Get the VIM port. The VIM port provides access to the vSphere API methods, including the **`LoginByToken`** method.	
3	Set the request context endpoint address to the vCenter Server URL.	
4	Retrieve the ServiceContent. This method establishes the HTTP connection.	

The following example shows Java code that saves the session cookie.

Saving the vCenter Server Session Cookie

```
/*
 * The example uses a SAML token (obtained from a vCenter SSO) and vCenter URL.
 * The following declarations indicate the datatypes; the token datatype (Element)
 * corresponds to the token datatype returned by the vCenter Single Sign On Server.
 *
 * Element token;       -- from vCenter Single Sign On Server
 * String vcServerUrl;  -- identifies vCenter Server
 *
 * First, save the default message handler.
 */

HandlerResolver defaultHandler = vimService.getHandlerResolver();

/*
 * Create a VIM service object.
```

```
 */
vimService = new VimService();


/*
 * Construct a managed object reference for the ServiceInstance.
 */
ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
SVC_INST_REF.setType("ServiceInstance");
SVC_INST_REF.setValue("ServiceInstance");

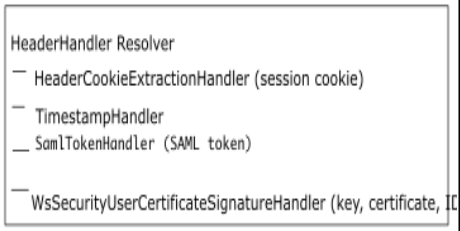

/*
 * Get the VIM port for access to vSphere API methods. This call clears the request
context.
 */
vimPort = vimService.getVimPort();


/*
 * Get the request context and set the connection endpoint.
 */
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, vcServerUrl);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);


/*
 * Retrieve the ServiceContent. This call establishes the HTTP connection.
 */
serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
```

Using LoginByToken in Java

The code fragment in this section sets up the message handlers and calls the **`LoginByToken`** method to get the session cookie. The following sequence describes the steps and shows the corresponding objects and methods.

1	Create a new **`HeaderHandlerResolver.`** Then set the message security handlers for cookie extraction and for inserting the SAML token and credentials in the SOAP header.	
2	Get the VIM port.	
3	Set the connection endpoint in the HTTP request context.	
4	Call the `LoginByToken()` method. The method invocation activates the handlers to insert the elements into the message headers. The method returns a session cookie that identifies the newly created session.	`VimPortType.LoginByToken()`
5	Extract the cookie and save it for later use.	`HeaderCookieExtractionHandler.getCookie()`

The following examples shows Java code that calls the **`LoginByToken()`** method.

Using LoginByToken

```
/*
 * Create a handler resolver and add the handlers.
 * Create a cookie extraction handler and add it to the handler resolver.
 */
HeaderHandlerResolver handlerResolver = new HeaderHandlerResolver();
HeaderCookieExtractionHandler cookieExtractor = new HeaderCookieExtractionHandler();
handlerResolver.addHandler(cookieExtractor);
handlerResolver.addHandler(new TimeStampHandler());
handlerResolver.addHandler(new SamlTokenHandler(token));
handlerResolver.addHandler(new WsSecuritySignatureAssertionHandler(
                                                userCert.getPrivateKey(),
                                    userCert.getUserCert(),
```

```
                                    Utils.getNodeProperty(token, "ID")));
vimService.setHandlerResolver(handlerResolver);

/*
 * Create a handler resolver.
 * Set the VIM service handler resolver.
 */
vimService.setHandlerResolver(handlerResolver);
/*
 *  Get the Vim port; this call clears the request context.
 */
vimPort = vimService.getVimPort();

/*
 * Retrieve the request context and set the server URL.
 */
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, vcServerUrl);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

/*
 * Call LoginByToken.
 */
UserSession us = vimPort.loginByToken(serviceContent.getSessionManager(), null);

/*
 * Save the HTTP cookie.
 */
 String cookie = cookieExtractor.getCookie();
```

Restoring the vCenter Server Session Cookie in a Java Client

After you log in, you must restore the standard vCenter Server session context. The code fragment in this section restores the default message handler and the session cookie. As the cookie handler has been replaced by the default handler, the

client resets the session cookie by calling request context methods to access the context fields directly. The following sequence describes these steps and shows the corresponding objects and methods.

1	Restore the default message handler. The handlers used for **`LoginByToken()`** are not used in subsequent calls to the vSphere API.	`VimService.setHandlerResolver()`
2	Get the VIM port.	
3	Set the connection endpoint in the HTTP request context.	
4	Set the HTTP request header (vCenter Server session cookie).	`RequestContext.get()` `RequestContext.put()`

The following example shows Java code that restores the vCenter Server session. This code requires the vCenter Server URL and the cookie and default handler that were retrieved before login. See Sample Code for a Java Client to the Web Services SDK.

Restoring the vCenter Server Session

```
/*

 * Reset the default handler. This overwrites the existing handlers, effectively removing
them.

 */

vimService.setHandlerResolver(defaultHandler);

vimPort = vimService.getVimPort();


/*

 * Restore the connection endpoint in the request context.

 */

// Set the validated session cookie and set it in the header for once,

// JAXWS will maintain that cookie for all the subsequent requests


Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();

ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, vcServerUrl);

ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);
```

```
/*
 * Reset the cookie in the request context.
 */
Map<String, List<String>> headers =
  (Map<String, List<String>>) ctxt.get(MessageContext.HTTP_REQUEST_HEADERS);
if (headers == null) {
   headers = new HashMap<String, List<String>>();
}
headers.put("Cookie", Arrays.asList(cookie));
ctxt.put(MessageContext.HTTP_REQUEST_HEADERS, headers);
```

Establishing a Session with Username and Password Credentials

Although not as secure as **`LoginByToken`**, you can specify username and password credentials to establish a session with vCenter Server. Follow these steps:

1. Connect to the vSphere Web service, for example with SSL.
2. Create a managed object referring to **`ServiceInstance`**. Use this object to retrieve object **`ServiceContent`** from the server. **`ServiceContent`** contains a reference to the root folder for the inventory and references to managed objects that provide vSphere services. Create another object for access to vSphere API methods.
3. Log in to the server using appropriate credentials (user account, password, and optionally locale).
4. Access server-side objects to retrieve data and perform management operations.
5. Close the connection.

For sample Java code, see topics in *`Getting Started with vSphere APIs`*.

Overview of a Java Sample Application for the Web Services SDK

This section includes an example of a complete client application that demonstrates the basic client capability. The sample client application prints out the product name, server type, and product version to demonstrate that it is connected and able to retrieve information from the server.

While Java Test Client Application is a complete client application that demonstrates the basic client capability, it uses a slightly different format than the Java sample files in the `SDK\` directory. This example, and the Java samples that are included with your vSphere Web Service SDK, have been compiled using JAX-WS bindings.

Most of the vSphere Web Services SDK samples do not handle exceptions, and they accept all security certificates. Use the samples as examples for extracting the types of data you want to view, but do not use these security or exception techniques in your production applications.

Build a Simple vSphere Client Application for the Web Services SDK

This simple client application accepts command-line arguments for the vSphere server name (DNS name or IP address), user name, and password.

To build a simple vSphere client application in Java, use the following steps.

1. Import the vSphere Web Services API libraries:

```
import com.vmware.vim25.*;
```

2. Import the necessary Java (and JAX-WS connection, bindings, and SOAP) libraries:

```
import java.util.*;

import javax.net.ssl.HostnameVerifier;

import javax.net.ssl.HttpsURLConnection;

import javax.net.ssl.SSLSession;

import javax.xml.ws.BindingProvider;

import javax.xml.ws.soap.SOAPFaultException;
```

3. Create the `TestClient` class:

```
public class TestClient {
```

4. Include the class variable declarations/definitions. Use a `TrustManager` class to accept all certificates, as shown in Accessing the vSphere Web Services HTTP Endpoint with JAX-WS . This is not appropriate for a production environment. Production code should implement certificate support.
5. Use the vSphere Web Services APIs to create the connection, as shown in Accessing the vSphere Server from a Web Services Client.
6. Retrieve data from the vSphere or vCenter Server. In this example, we are just going to print out the product name, server type, and product version to prove that the client is connected and working correctly.

```
System.out.println(serviceContent.getAbout().getFullName());

System.out.println("Server type is " + serviceContent.getAbout().getApiType());

System.out.println("API version is " + serviceContent.getAbout().getVersion());
```

7. Use the `VimPort` object to close the connection, as shown in Closing the Connection from a Web Services Client. Always close your server connections to maintain security.

Java Client Example for the Web Services SDK

This example shows the complete sample client application code, without the explanatory steps. The example opens a connection with the server, retrieves the service content, uses the session manager managed object reference to log in, displays information about the server, and closes the connection.

Java Test Client Application

```
import com.vmware.vim25.*;

import java.util.*;

import javax.net.ssl.HostnameVerifier;

import javax.net.ssl.HttpsURLConnection;

import javax.net.ssl.SSLSession;

import javax.xml.ws.BindingProvider;

import javax.xml.ws.soap.SOAPFaultException;
```

```
public class TestClient {

  // Authentication is handled by using a TrustManager and supplying
  // a host name verifier method. (The host name verifier is declared
  // in the main function.)
  private static class TrustAllTrustManager
     implements javax.net.ssl.TrustManager,
                javax.net.ssl.X509TrustManager {

     public java.security.cert.X509Certificate[] getAcceptedIssuers() {
        return null;
     }

     public boolean isServerTrusted(java.security.cert.X509Certificate[] certs) {
        return true;
     }

     public boolean isClientTrusted(java.security.cert.X509Certificate[] certs) {
        return true;
     }

     public void checkServerTrusted(java.security.cert.X509Certificate[] certs,
                                    String authType)
        throws java.security.cert.CertificateException {
           return;
     }

     public void checkClientTrusted(java.security.cert.X509Certificate[] certs,
                                    String authType)
        throws java.security.cert.CertificateException {
           return;
```

```
    }
}// end class TrustAllTrustManager

public static void main(String[] args) {
  try {

     // Server URL and credentials.
     String serverName = args[0];
     String userName   = args[1];
     String password   = args[2];
     String url = "https://"+serverName+"/sdk/vimService";
     // Variables of the following types  for access to the API methods
     // and to the vSphere inventory.
     // -- ManagedObjectReference for the ServiceInstance on the Server
     // -- VimService for access to the vSphere Web service
     // -- VimPortType for access to methods
     // -- ServiceContent for access to managed object services
     ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
     VimService vimService;
     VimPortType vimPort;
     ServiceContent serviceContent;

     // Declare a host name verifier that will automatically enable
     // the connection. The host name verifier is invoked during
     // the SSL handshake.
     HostnameVerifier hv = new HostnameVerifier() {
        public boolean verify(String urlHostName, SSLSession session) {
           return true;
        }
      };
      // Create the trust manager.
      javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
```

```
        javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
        trustAllCerts[0] = tm;

        // Create the SSL context
        javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL");

        // Create the session context
        javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();

        // Initialize the contexts; the session context takes the trust manager.
        sslsc.setSessionTimeout(0);
        sc.init(null, trustAllCerts, null);

        // Use the default socket factory to create the socket for the secure connection
javax.net.ssl.HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
        // Set the default host name verifier to enable the connection.
        HttpsURLConnection.setDefaultHostnameVerifier(hv);

        // Set up the manufactured managed object reference for the ServiceInstance
        SVC_INST_REF.setType("ServiceInstance");
        SVC_INST_REF.setValue("ServiceInstance");

        // Create a VimService object to obtain a VimPort binding provider.
        // The BindingProvider provides access to the protocol fields
        // in request/response messages. Retrieve the request context
        // which will be used for processing message requests.
        vimService = new VimService();
        vimPort = vimService.getVimPort();
        Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();

        // Store the Server URL in the request context and specify true
        // to maintain the connection between the client and server.
```

```
        // The client API will include the Server's HTTP cookie in its
        // requests to maintain the session. If you do not set this to true,
        // the Server will start a new session with each request.
        ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);
        ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

        // Retrieve the ServiceContent object and login
        serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
        vimPort.login(serviceContent.getSessionManager(),
                      userName,
                      password,
                      null);

        // print out the product name, server type, and product version
        System.out.println(serviceContent.getAbout().getFullName());
        System.out.println("Server type is " + serviceContent.getAbout().getApiType());
        System.out.println("API version is " + serviceContent.getAbout().getVersion());

        // close the connection
        vimPort.logout(serviceContent.getSessionManager());
      } catch (Exception e) {
         System.out.println(" Connect Failed ");
         e.printStackTrace();
      }
   }//end main()
}// end class TestClient
```

// See Obtaining a Session Token - Code Fragments from VMPromoteDisks.java for more details.

Compiling the Java Test Client in the Web Services SDK

Use the following command to compile the code for the Java Test Client Application, after you have saved it as a `.java` file:

```
c:>javac -classpath path-to-vim25.jar TestClient.java
```

Use the following command to run the compiled class (binary) file:

```
c:>java -classpath path-to-vim25.jar TestClient web-service-url user-name user-password
```

Web Server Session Token

As with other Web services, the vSphere Web service maintains session state for each client connection by using a token in the HTTP header to identify the session. The vSphere server returns a session token to the client in its response to the client connection request. Subsequent messages between client and server automatically include the token.

Each of the stand-alone samples in the `SDK\vsphere-ws\java\JAX-WS\samples\com\vmware\` uses the JAX-WS TrustAllTrustCertificates class, as discussed in Obtaining a Session Token - Code Fragments from VMPromoteDisks.java to ignore certificates, obtain a session token, and then connect to the server.

> **CAUTION**
> We do not recommend that you trust all certificates in a production environment. Instead, you can look at the sample code to see how the JAX-WS libraries are used when making the connection, but set up an SSL policy that allows connection only with trusted certificates.

The logic for getting a cookie and putting it in the header looks like this:

```
//cookie logic

List cookies = (List) headers.get("Set-cookie");

cookieValue = (String) cookies.get(0);

StringTokenizer tokenizer = new StringTokenizer(cookieValue, ";");

cookieValue = tokenizer.nextToken();

String path = "$" + tokenizer.nextToken();

String cookie = "$Version=\"1\"; " + cookieValue + "; " + path;


// set the cookie in the new request header

Map map = new HashMap();

map.put("Cookie", Collections.singletonList(cookie));


((BindingProvider) vimPort).getRequestContext().put(

MessageContext.HTTP_REQUEST_HEADERS, map);
```

Accessing the vSphere Web Services HTTP Endpoint with JAX-WS

The steps for accessing any HTTP endpoint with JAX-WS bindings include the vSphere Web Services SDK Server URL, vSphere server object, and variables.

These steps are listed at the beginning of Obtaining a Session Token - Code Fragments from VMPromoteDisks.java.

1. Create a `TrustManager` class to handle certificate checking.

   In this example we use a `TrustManager` class to accept all certificates. This is not appropriate for a production environment. Production code should implement certificate support.

```
private static class TrustAllTrustManager
  implements javax.net.ssl.TrustManager,
  javax.net.ssl.X509TrustManager {

   public java.security.cert.X509Certificate[] getAcceptedIssuers() {
      return null;
   }

   public boolean isServerTrusted(
      java.security.cert.X509Certificate[] certs) {
      return true;
   }

   public boolean isClientTrusted(java.security.cert.X509Certificate[] certs) {
      return true;
   }

   public void checkServerTrusted(java.security.cert.X509Certificate[] certs,
                                  String authType)
         throws java.security.cert.CertificateException {
      return;
   }

   public void checkClientTrusted(java.security.cert.X509Certificate[] certs,
                                  String authType)
         throws java.security.cert.CertificateException {
      return;
   }
}
```

2. Include the Server URL and credentials as arguments in the main method:

```
public static void main(String[] args) {
  try {
```

```
String serverName = args[0];
String userName   = args[1];
String password   = args[2];
String url = "https://"+serverName+"/sdk/vimService";
```

3. Declare variables of the following types for access to vSphere server objects:
   - **`ManagedObjectReference`** for the **`ServiceInstance`**.
   - **`VimService`** object for access to the Web service.
   - **`VimPortType`** object for access to all of the methods defined in the vSphere API.
   - **`ServiceContent`** for access to the managed object services on the server.

   The following Java code fragment shows these variable declarations:

   ```
   ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
   VimService vimService;
   VimPortType vimPort;
   ServiceContent serviceContent;
   ```

4. Declare a host name verifier that will automatically enable the connection. The host name verifier is invoked during the SSL handshake.

   ```
   HostnameVerifier hv = new HostnameVerifier() {
     public boolean verify(String urlHostName, SSLSession session) {
        return true;
     }
   };
   ```

5. Instantiate the trust manager object.

   ```
   // Create the trust manager.
   javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
   javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
   trustAllCerts[0] = tm;
   ```

6. Create the SSL context

   ```
   javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL");
   ```

7. Create the session context

   ```
   javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();
   ```

8. Initialize the contexts; the session context takes the trust manager.

```
sslsc.setSessionTimeout(0);

sc.init(null, trustAllCerts, null);
```

9. Use the default socket factory to create the socket for the secure connection

   ```
   javax.net.ssl.HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
   ```

10. Set the default host name verifier to enable the connection.

    ```
    HttpsURLConnection.setDefaultHostnameVerifier(hv);
    ```

Accessing the vSphere Server from a Web Services Client

The steps that use the vSphere Web Services API to create the connection are:

1. Create a managed object reference for the **`ServiceInstance`** object on the server.

   ```
   ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();

   SVC_INST_REF.setType("ServiceInstance");

   SVC_INST_REF.setValue("ServiceInstance");
   ```

2. Create a `VimService` object to obtain a `VimPort` binding provider. The `BindingProvider` object provides access to the protocol fields in request/response messages. Retrieve the request context which will be used for processing message requests.

   The **`VimServiceLocator`** and **`VimPortType`** objects provide access to vSphere servers. The **`getVimPort`** method returns a **`VimPortType`** object that provides access to the vSphere API methods.

   ```
   vimService = new VimService();

   vimPort = vimService.getVimPort();

   Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
   ```

3. Store the Server URL in the request context and specify `true` to maintain the connection between the client and server. The client API will include the server's HTTP cookie in its requests to maintain the session. If you do not set this to true, the server will start a new session with each request.

   ```
   ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);

   ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);
   ```

4. Retrieve the **`ServiceInstance`** content (the **`ServiceContent`** data object) and log in to the server.

   ```
   serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);

   vimPort.login(serviceContent.getSessionManager(),

                 userName,

                 password,

                 null);

   isConnected = true;
   ```

Closing the Connection from a Web Services Client

Use the `VimPort` object again to close the connection. Always close your server connections to maintain security.

```
         vimPort.logout(serviceContent.getSessionManager());
      } catch (Exception e) {
         System.out.println(" Connect Failed ");
         e.printStackTrace();
      }
   }//end main()
}// end class TestClient
```

Using the Java Samples as Reference

The following code fragment from the vSphere Management SDK shows another implementation of the server connection. You can use code similar to this sample in the vSphere Web Services SDK (path below) to get a session token for your client application.

`SDK/vsphere-ws/java/JAX-WS/samples/com/vmware/vm/VMPromoteDisks.java`

Obtaining a Session Token - Code Fragments from VMPromoteDisks.java

```
   ...
   private static String cookieValue = "";
   private static Map headers = new HashMap();
   ...
   private static void trustAllHttpsCertificates()
      throws Exception {

      javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
      javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
      trustAllCerts[0] = tm;
      javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL");
      javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();
      sslsc.setSessionTimeout(0);
      sc.init(null, trustAllCerts, null);
      javax.net.ssl.HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
   }
   ...
```

```
private static void connect()
   throws Exception {

   HostnameVerifier hv = new HostnameVerifier() {
      public boolean verify(String urlHostName, SSLSession session) {
         return true;
      }
   };
   trustAllHttpsCertificates();
   HttpsURLConnection.setDefaultHostnameVerifier(hv);

   SVC_INST_REF.setType(SVC_INST_NAME);
   SVC_INST_REF.setValue(SVC_INST_NAME);

   vimService = new VimService();
   vimPort = vimService.getVimPort();
   Map<String, Object> ctxt =
      ((BindingProvider) vimPort).getRequestContext();

   ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);
   ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

   serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
   headers =
      (Map) ((BindingProvider) vimPort).getResponseContext().get(
               MessageContext.HTTP_RESPONSE_HEADERS);
   vimPort.login(serviceContent.getSessionManager(),
                 userName,
                 password, null);
   isConnected = true;

   propCollectorRef = serviceContent.getPropertyCollector();
```

```
      rootRef = serviceContent.getRootFolder();
   }
   ...
```

Multiple Versions of the vSphere API

When a client application connects to a Web service running on a vSphere server (ESXi or vCenter Server system), the server detects the version of the API that was used to develop the client and makes available only those operations supported by the client.

Client applications convey information about the API version used in the SOAP messages that they send to a vSphere server. These SOAP messages include a **`versionID`** in the **`soapAction`** attribute. The details are handled transparently by the SOAP toolkit and the client proxy code. The server adjusts its behavior based on the client's version information, exposing the API version that the client supports to the client.

Starting with vSphere 4.0, information about the supported API versions is contained in an XML file, **`vimServiceVersions.xml`**, located on the server (see Service-Versions File (vimServiceVersions.xml)). You can access this file with the URL https:// *`server_hostname`*`/sdk/vimServiceVersions.`**`xml`**.

Service-Versions File (vimServiceVersions.xml)

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!--    Copyright 2008-2010 VMware, Inc.  All rights reserved. -->
- <namespaces version="1.0">
     - <namespace>
            <name>urn:vim25</name>
            <version>5.0</version>
     - <priorVersions>
          <version>2.5u2</version>
          <version>2.5</version>
       </priorVersions>
     </namespace>
     - <namespace>
          <name>urn:vim2</name>
          <version>2.0</version>
       </namespace>
  </namespaces>
```

If you are developing a client application that must support multiple server versions at the same time (ESXi 5.0 and ESXi 5.5, for example), you must obtain information about the API versions that are supported on the server and provide logic in your code to use or not use features, based upon the version information.

One approach to targeting multiple versions of the API from the same client application code is to check for the existence of the server versions file on the server. If you do not find a `vimServiceVersions.xml` file on the server, the server is older than ESX/ESXi 4.x, vCenter Server 4.x.

Java Sample Applications in the Web Services SDK

The vSphere Web Services SDK includes sample applications, written in Java, that demonstrate features of the vSphere API and object model. Most of the samples do not handle exceptions, and they accept all security certificates. So use the applications as examples for extracting the types of data you want to view, but do not use the helper classes, trust store methods or exception handling techniques in your production environment.

Java Samples in the Web Services SDK

The Java samples in your vSphere Web Services SDK include `.java` files that you can compile and then run using any Java editor or IDE. The samples accept command-line arguments for the vSphere server name (DNS name or IP address), user name, and password.

Client Applications Using the JSON Protocol

You can create client applications for the Web Services API using the JSON data protocol instead of XML. JSON offers functionality similar to the SOAP protocol, but with a REST-like interface and on a separate service endpoint. JSON clients must perform functions such as authenticating a session, retrieving data, and ending the session. This chapter outlines how to code basic functions using Python to drive the JSON protocol.

Client Requirements for the JSON Protocol

A JSON protocol client needs a mid-level capability to build HTTP request messages, send the requests over the network, and manage server responses. Many languages have libraries to abstract TCP/IP layers while also providing access to URLs, headers, and bodies.

For example, some popular libraries include:

Python 3 – `urllib.request` module

Java – `java.net.HttpUrlConnection` or for Java 11 and later, `java.net.HTTP` module

Go – `net/http` package

Rust – `http` module

Perl – `LWP` module

Swift – `URL`, `URLRequest`, `URLSession`, and `URLSessonDataTask` interfaces

C++ – `libcurl` or `curlpp`

Command line and graphical tools for passing HTTP messages include:
- Curl or **`wget`**
- Postman

- Insomnia
- Hoppscotch

To use the JSON protocol, you build URLs that embed the names of managed objects, properties, and methods. You build request bodies that specify the parameters for method calls. You read response bodies and response headers. You include a session identifier in your request headers.

Outlining a Client for the JSON Protocol

A client to work with the JSON protocol generally contains the following parts, at minimum.

- The basic elements of the URL, including the address of the vCenter Server instance, the service endpoint specifier, and the version specifier.
- The credentials with which you authenticate your client session.
- Code to GET managed object properties, such as the `ServiceInstance` content, which contains managed object references to singletons, including the `SessionManager`.
- Code to POST requests that invoke methods on managed objects, such as `LoginByToken`.
- A request to fetch the `ServiceInstance` content and extract the `SessionManager` managed object reference.
- A request to authenticate the client session with `SessionManager`.
- A request to end the client session.

Following sections use fragments of Python code to show how you can build these parts of the request.

Building JSON Request URLs

The service endpoint for the JSON API is determined by the vCenter Server address and the API version that the client prefers for the response. These data form part of the service URL.

All requests in the JSON API specify a managed object. Some requests retrieve properties of a managed object. Other requests invoke a method on a managed object.

The parts of the service URL are as follows:

1. `scheme = 'https://'`
2. `domain = vCenter Server IP address or FQDN`
3. The URL path contains the following parts:
    a. `endpoint = '/sdk/vim25'`
    b. `version = '8.0.1.0'`
    c. A managed object reference, generated by the server, which has two subparts, `type` and `value`, where:
        a. `type` is the name of a type of managed object, such as `Folder` or `VirtualMachine` or `ServiceInstance`
        b. `value` is a unique identifier (within this vCenter Server instance) for a specific object of the type named, such as `group-d1` or `vm-015` or `ServiceInstance`.
        The object notation must be serialized as two strings separated by a slash (`/`). For example: `'Folder/group-d1'`.
        ```
        def serialize_moref_for_request_url(moref):

          return('{}/{}'.format(moref['type'], moref['value']))
        ```
    d. A property name or method name, such as `childType` or `CreateVM`

An example request URL to retrieve the type or types of managed objects that a folder can contain::

```
>>> scheme = 'https://'
>>> domain = 'vcenter.example.com'
>>> endpoint = '/sdk/vim25'
>>> version = '8.0.1.0'
>>> motype = 'Folder'
>>> moid = 'group-d1'
>>> moref = motype + '/' + moid
>>> property = 'childType'
>>> url = scheme + domain + endpoint + '/' + version + '/' + moref + '/' + property
>>> print(url)
https://vcenter.example.com/sdk/vim25/8.0.1.0/Folder/group-d1/childType
```

An example request URL to cancel a long-running operation:

```
...
>>> motype = 'Task'
>>> moid = 'task-124'
>>> moref = motype + '/' + moid
>>> method = 'Cancel'
>>> url = scheme + domain + endpoint + '/' + version + '/' + moref + '/' + method
>>> print(url)
https://vcenter.example.com/sdk/vim25/8.0.1.0/Task/task-124/Cancel
```

Building JSON Request Bodies

Methods are invoked with the JSON protocol by sending a POST request with the method name in the URL and the method parameters in the body of the request.

The request body for a method invocation is a single JSON object containing the method parameters as named objects or arrays. For instance, a request body for the `MoveIntoFolder_Task` method might look like this:

```
{'list': [
    {'_typeName': 'ManagedObjectReference',
    'type': 'VirtualMachine',
    'value': 'vm-23'},
    {'_typeName': 'ManagedObjectReference',
    'type': 'VirtualMachine',
    'value': 'vm-96'},
    {'_typeName': 'ManagedObjectReference',
```

```
    'type': 'VirtualApp',
    'value': 'resgroup-v44'}
  ]
}
```

The method name itself becomes part of the URL, as described in Building JSON Request URLs.

> **NOTE**
> Use the WSDL forms of type names. WSDL names are shown in the *vSphere Web Services API* reference. For example, use the WSDL name `ApplyHostProfileConfigurationSpec`, rather than the package name `vim.profile.host.ProfileManager.ApplyHostConfigSpec`, to build a data object as a parameter to the method `ApplyEntitiesConfig_Task`. Similarly, use the method's WSDL name, `ApplyEntitiesConfig_Task`, rather than the source name, `applyEntitiesConfiguration`.

Retrieving the Service Instance with the JSON Protocol

`ServiceInstance` is a singleton managed object that acts as a gateway to the resources available on the server. You can locate other important singletons such as `SessionManager` and `SearchIndex` from `ServiceInstance` properties. `ServiceInstance` is accessible without authentication, so you can retrieve a reference to the `SessionManager` which you can use to authenticate a session.

You need the IP address or FQDN of a vCenter Server to which you will send the request.

To retrieve the properties of the `ServiceInstance` managed object, do the following.

1. Synthesize a managed object reference to the `ServiceInstance` managed object. This managed object reference always has a fixed value of `{type: "ServiceInstance", value: "ServiceInstance"}` because it is a prerequisite for access to other managed object references.
   ```
   motype = 'ServiceInstance'

   moid = 'ServiceInstance'

   moref = motype + '/' + moid
   ```
2. Specify the property to retrieve from the managed object.
   ```
   property = 'content'
   ```
3. Assemble the URL from the pieces previously defined.
   ```
   scheme = 'https://'

   domain = 'vc.example.com'

   endpoint = '/sdk/vim25'

   version = '8.0.3'

   url = scheme + domain + endpoint + '/' + version + '/' + moref + '/' + property

   print(url)
   href="https://vc.example.com/sdk/vim25/8.0.3/ServiceInstance/ServiceInstance/content"
   ```
4. Send the URL in an HTTP GET request.
   ```
   from urllib.request import urlopen, Request

   import ssl
   ```

```
# Skip certificate verification on test systems only:
unverified_context = ssl._create_unverified_context()
request = Request( url, headers=headers or {} )
with urlopen( request, context=unverified_context ) as response :
  response_headers = response.headers
  response_body = response.read()
```

5. Display the result of the query.

```
from pprint import pprint
json_body = json.loads(response_body)
pprint(json_body)
```

The program displays a number of managed object references for singletons that you can use for additional requests. The following excerpt includes several important managed objects that you will use for basic operations, including authentication.

Excerpt from Service Content

```
...
'propertyCollector': {'_typeName': 'ManagedObjectReference',
                      'type': 'PropertyCollector',
                      'value': 'propertyCollector'},
'rootFolder': {'_typeName': 'ManagedObjectReference',
               'type': 'Folder',
               'value': 'group-d1'},
'scheduledTaskManager': {'_typeName': 'ManagedObjectReference',
                         'type': 'ScheduledTaskManager',
                         'value': 'ScheduledTaskManager'},
'searchIndex': {'_typeName': 'ManagedObjectReference',
                'type': 'SearchIndex',
                'value': 'SearchIndex'},
'serviceManager': {'_typeName': 'ManagedObjectReference',
                   'type': 'ServiceManager',
                   'value': 'ServiceMgr'},
'sessionManager': {'_typeName': 'ManagedObjectReference',
```

```
                    'type': 'SessionManager',
                    'value': 'SessionManager'},
...
```

Use the `SessionManager` URL in the Service Content to open an authenticated session with the vCenter Server instance.

Authenticating a JSON Client with the Session Manager

Most method calls must carry a session ID to authenticate with the server at the time of the call. The session ID is a temporary substitute for username and password, thereby limiting risk to the principal's credentials.

To acquire a session ID, you have these options:

- Invoke the `SessionManager.Login` method with username and password, exchanging the principal's credentials for a session ID token.
- Request a SAML token from the vCenter Single-Sign-On server, and use the token to invoke the `SessionManager.LoginByToken` method, receiving a session ID token in exchange.

After a successful authentication operation with vCenter Server, you extract the session ID token from the HTTP header of the response message. Include the same header in subsequent API request messages.

For example, the response headers might contain lines similar to the following:

```
content-type: application/json

date: Tue, 14 Aug 2007 13:30:00 GMT

vmware-api-session-id: 0123498765fabdec526374859607182930 4100ab

x-envoy-upstream-service-time: 45
```

The value `0123498765fabdec526374859607182930 4100ab` is the session ID token. Use it in the headers of your API requests, like this:

```
content-type: application/json

vmware-api-session-id: 0123498765fabdec526374859607182930 4100ab
```

Basic Authentication for the JSON Protocol

Basic authentication relies on username and password. These credentials are transported over HTTPS with TLS encryption, but for stricter security consider using SAML token authentication instead.

Basic authentication requires that you have a username and corresponding password that have privileges on the vCenter Server instance to which you connect. You also need the IP address or fully qualified domain name of the vCenter Server instance.

To authenticate with username and password, follow these steps.

1. Prepare a JSON serialization of your credentials. This will be the body of your request.
   ```
   body_string = '{"userName": "Administrator@vsphere.local", "password":
   "betyoucantguess"}'
   ```
2. Get the `SessionManager` managed object reference from the `ServiceInstance` content. Format it for the resource portion of the service URL.
   ```
   moref = si_content['sessionManager']

   url_moref = '{}/{}'.format(moref['type'], moref['value'])
   ```
3. Assemble the complete URL for a request to invoke the Login method of `SessionManager`.

```
scheme = 'https://'

domain = 'vcenter.example.com'

endpoint = '/sdk/vim25'

version = '8.0.1.0'

method = 'Login'

url = scheme + domain + endpoint + '/' + version + '/' + url_moref + '/' + method

print(url)
https://vcenter.example.com/sdk/vim25/8.0.1.0/SessionManager/SessionManager/Login
```

4. Set the request header `'Content-Type' = 'application/json'`.
```
request_headers = {}

request_headers['Content-Type'] = 'application/json'
```
5. Send the URL and the body in an HTTP POST request.
```
from urllib.request import urlopen, Request

import ssl

# Skip certificate verification on test systems only:

unverified_context = ssl._create_unverified_context()

request = Request(url, headers=request_headers)

with urlopen(request, context=unverified_context) as response :

  response_headers = response.headers

  response_body = response.read()
```
6. Extract the session token from the authentication header in the response.
```
token = response_headers['vmware-api-session-id']
```

The token you extracted is your current session ID and is required to access most managed object properties and methods with the API.

Save the token. Add the token header to all subsequent requests before sending.
```
request_headers['vmware-api-session-id'] = token
```

SAML Token Authentication for a JSON Client

To authenticate by SAML token, you must use SOAP message protocol rather than the JSON protocol. After authentication, you can send authenticated requests over the JSON protocol.

SAML token authentication using the SOAP protocol is described in the *vCenter Single Sign-On Programming Guide*. Use the Single-Sign-On server to exchange the principal's credentials for a SAML token, then invoke the `LoginByToken` method of the `SessionManager` to exchange the SAML token for a session ID token.

The `SessionManager` returns the session ID token in a cookie header. Use the value of the cookie as the value of `vmware-api-session-id` in request message headers, as described in Authenticating a JSON Client with the Session Manager.

Mapping Methods for the JSON Protocol

All methods defined in the WSDL schema for the vSphere Web Services API can be invoked by using a REST-like URL with the POST verb. The method name appends to the URL and the method parameters appear as JSON in the body of the request.

Method names begin with an uppercase letter and match the method name in the vSphere Web Services API Reference. All methods attach to a managed object reference, as in the SOAP API.

For example, the `SessionManager` managed object has a method whose WSDL name is `AcquireLocalTicket`.

The URL format that you use to invoke a method is
`https://vcenter_fqdn/sdk/vim25/8.0.1.0/mo_ref/method_name`

For example, you invoke the `AcquireLocalTicket` method with a URL like this:

`https://vcenter_fqdn/sdk/vim25/8.0.1.0/SessionManager/SessionManager/AcquireLocalTicket`

When you POST a request to invoke a method on a managed object, the body of the request contains a single anonymous JSON object that specifies the parameter names and values. Unlike the SOAP request protocol, the JSON does not specify a reference to "this" because the managed object reference is in the URL path.

For example, the method `Folder.CreateFolder` requires a name for the new folder. The body for that request would look similar to this:

```
{"name": "spare_VM_folder"}
```

Accessing Managed Object Properties with the JSON Protocol

You retrieve a property of a managed object by appending the property name to the URL, after the managed object reference. Property names always begin with a lowercase letter, matching the name in the vSphere Web Services API Reference.

For example, you can retrieve the string property `HostProfile.validationState` with a URL similar to this:
`https://vcenter_fqdn/sdk/vim25/8.0.1.0/hostprofile_mo_ref/validationState`

> **NOTE**
> You cannot modify properties directly. To modify properties of a managed object, you must invoke a method of the managed object or a related managed object that affects its state on the server.

Retrieving Nested Properties

You can retrieve a top-level property by naming it in the URL, but nested properties are not directly accessible. Many top-level properties are data objects or arrays, containing nested properties of interest. To access nested properties, first retrieve the top-level property, then deserialize the JSON in the response body. After deserialization, you can access the nested properties of the data object or array in the client.

For example, code this request:

```
GET https://vcenter_fqdn/sdk/vim25/8.0.1.0/vm_mo_ref/runtime
```

Then extract and display the nested property like this:

```
data_object = json.loads(response_body)
```

```
state = data_object['powerState']

print('VM powerState = ' + state)
```

The nested property displays.

```
VM powerState = poweredOff
```

Retrieving Managed Object Reference Properties

Managed objects often have properties to reference other managed objects. In these cases, you can use a GET request to obtain the managed object reference, but to access the properties or methods of the other managed object you must issue a new GET request, using the managed object reference in the request URL.

If you need to follow a chain of managed object references, you can avoid extra requests by using the `PropertyCollector` managed object. The purpose of the `PropertyCollector` is to delegate the search functions to the server, where they can be done more efficiently.

Using the Property Collector to Retrieve Object Properties

You can use the `PropertyCollector` to retrieve properties from one or more managed objects, whether or not they are related to each other, if you can specify the set of objects in your request. The `PropertyCollector` provides a flexible interface by which you can specify search criteria to execute in the server.

To specify a property retrieval, several approaches are available to you. A few of them are:

- Provide a managed object reference and a set of its properties that you want to retrieve.
- Provide a managed object reference as a starting point for a search that traverses inventory paths to find related managed objects and return their properties.
- Provide a reference to a previous search set as a starting point for a traversal search, often directed to the managed objects contained in the set.

This example shows a hybrid approach where you first locate a `Datacenter` managed object by using the `SearchIndex` method. Then you use that managed object as a starting point to traverse to `VirtualMachine` objects in the data center's virtual machine folder. If there are subfolders within the virtual machine folder, the search specifications traverse the subfolders recursively.

The goal of this example is to report whether the virtual machines are powered on or powered off. That information is in the `VirtualMachine.runtime.powerState` property.

The `powerState` property is a nested property of the virtual machine, so you need to locate the `VirtualMachine` managed objects and specify a property within the `runtime` data object belonging to each `VirtualMachine`. The JSON protocol has the ability to retrieve the entire runtime data object as a single property, but the `PropertyCollector` is able to extract the nested property in a single query.

Use the following procedure to build a server-side specification that retrieves the power states of virtual machines in the data center's virtual machine folder.

> **NOTE**
> For simplicity, this procedure ignores vApps.

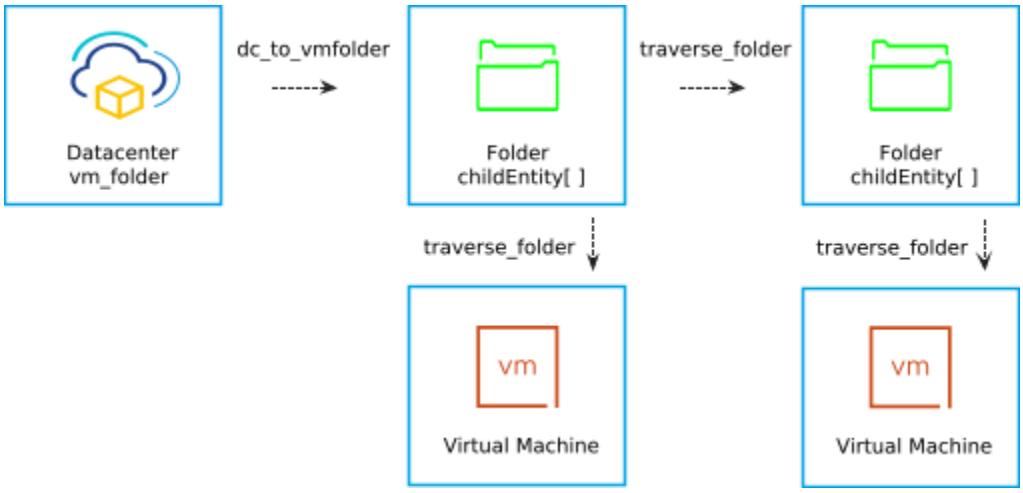
1. Build a `PropertySpec` data object to identify the managed object type (`VirtualMachine`) and the nested property name (`runtime.powerState`).

```
propSet = [ {'_typeName': 'PropertySpec',

             'type': 'VirtualMachine', 'pathSet': ['runtime.powerState'], 'all':
False}]
```

The `all` property, if `True`, overrides the `pathSet` property and collects all properties of the managed object. This behavior is useful if you want to maintain a client-side copy in synch with the server-side managed object.

2. (Optional) If it helps to clarify the inventory paths to the virtual machines, sketch `VirtualMachine` objects and their parent managed objects, all the way up the chain to the starting `Datacenter` managed object.



For each path between managed objects, note the name of the property in the parent object that links to the child object. You must create a `TraversalSpec` (or a `SelectionSpec` reference to a `TraversalSpec`) to follow each of those links to the next managed object. The `VirtualMachine` managed objects whose properties you want to retrieve do not need `TraversalSpec` objects.

3. Working backwards from `VirtualMachine` objects to `Datacenter` object, create a named `TraversalSpec` data object for each path segment.

```
traverse_folder = {'_typeName': 'TraversalSpec'}

traverse_folder['name'] = 'traverse_folder'

traverse_folder['type'] = 'Folder'

traverse_folder['path'] = 'childEntity'

traverse_folder['skip'] = False


dc_to_vmfolder = {'_typeName': 'TraversalSpec'}

dc_to_vmfolder['name'] = 'dc_to_vmfolder'

dc_to_vmfolder['type'] = 'Datacenter'

dc_to_vmfolder['path'] = 'vmFolder'

dc_to_vmfolder['skip'] = True
```

You do not need a `TraversalSpec` for the `VirtualMachine` object, because it is a leaf object in the traversal tree. You do need a `TraversalSpec` for the `Datacenter` object and the `Folder` object, so the `PropertyCollector` will know how to traverse down the chain to the child objects. The `skip` property is an optimization. It tells the `PropertyCollector` that the next managed object after the traversal can be traversed without checking the `PropertySpec`.

NOTE
The `TraversalSpec.selectSet[]` property is not yet set. You will use it to link the specs together.

4. If your data center might have nested folders inside its VM folder, create a `SelectionSpec` data object that specifies the name of the `TraversalSpec` for `Folder` objects, so the `PropertyCollector` can follow the `childEntity` path recursively for any nested folders it finds.
```
folder_recursion = {'_typeName': 'SelectionSpec', 'name': 'traverse_folder'}
```
5. Link the `TraversalSpec` objects together by using the `selectSet[]` property to build a managed object traversal chain.
```
traverse_folder['selectSet'] = [folder_recursion]

dc_to_vmfolder['selectSet'] = [traverse_folder]
```
6. Create an `ObjectSpec` data object to specify starting from your `Datacenter` managed object, and to direct the `PropertyCollector` to the `TraversalSpec` that leads to the top-level VM folder.
```
objectSet = [{'_typeName': 'ObjectSpec',

              'obj': dc_moref, 'skip': False,

              'selectSet': [dc_to_vmfolder] }]
```
The `skip` property in the `ObjectSpec` is different from the `skip` property in a `TraversalSpec`. In the `ObjectSpec`, it tells the `PropertyCollector` that the starting object is not one of the managed objects it is looking for.
7. Combine the `PropertySpec` and the `ObjectSpec` into a `PropertyFilterSpec` that will be a parameter to the `PropertyCollector.RetrievePropertiesEx` method.
```
specSet = [{'_typeName': 'PropertyFilterSpec',

            'propSet': propSet, 'objectSet': objectSet}]
```
8. Specify the `RetrievePropertiiesEx` method parameters in the request body.
```
import json

param_dict = {'options': {'_typeName': 'RetrieveOptions',

                          'maxObjects': 1000}, 'specSet': specSet}

request_body = json.dumps(param_dict).encode('utf-8')
```
9. Send the request, using the POST verb.
```
from urllib.request import urlopen, Request

import ssl

moref = si_content['propertyCollector']

method = 'RetrievePropertiesEx'

url = make_url(domain, mo=moref, m_p=method)

request_headers = {}

request_headers['vmware-api-session-id'] = token

request_headers['Content-Type'] = 'application/json'

headers['Content-Type'] = 'application/json'

# Skip certificate verification on test systems only:
```

```
  unverified_context = ssl._create_unverified_context()
  request = Request(url, headers, data=body)
  with urlopen(request, context=unverified_context) as response :
    response_headers = response.headers
    response_body = response.read()
  json_body = json.loads(response_body)
  print(json_body)
```

The results are returned in a `RetrieveResult` data object, which encloses an array of `ObjectContent` data objects. Each `ObjectContent` contains a managed object reference and a corresponding array of named properties found by the `PropertyCollector` that match your `PropertySpec`. For this example, the managed object references identify `VirtualMachine` objects, and the property name-value pairs show the `powerState` of each `VirtualMachine`.

Using Polymorphic Types with the JSON Protocol

The JSON protocol deals with managed objects whose properties include WSDL-defined data objects, such as configuration objects. Method parameters can also be data objects. Some of these data objects are polymorphic, in the sense that they can inherit from other data objects and can be referenced as the parent type.

The JSON protocol distinguishes the assigned type by adding a `_typename` property to the JSON serialization. For example, a `TaskInfo` data object contains a `result` property that has the XSD type `xsd:anyType`. The actual type of the assigned value is specified with a `_typename` property in `result`.

For example, the following JSON shows how a result data object is returned in the server response and the assigned type is identified by the `_typeName` property.

```
"result": {
    "_typeName": "ManagedObjectReference",
    "_value": {
        "value": "domain-s16",
        "type": "ComputeResource"
    }
}...
```

When you use a polymorphic type in a request, you must add a `_typename` property to identify the assigned type for the server. For example, if you create a `PropertyFilter` for a request to the PropertyCollector, you might include a `TraversalSpec` object, which inherits from the `SelectSet` object. Add the `_typename` property to distinguish a `TraversalSpec` value from a `SelectSet` value:

```
"traverse_vm_folder": {
   "_typeName": "TraversalSpec",
   "type": "Folder",
   "path": "childEntity",
```

```
   "skip": false
}
```

Many properties are defined in the WSDL as inheriting from `DynamicData`. For that reason, you will use `_typeName` frequently. A best practice is to add the `_typeName` property to all objects you create for the JSON API.

Python Client Example for the JSON Protocol

This example illustrates several basic operations needed to implement a client that uses the JSON protocol.

```
import json
from pprint import pprint


domain = 'vcenter.example.com'
credentials = {'userName': 'Administrator@vsphere.local', 'password': 'betyoucantguess'}
dc_name = 'DC-1'


def make_url(vc, ver='8.0.1.0', mo='ServiceInstance/ServiceInstance', m_p='content'):
  if isinstance(mo, dict):
    mo = '{}/{}'.format(mo['type'], mo['value'])
  url = 'https://{}/sdk/vim25/{}/{}/{}'.format(vc, ver, mo, m_p)
  return url


def send_request(url, headers=None, body=None):
  from urllib.request import urlopen, Request
  import ssl
  if headers is None: headers = {}
  headers['Content-Type'] = 'application/json'
  # Skip certificate verification on test systems only:
  unverified_context = ssl._create_unverified_context()
  request = Request(url, headers=headers or {}, data=body)
  with urlopen( request, context=unverified_context ) as response :
    response_headers = response.headers
    response_body = response.read()
  return( response_headers, response_body )
```

```
def get_service_content(vc):
  url = make_url(vc)
  (response_headers, response_body) = send_request(url)
  json_body = json.loads( response_body )
  return( json_body )


def basic_auth(domain, credentials):
  si_content = get_service_content(domain)
  moref = si_content['sessionManager']
  method = 'Login'
  url = make_url(domain, mo=moref, m_p=method)
  request_headers = {}
  request_headers['Content-Type'] = 'application/json'
  request_body = json.dumps(credentials).encode('utf-8')
  (response_headers, response_body) = send_request(url, headers=request_headers,
body=request_body)
  print('Response:')
  print(response_headers)
  print(json.loads(response_body))
  token = response_headers['vmware-api-session-id']
  return(token)


def logout(domain, session_moref) :
  request_body = {}
  method = 'Logout'
  url = make_url(domain, mo=session_moref, m_p=method)
  request_headers = {}
  request_headers['vmware-api-session-id'] = token
  request_headers['Content-Type'] = 'application/json'
  (response_headers, response_body) = send_request(url, headers=request_headers,
body=request_body)
  return(response_headers)
```

```
def get_datacenter_moref(domain, dc_name, token):
  si_content = get_service_content(domain)
  moref = si_content['searchIndex']
  method = 'FindByInventoryPath'
  url = make_url(domain, mo=moref, m_p=method)
  body_dict = {'inventoryPath': dc_name}
  request_body = json.dumps(body_dict).encode('utf-8')
  request_headers = {}
  request_headers['vmware-api-session-id'] = token
  request_headers['Content-Type'] = 'application/json'
  (response_headers, response_body) = send_request(url, headers=request_headers,
body=request_body)
  json_body = json.loads(response_body)
  return(json_body)


def get_vm_power_states(domain, dc_moref, token):
  # MO type & properties to retrieve.
  propSet = [ {'_typeName': 'PropertySpec',
               'type': 'VirtualMachine', 'pathSet': ['runtime.powerState'], 'all': False}]
  # Work backwards to define the steps in the traversal path.
  # traversalspec = type to traverse, path out, skip next?, (selectSet).
  # Traverse any nested folders inside vmFolder:
  traverse_folder = {'_typeName': 'TraversalSpec'}
  traverse_folder['name'] = 'traverse_folder'
  traverse_folder['type'] = 'Folder'
  traverse_folder['path'] = 'childEntity'
  traverse_folder['skip'] = False
  traverse_folder['selectSet'] = [{'_typeName': 'SelectionSpec',
                                   'name': 'traverse_folder'}]
  # Traverse Datacenter to vmFolder:
  dc_to_vmfolder = {'_typeName': 'TraversalSpec'}
  dc_to_vmfolder['name'] = 'dc_to_vmfolder'
  dc_to_vmfolder['type'] = 'Datacenter'
```

```
  dc_to_vmfolder['path'] = 'vmFolder'
  dc_to_vmfolder['skip'] = True
  dc_to_vmfolder['selectSet'] = [traverse_folder]
  # objectSet = Starting MO + traversal steps to retrieval objects.
  objectSet = [{'_typeName': 'ObjectSpec',
                'obj': dc_moref, 'skip': False,
                'selectSet': [dc_to_vmfolder]}]
  # specSet = total filter specification.
  specSet = [{'_typeName': 'PropertyFilterSpec',
              'propSet': propSet, 'objectSet': objectSet}]
  # Params to RetrievePropertiesEx():
  param_dict = {'options': {'_typeName': 'RetrieveOptions',
                            'maxObjects': 1000}, 'specSet': specSet}
  # Build request message.
  request_body = json.dumps(param_dict).encode('utf-8')
  moref = si_content['propertyCollector']
  method = 'RetrievePropertiesEx'
  url = make_url(domain, mo=moref, m_p=method)
  request_headers = {}
  request_headers['vmware-api-session-id'] = token
  request_headers['Content-Type'] = 'application/json'
  (response_headers, response_body) = send_request(url, headers=request_headers,
body=request_body)
  json_body = json.loads(response_body)
  return(json_body)


# MAIN


si_content = get_service_content(domain)
pprint(si_content); print()


token = basic_auth(domain, credentials)
print('Session token: {}'.format(token)); print()
```

```
dc_moref = get_datacenter_moref(domain, dc_name, token)

print('{} MOref = {}'.format(dc_name, dc_moref)); print()


response_body = get_vm_power_states(domain, dc_moref, token)

print('Response body:'); pprint(response_body)


moref = si_content['sessionManager']

headers = logout(domain, moref)

print('Logging out:')

print(headers)
```

Datacenter Inventory

The vSphere inventory is a representation of the vSphere datacenter and the objects in the datacenter. Knowing how the objects in the datacenter relate to each other helps you traverse the inventory hierarchy and access the objects you want to manipulate.

Inventory Overview for the Web Services SDK

The vSphere inventory contains the following types of objects:

- Systems in the datacenter: **`Host`**, **`VirtualMachine`**, and **`VirtualApp`**.
- Support components: **`ComputeResource`**, **`Datastore`**, **`Network`**, and virtual devices.
- Organizational components: **`Folder`** and **`Datacenter`**

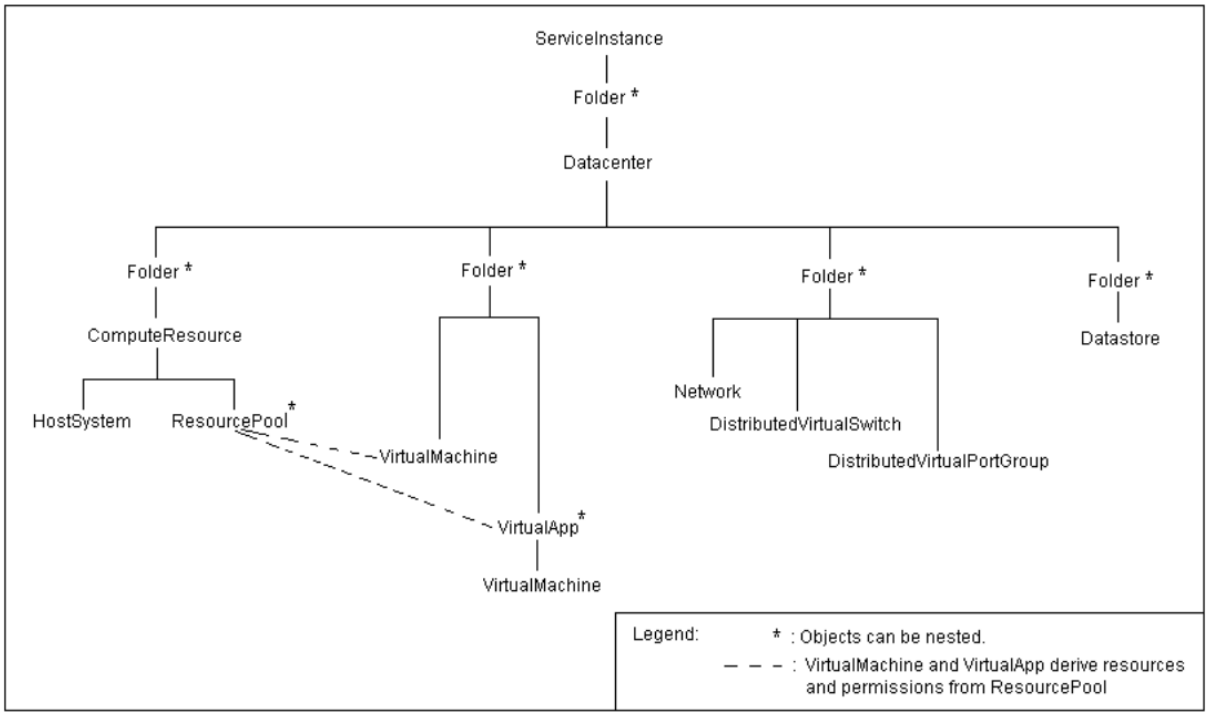
When you manage the virtual infrastructure, you access objects and their properties and methods based on their location in the inventory. Understanding the inventory structure is therefore critical for any programming task. You always start with the **`ServiceInstance`** associated with a session, which is the root object of the inventory, and traverse the inventory hierarchy from there. See Property Collector. How you access objects depends on whether your client application is connected to a vCenter Server or an ESXi host.

Inventory Hierarchies and ServiceInstance

When you start a session, vSphere creates a **`ServiceInstance`** with one root folder, one **`Datacenter`**, and four folders that hold the different types of inventory objects.

When you access a vCenter Server System, the hierarchy shown in the illustration below allows you to traverse the inventory.

Figure 11: vCenter Server Inventory Hierarchy



CAUTION

If your ESXi hosts are managed by vCenter Server, you must always access your hosts through vCenter Server. The vCenter service keeps track of all synchronous and asynchronous operations, and will have the latest status and inventory information about each ESXi host. Therefore, connecting directly to a managed host may give you incorrect or incomplete data.

When you have ESXi hosts that are not managed by vCenter Server, your application can connect to each host directly.

Folders in the Hierarchy

If your installation includes a vCenter Server system, you can create additional datacenters under the root folder. For every **`Datacenter`** object, the server automatically creates the following **`Folder`** objects:

- A folder for **`VirtualMachine`**, template, and **`VirtualApp`** objects.
- A folder for a **`ComputeResource`** hierarchy.
- A folder for **`Network`**, **`DistributedVirtualSwitch`**, and **`DistributedVirtualPortgroup`** objects.
- A folder for **`Datastore`** objects.

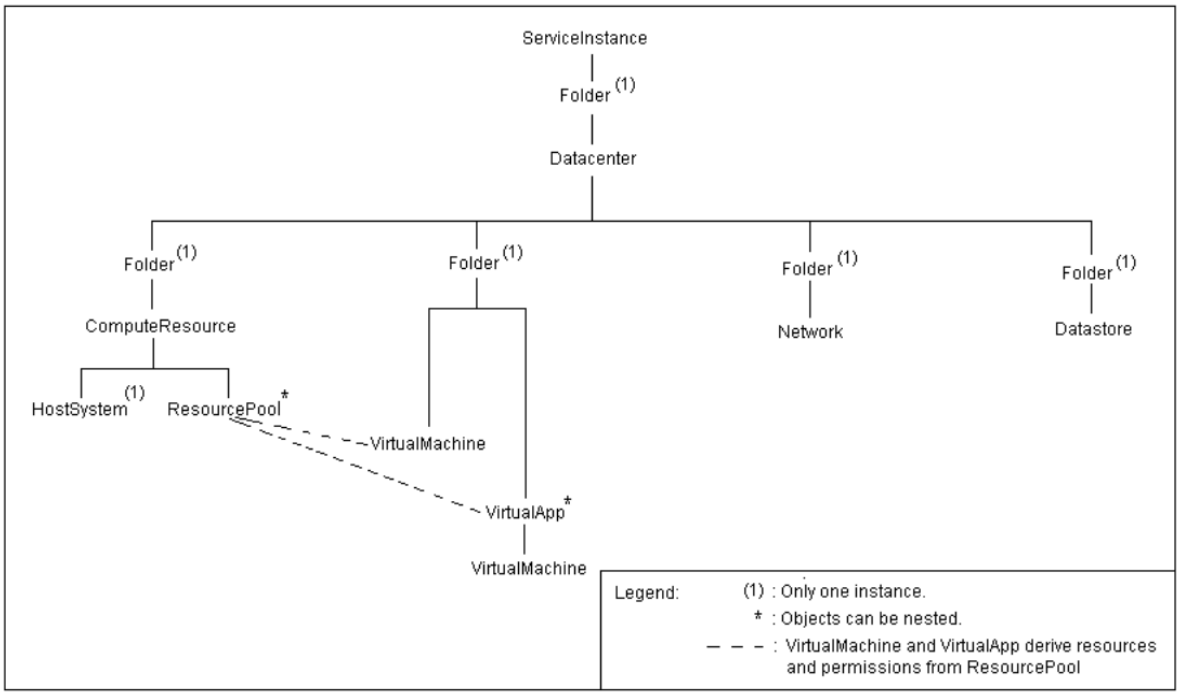
In a large deployment, the nested structure allows you to organize the objects in the datacenter into an easily manageable structure by using multiple folders and datacenters.

For a standalone ESXi system, only a single datacenter is supported, and the **`Folder`** managed entity does not support creating additional **`Folder`** objects or **`Datacenter`** objects.

ESXi Inventory Hierarchy

When you access an ESXi host directly, rather than accessing the host through a vCenter Server system, the hierarchy shown in the illustration below allows you to traverse the inventory.

Figure 12: ESXi Inventory Hierarchy



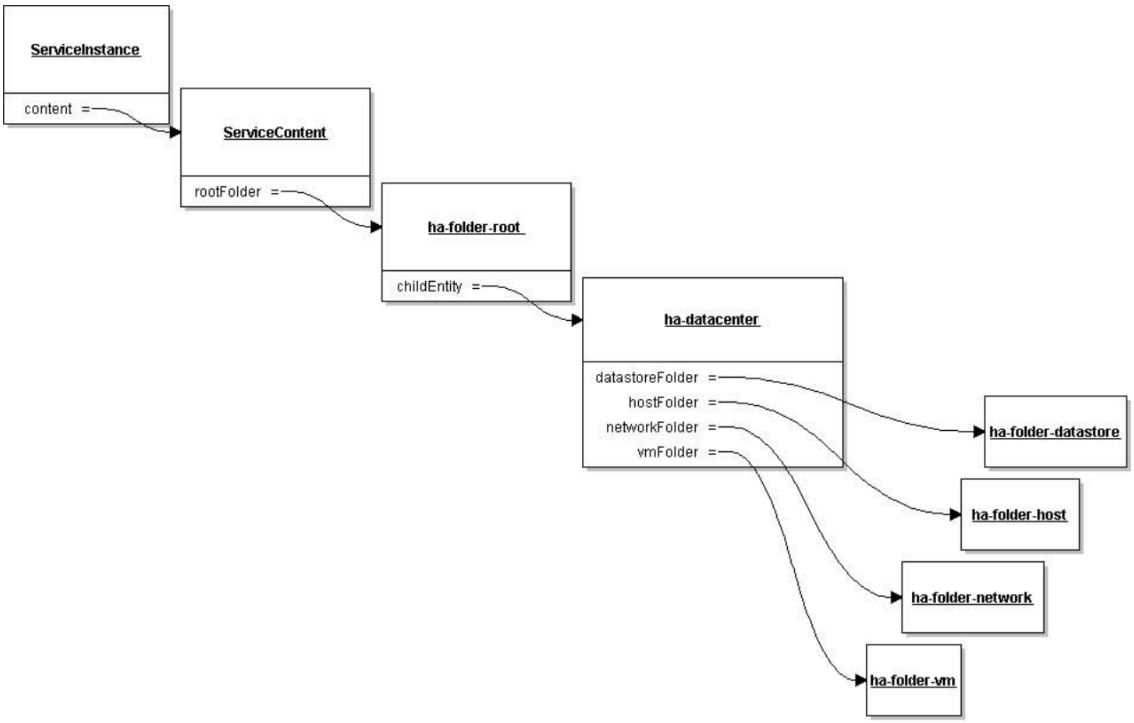
Accessing Inventory Objects

To retrieve information from an inventory object, you start with **`ServiceInstance`**, the root object of the inventory. You access an object using a **`TraversalSpec`** in conjunction with a property collector, using the properties that identify an object's position in the hierarchy.

- Every managed entity has a **`parent`** property that identifies its relative position in the inventory hierarchy.
- The **`Folder`** managed object has a **`childEntity`** property that identifies objects in a folder instance.

Figure: Instance Diagram of Root Folders in an Inventory shows the **`childEntity`** and **`folder`** properties that define the default objects in the inventory of a standalone ESXi system. The inventory begins with the **`ServiceContent.rootFolder`** property. The **`rootFolder`** has a **`childEntity`** that consists of a managed object reference to a **`Datacenter`** managed object.

Figure 13: Instance Diagram of Root Folders in an Inventory



Creating Inventory Objects

The **`Folder`** managed entity provides methods for creating instances of the following managed entities.

- **`Datacenter`**
- **`DistributedVirtualSwitch`**
- **`VirtualMachine`**
- **`Cluster`**
- **`Folder`**

When you create these objects, they appear in the folder you invoked the creation method from.

While some managed entities are created through a method on the **`Folder`** managed entity, other managed entities are instantiated directly. For example, the **`HostDatastoreSystem`** has methods for creating datastores such as **`CreateNasDatastore`** and **`CreateVmfsDatastore`**.

> **IMPORTANT**
> When you create an inventory object, you must stay within the bounds of the host's capabilities, accessible through the **`HostSystem.capability`** property, which is a **`HostCapability`** data object. For example, a **`HostCapability`** object might have the **`maxSupportedVMs`** property specified.

Privileges Required for Inventory Management

Navigating the inventory requires a user account that can connect to the server and obtain a valid session. The user identity associated with the session is called a principal. When a client application attempts to access an object in the inventory, the server checks the permission object or objects and compares the permissions with the principal's privileges.

For example, creating a virtual machine requires that the principal associated with the session have the following privileges:

- The **VirtualMachine.Inventory.Create** privilege on the folder in which to create the virtual machine.
- The **Resource.AssignVMToPool** privilege on the resource pool from which the virtual machine obtains its allocation of CPU and memory resources.

Reading the **perfCounter** property of the **PerformanceManager** managed object requires the **System.View** privilege on the root folder.

> **IMPORTANT**
> Some privileges are specific to objects on vCenter Server or specific to ESXi. For example, the **Alarm.Create** privilege associated with **AlarmManager** is available only through vCenter Server systems.

See Authentication and Authorization for more information on authentication, authorization, roles, and user identity.

Privileges

A privilege is a system-defined requirement associated with a VMware vSphere managed object. Privileges are static and do not change for a version of a product. Privileges for vSphere components are defined as follows:

```
<group>[.<group>].privilege
```

For example:

> **Datacenter.Create**
>
> **Host.Config.Connection**
>
> Host.Config.Snmp

Permissions

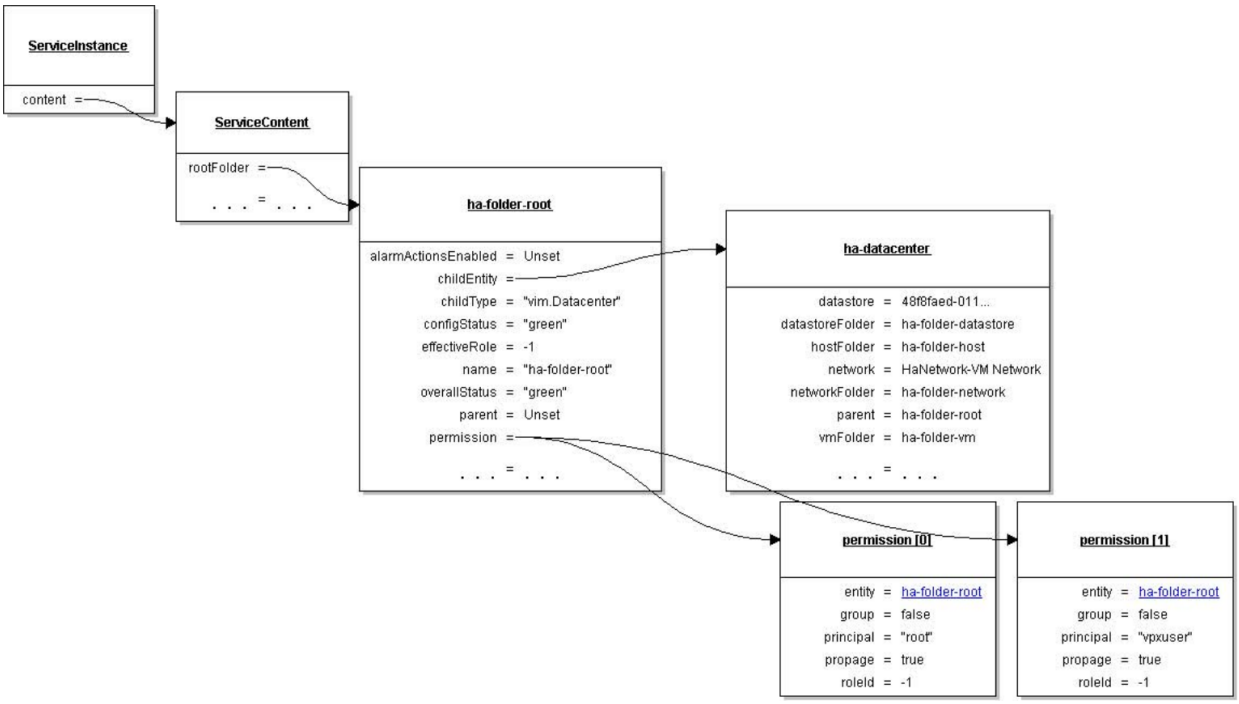
Permissions are the associations of roles with privileges on a specified managed entity. You use permissions to specify which users can access which managed entity.

A child entity inherits the permissions of its parent if the parent's **propagate** property is set to true. A permission that is set directly on a child overrides the permission in the parent. To grant permission to all child entities of a **Datacenter** object, assign permissions to the **Datacenter** object and set the **Permission** object's **propagate** property to **true**.

Figure: Inventory and Permissions shows that users **root** and **vpxuser** both have permissions on the **rootFolder** of the inventory. The **vpxuser** is the account created on a host by the vCenter Server system when that host is added to the vCenter Server system. The vCenter Server system needs access to the inventory objects of the host systems that it manages, so the **vpxuser** account is granted privileges to the **rootFolder** of each host.

> **IMPORTANT**
> See Authentication and Authorization for a detailed discussion of privileges, permissions, and user management.

Figure 14: Inventory and Permissions



Managed and Standalone ESXi Hosts

You can run ESXi as a managed or standalone ESXi host.

- Standalone ESXi hosts are standalone hosts with limited capabilities. The inventory of a standalone host can support multiple virtual machines and multiple resource pools, but it contains a single default datacenter and a single root folder. The default datacenter and root folder are not visible in the vSphere Client, but they exist in the inventory of a standalone host and they are visible in the MOB.
- Managed ESXi hosts have been added to the vCenter Server inventory. Available features depend on the licenses available for that host. For example, you can configure two or more hosts for VMware DRS resource management or VMware HA failover protection.

The following table summarizes the differences between the number of objects that an inventory can contain. See also Figure: ESXi Inventory Hierarchy and Figure: vCenter Server Inventory Hierarchy.

ManagedEntity Subtype	ESX/ESXi Inventory	vCenter Server Inventory
`ClusterComputeResource`	None.	Multiple instances supported.
`ComputeResource`	Exactly one only.	Multiple instances supported.
`Datacenter`	Exactly one only. Cannot be destroyed. Transparent.	Multiple instances supported.
`Datastore`	Multiple instances supported.	Multiple instances supported.
`DistributedVirtualSwitch`	Multiple instances supported.	Multiple instances supported.
`Folder`	Exactly one only. Cannot be destroyed. Transparent.	Multiple instances supported.
`HostSystem`	Exactly one only.	Multiple instances supported.

Table continued on next page

Continued from previous page

ManagedEntity Subtype	ESX/ESXi Inventory	vCenter Server Inventory
`Network`	Multiple instances supported.	Multiple instances supported.
`ResourcePool`	Multiple instances supported.	Multiple instances supported.
`VirtualApp`	None.	Multiple instances supported.
`VirtualMachine`	Multiple instances supported.	Multiple instances supported.

Property Collector

vSphere provides the **`PropertyCollector`** service for accessing data and monitoring changes. Use the **`RetrievePropertiesEx`** to obtain values of managed object properties, and **`WaitForUpdatesEx`** to monitor and retrieve modified property values.

Introduction to the PropertyCollector

The **`PropertyCollector`** service provides a way to monitor and retrieve information about managed objects, such as whether a host in a cluster is offline, or whether a virtual machine is powered on.

The **`PropertyCollector`** uses one or more filters to determine the scope of collection and it has methods to retrieve data. A filter uses a set of data objects that specify the following information:

- Starting point for inventory traversal during the collection operation.
- Inventory traversal path.
- Objects and properties from which data will be collected.

A vSphere server creates a default **`PropertyCollector`** for every session, and allows you to create multiple, additional **`PropertyCollector`** objects. Create additional **`PropertyCollector`** objects, using one per thread, to perform concurrent collection operations.

Data Retrieval

`PropertyCollector` can retrieve data all at once, or incrementally.

- Property retrieval as a single operation uses the **`RetrievePropertiesEx`** and **`ContinueRetrievePropertiesEx`** methods. These methods perform a single collection operation.
- Incremental property retrieval, also referred to as property monitoring, uses the **`WaitForUpdatesEx`** method. The initial call to this method retrieves a baseline set of managed object property values. Subsequent calls retrieve changes in property values since the last retrieval. Use **`WaitForUpdatesEx`** to monitor changes to the inventory or any managed object properties.

  NOTE
  The **`PropertyCollector`** does not guarantee the order of data that it returns in response to a request for data.

Inventory Traversal and Property Selection

To use **`PropertyCollector`**, first build a `PropertyFilterSpec` to guide the collector's actions on the server.

You pass your `PropertyFilterSpec` to either the `RetrievePropertiesEx` method or the `CreateFilter` method. The latter establishes a persistent filter on the server side for use with the `WaitForUpdatesEx` method.

A `PropertyFilterSpec` contains:

- The name of at least one managed object class from which you want to collect properties. (`PropertySpec.type`)
- For each managed object class specified, at least one name of a property to return. (`PropertySpec.pathSet`).
- If needed, one or more paths through the inventory hierarchy that the PropertyCollector can traverse to reach the managed objects from which you want to collect properties. (`ObjectSpec.selectSet`)
- For each path, a starting managed object reference from which to traverse that path. (`ObjectSpec.obj`)

If you do not have a starting managed object reference from a prior API call, you can start with the root folder at the base of the inventory hierarchy. The `rootFolder` reference is available in the `ServiceContent` data object. For information about getting the `ServiceContent`, see Overview of a Java Sample Application for the Web Services SDK.

Gathering Data with a ViewManager Object

You can use a vSphere view object in a `PropertyCollector` filter to simplify access to a subset of inventory objects. For example, **`ContainerView`** collects information about all VirtualMachine objects in a datacenter, and `ListView` tracks information about **`Task`** objects while an operation is in progress.

The `ViewManager`'s **`ListView`** method allows you to customize your view with an input object list, the **`ContainerView`** method lets you view all objects in a folder, datacenter, resource pool, or other container of managed objects, and the **`InventoryView`** method lets you monitor the entire inventory.

> **NOTE**
> A best practice is to use the smallest view you can create. This will be the most efficient way to retrieve data with the PropertyCollector.

The **`ViewManager`** has the following property:

- **`viewList`** – An array of view references. Each array entry is a managed object reference to a view created by this View Manager.

The view acts as a virtual parent object to the set, which provides the following benefits:

- The view can collect objects from different parts of the inventory and make them available under the same parent.
- All objects in the set are accessible by doing a single traversal step, rather than a chain of several steps starting at the root folder.

A best practice when using views is to call the `DestroyView()` method when a view is no longer needed. This practice frees memory on the server.

See Retrieve Properties from a Set of Managed Objects with a View for an example that uses the **`ContainerView`** method to access Inventory data.

For more information about the vSphere inventory, browse the managed object definitions in the *`vSphere Web Services API Reference`*.

vSphere Data Objects for Property Collection

`PropertyCollector` data objects are used to specify objects and properties for the collector to retrieve.

Data Object	Description
`PropertyFilterSpec`	Provides access to object and property selection data. A **`PropertyFilterSpec`** must have at least one **`ObjectSpec`** and at least one **`PropertySpec`**.
`ObjectSpec`	Identifies the starting object for property collection, which can be either an inventory object or a view you created. An **`ObjectSpec`** also identifies the traversal specs used by the property collector.

Table continued on next page

Continued from previous page

Data Object	Description
PropertySpec	Identifies object types and their properties for collection.
TraversalSpec	Identifies the objects and paths that can be traversed to find the objects and properties identified by the **PropertySpec**.
SelectionSpec	Lists the **TraversalSpec** objects available at each step of an inventory traversal. Supports recursive traversal by name.
View objects	Use a `view` to form a set of starting objects for traversal and collection.

vSphere Methods for Property Collection

The **PropertyCollector** supports the following approaches to obtaining objects and properties from the server:

- If your client application does not keep a synchronized representation of server state, use the **RetrievePropertiesEx** method. **RetrievePropertiesEx** instantiates a filter, collects the specified objects and properties, and returns the data to your client application as an **ObjectContent** data object. The server does not add the filter to the **PropertyCollector.filter** array. The server destroys the filter after returning the results to your client.
- If your application maintains a synchronized representation of server state, use the **CreateFilter** and **WaitForUpdatesEx** methods. **WaitForUpdatesEx** returns descriptions of property changes, organized by the filter that identified the properties.

In either case, you create a **PropertyFilterSpec** data object to specify the objects and properties you want to retrieve from the server.

The following table shows the **PropertyCollector** methods organized by the context in which you use them. For more information about these methods, see the *vSphere API Reference*.

Method Context	Method	Description
Monitor properties using different filters	**CreatePropertyCollector**	Creates a new **PropertyCollector** object to monitor properties using different filters. The vSphere server handles requests for a **PropertyCollector** instance independently of any other instances of the **PropertyCollector** on the server.
	DestroyPropertyCollector	Destroys an instance of a **PropertyCollector** that was created by a call to **CreatePropertyCollector** from your client application.
Single collection operation	**RetrievePropertiesEx**	Retrieves property data for the managed objects specified by a single-use **PropertyFilterSpec**.
	ContinueRetrievePropertiesEx	Retrieves additional property data for an operation started by **RetrievePropertiesEx**.
	CancelRetrievePropertiesEx	Cancels a **RetrievePropertiesEx** or **ContinueRetrievePropertiesEx** operation.
Incremental collection or monitoring operation	**WaitForUpdatesEx**	Retrieves changes to property data since the last **WaitForUpdatesEx** cycle. **WaitForUpdatesEx** blocks until one or more changes are detected or until the request times out.
	CancelWaitForUpdatesEx	Cancels a **WaitForUpdatesEx** operation.

Table continued on next page

Continued from previous page

Method Context	Method	Description
General	**`CreateFilter`**	Creates a new instance of a **`PropertyFilter`** managed object.

Using the PropertyCollector with RetrievePropertiesEx

Two use cases show how to retrieve properties of selected managed objects with the `RetrievePropertiesEx` method.

A simple use case is to retrieve properties from a managed object for which a reference has been supplied. You need only create a **`PropertyFilterSpec`** that contains a **`PropertySpec`** to identify the property to retrieve and an **`ObjectSpec`** to identify the managed object. Pass this **`PropertyFilterSpec`** to the `RetrievePropertiesEx` method and get the property value from the structured result.

A more complicated use case retrieves properties from a set of managed objects found in the inventory hierarchy. For this use case, you create a **`View`** managed object that provides a path to all the target managed objects within one traversal step. You need to create a **`PropertyFilterSpec`** that contains a **`PropertySpec`** to identify the property to retrieve and an **`ObjectSpec`** that identifies the **`View`** object and also contains a **`TraversalSpec`** to describe the path from the **`View`** to its children. Pass this **`PropertyFilterSpec`** to the `RetrievePropertiesEx` method and get the property value from the structured result.

Retrieve a Property from a Specified Managed Object

This procedure uses **`PropertyCollector`** to retrieve a property from a single **`VirtualMachine`** object with the `RetrievePropertiesEx` method, by obtaining a managed object reference to the parent object of the **`VirtualMachine`** in the inventory.

For this task you need:
- A virtual machine managed object reference in a variable named **`vmRef`**.
- An authenticated Web Services session with the vSphere server that manages the virtual machine.
- A **`VimPort`** binding provider referenced by the variable **`methods`**, which is attached to the vSphere server connection context.
- A **`PropertyCollector`** instance referenced by the variable **`pCollector`**.

> **NOTE**
> This procedure shows only how to use the **`PropertyCollector`**. For a description of server connection and getting a reference to the **`PropertyCollector`**, see Build a Simple vSphere Client Application for the Web Services SDK.

To retrieve a reference to the parent object of a specific virtual machine, you must:
- Prepare a **`PropertySpec`** to specify the **`parent`** property and the **`parentVApp`** property of the **`VirtualMachine`** class. A virtual machine can have only one parent, but the parent can be either a **`Folder`** object or a **`VirtualApp`** object. This example collects both properties, one of which will be null.
- Prepare an **`ObjectSpec`** to identify the starting object, which is the specified **`VirtualMachine`**. There is no need for a **`TraversalSpec`** because the property belongs to the starting object itself. However, the **`ObjectSpec.skip`** property is set to **`false`**; a value of **`true`** would cause the **`PropertyCollector`** not to collect properties from the starting object.
- Assemble a **`PropertyFilterSpec`** from the prepared data.
- Invoke the `RetrievePropertiesEx` method.

1. Declare a function that accepts a virtual machine **`MOref`** and returns a **`MOref`** to its parent **`object.in`** a key-value pair.

```
private static DynamicProperty getVMParentProperty(ManagedObjectReference vmRef)
throws Exception {
```

2. Specify the properties for retrieval (**`VirtualMachine.parent`** and **`VirtualMachine.parentVApp`**).

   Because the **`VirtualMachine`** has two mutually exclusive parent properties, depending on the parent type, this code collects both properties.

```
PropertySpec pSpec = new PropertySpec();
pSpec.setType("VirtualMachine");
pSpec.getPathSet().add("parent");
pSpec.getPathSet().add("parentVApp");
```

3. Create an **`ObjectSpec`** to define the property collection.

   Use the `setObj` method to specify that the **`vmRef`** is the starting object for this property collection. Set the **`skip`** property to **`false`** to indicate that you want to collect properties from the starting object. Omit the **`selectSet`** property because the property collection does not need to traverse away from the starting object.

```
ObjectSpec oSpec = new ObjectSpec();
oSpec.setObj(vmRef);
oSpec.setSkip(false);
```

4. Create a **`PropertyFilterSpec`** and add the object and property specs to it.

```
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);
```

5. Create a list for the filters and add the property filter spec to it.

```
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);
```

6. Use the filter spec to retrieve the property collection from the server.

```
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(pCollector,fSpecList,ro);
```

7. Unwrap the parent property.

```
if (props != null) {
  for (ObjectContent oc : props.getObjects()) {
    List<DynamicProperty> dps = oc.getPropSet();
    if (dps != null) {
      for (DynamicProperty dp : dps) {
        if (dp.getName().equals("parent") || dp.getName().equals("parentVApp")) {
          return dp;
        }
      }
```

```
        }
      }
    }
    System.out.println("Parent not found.");
    throw new Exception();
  }
```

The method returns a key-value pair, **name** and **val**, indicating the name of the parent property and a managed object reference to the parent object.

```
DynamicProperty parent = getVMParentProperty( vmRef );
if (parent.getName().equals("parentVApp")) {
  System.out.format("VApp MOref: %s", parent.getVal());
} else {
  System.out.format("Folder MOref: %s", parent.getVal());
}
```

Retrieve Properties from a Set of Managed Objects with a View

This procedure uses **PropertyCollector** with a **ContainerView** to retrieve properties from multiple **VirtualMachine** objects with the `RetrieveProperiesEx` method.

To collect the names of all virtual machines in the inventory, you must:

- Prepare a **ContainerView** that contains only the virtual machines.
- Prepare a **PropertySpec** to specify the **name** property of the **VirtualMachine** class.
- Prepare a **TraversalSpec** to specify the path from the view to its contents.
- Assemble a **PropertyFilterSpec** from the prepared data.
- Invoke the **RetrievePropertiesEx** method.

> **NOTE**
> This procedure shows only how to use the **PropertyCollector**. For a description of server connection, see Build a Simple vSphere Client Application for the Web Services SDK.

To collect the names of all virtual machines in the inventory, use the following steps.

1. Get references to the **ViewManager** and the **PropertyCollector**.

   In the example, **sContent** is the variable for the **ServiceContent** data object. **sContent** provides the methods to retrieve the managed object references to the vSphere services.

   ```
   ManagedObjectReference viewMgrRef = sContent.getViewManager();
   ManagedObjectReference propColl = sContent.getPropertyCollector();
   ```

2. Create a container view for virtual machines.

methods is the variable for the **VimPortType** object. **VimPortType** defines the Java methods that correspond to the vSphere API methods. The **createContainerView** parameters are as follows:

- **container**: Selection starts within the inventory root folder.
- **type**: Only **VirtualMachine** type managed objects are selected for the **ContainerView**, starting at the root folder.
- **recursive**: The value **true** for the last parameter extends the selection beyond the children of the root folder so that the **ViewManager** will follow child folder paths to add virtual machines to the view.

This container view provides references to all virtual machines in the inventory.

```
List<String> vmList = new ArrayList<String>();

vmList.add("VirtualMachine");


ManagedObjectReference cViewRef = methods.createContainerView(viewMgrRef,

    sContent.getRootFolder(),

    vmList,

    true );
```

3. Create an object specification to define the starting point for inventory navigation.

   The `ObjectSpec.obj` property identifies the starting object (the container view). This example collects only virtual machine data, so the **skip** property is set to **true** to ignore the container view itself during collection.

   ```
   ObjectSpec oSpec = new ObjectSpec();

   oSpec.setObj(cViewRef);

   oSpec.setSkip(true);
   ```

4. Create a traversal specification to identify the path for collection.

   The **TraversalSpec** properties **type** and **path** determine path traversal. **TraversalSpec.type** identifies the type of managed object that this spec can traverse. **TraversalSpec.path** identifies a property that links to other managed objects. The **PropertyCollector** uses the **path** object to select additional objects.

   This example uses a single **TraversalSpec** to walk the list of virtual machines that are available through the container view. The following code fragment specifies the **ContainerView** object for the **TraversalSpec.type** property and the **view** property in the **ContainerView** for the **TraversalSpec.path** property. The **skip** property is set to **false**, so the **PropertyCollector** will collect data from the **path** objects (the virtual machines in the container view).

   ```
   TraversalSpec tSpec = new TraversalSpec();

   tSpec.setName("traverseEntities");

   tSpec.setPath("view");

   tSpec.setSkip(false);

   tSpec.setType("ContainerView");
   ```

5. Add the **TraversalSpec** to the **ObjectSpec.selectSet** array. The **TraversalSpec** tells the PropertyCollector how to traverse from the starting object to other managed objects.

```
oSpec.getSelectSet().add(tSpec);
```

6. Identify a managed object type and the properties to be retrieved from it.

   The example program creates a **PropertySpec** data object to specify the properties to be collected. The **type** property is set to a managed object type present in the container view. The **pathSet** property identifies one or more properties in the **type** object.

   This example specifies the `VirtualMachine.name` property.

   ```
   PropertySpec pSpec = new PropertySpec();
   pSpec.setType("VirtualMachine");
   pSpec.getPathSet().add("name");
   ```

7. Add the object and property specifications to the property filter specification.

   A **PropertyFilterSpec** must have at least one **ObjectSpec** and one **PropertySpec**. The **PropertyFilterSpec** specifies what properties to collect, while the **ObjectSpec** tells the PropertyCollector how to find them.

   ```
   PropertyFilterSpec fSpec = new PropertyFilterSpec();
   fSpec.getObjectSet().add(oSpec);
   fSpec.getPropSet().add(pSpec);
   ```

8. Create a list for the filters and add the **PropertyFilterSpec** to it.

   This example needs only one filter to collect the names of virtual machines.

   ```
   List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
   fSpecList.add(fSpec);
   ```

9. Retrieve the data.

   To invoke a single property collection operation, call the **RetrievePropertiesEx** method. The example application passes the populated **PropertyFilterSpec** and an empty **options** structure to the method. The default for the **RetrieveOptions.maxObjects** specifies no maximum for the number of objects that can be returned.

   > **NOTE**
   > The **PropertyCollector** can impose a maximum in some circumstances. If the number of collected objects is greater than the maximum, the **PropertyCollector** returns a **token** value in the **RetrieveResult** data object and this token is used to retrieve the remaining properties using the **ContinueRetrievePropertiesEx** API method. For more information, see Server Data Transmission.

   ```
   RetrieveOptions ro = new RetrieveOptions();
   RetrieveResult props = methods.retrievePropertiesEx(propColl,fSpecList,ro);
   ```

10. Print the virtual machine names.

    The following code fragment walks the list of **ObjectContent** objects returned in the **RetrieveResult** object. For each object (**ObjectContent**), the inner loop prints the name-value pairs.

```
if (props != null) {
   for (ObjectContent oc : props.getObjects()) {
   String vmName = null;
   String path = null;
   List<DynamicProperty> dps = oc.getPropSet();
   if (dps != null) {
      for (DynamicProperty dp : dps) {
         vmName = (String) dp.getVal();
         path = dp.getName();
         System.out.println(path + " = " + vmName);
      }
   }
}
```

You can also see these code fragments in the context of an end-to-end example that includes connection and authentication logic, in Build a Simple vSphere Client Application for the Web Services SDK.

Property Collector Program in Java for Web Services SDK

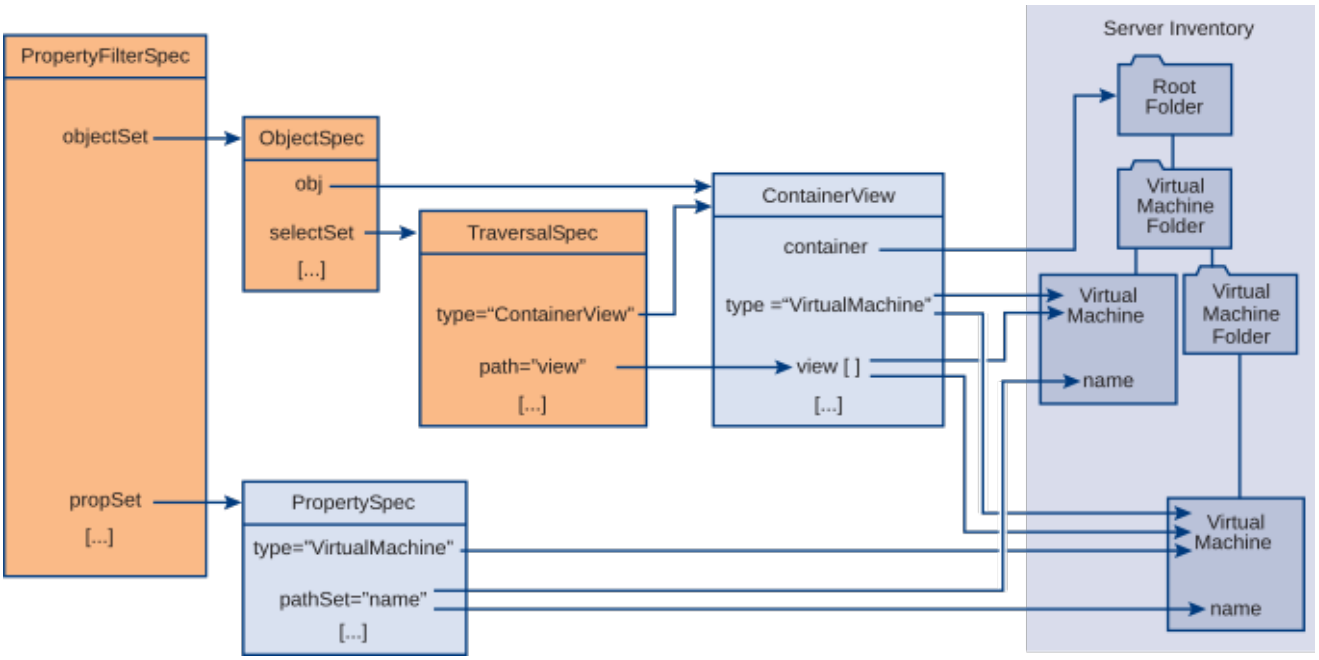
`PropertyCollector` provides a powerful, flexible, and efficient way to collect data from a vCenter Server or ESXi host.

Simple PropertyCollector Example (Java) is a simple **`PropertyCollector`** example written in Java. The example uses a **`ContainerView`** for efficient access to the inventory and a **`PropertyFilterSpec`** that contains one **`ObjectSpec`**, one **`TraversalSpec`**, and one **`PropertySpec`**. The program performs the following tasks:

1. Accepts command line arguments for the vSphere server name (DNS name or IP address), user name, and password.
2. Connects to a vSphere server.
3. Uses a **`ContainerView`** to create a subset of the inventory; the subset contains only virtual machines.
4. Uses the **`RetrievePropertiesEx`** method for a single retrieval operation.
5. Collects the names of all of the virtual machines in the inventory and prints the names using the standard output stream.
6. Closes the connection to the server.

Figure: Property Filter Specification shows the objects used in Simple PropertyCollector Example (Java). The figure represents properties that identify inventory elements directly or indirectly. It does not show all the properties for the different objects.

Figure 15: Property Filter Specification



Simple PropertyCollector Example (Java)

```
import com.vmware.vim25.*;


import java.util.*;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.SSLSession;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.soap.SOAPFaultException;



// PropertyCollector example
// command line input: server name, user name, password


public class PCollector {


  private static void collectProperties(VimPortType methods,
                                        ServiceContent sContent) throws Exception {
```

```
   // Get references to the ViewManager and PropertyCollector
   ManagedObjectReference viewMgrRef = sContent.getViewManager();
   ManagedObjectReference propColl = sContent.getPropertyCollector();

   // use a container view for virtual machines to define the traversal
   // - invoke the VimPortType method createContainerView (corresponds
   //   to the ViewManager method) - pass the ViewManager MOR and
   //   the other parameters required for the method invocation
   // - createContainerView takes a string[] for the type parameter;
   //   declare an arraylist and add the type string to it
   List<String> vmList = new ArrayList<String>();
   vmList.add("VirtualMachine");

   ManagedObjectReference cViewRef = methods.createContainerView(viewMgrRef,
sContent.getRootFolder(),
                                                                 vmList,
                                                                 true);

   // create an object spec to define the beginning of the traversal;
   // container view is the root object for this traversal
   ObjectSpec oSpec = new ObjectSpec();
   oSpec.setObj(cViewRef);
   oSpec.setSkip(true);

   // create a traversal spec to select all objects in the view
   TraversalSpec tSpec = new TraversalSpec();
   tSpec.setName("traverseEntities");
   tSpec.setPath("view");
   tSpec.setSkip(false);
   tSpec.setType("ContainerView");

   // add the traversal spec to the object spec;
```

```
// the accessor method (getSelectSet) returns a reference
// to the mapped XML representation of the list; using this
// reference to add the spec will update the list
oSpec.getSelectSet().add(tSpec);

// specify the property for retrieval (virtual machine name)
PropertySpec pSpec = new PropertySpec();
pSpec.setType("VirtualMachine");
pSpec.getPathSet().add("name");

// create a PropertyFilterSpec and add the object and
// property specs to it; use the getter method to reference
// the mapped XML representation of the lists and add the specs
// directly to the list
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);

// Create a list for the filters and add the spec to it
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);

// get the data from the server
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(propColl,fSpecList,ro);

// go through the returned list and print out the data
if (props != null) {
  for (ObjectContent oc : props.getObjects()) {
    String vmName = null;
    String path = null;
    List<DynamicProperty> dps = oc.getPropSet();
```

```
      if (dps != null) {
        for (DynamicProperty dp : dps) {
          vmName = (String) dp.getVal();
          path = dp.getName();
          System.out.println(path + " = " + vmName);
        }
      }
    }
  }
}//end collectProperties()

// Authentication is handled by using a TrustManager and supplying
// a host name verifier method. (The host name verifier is declared
// in the main function.)
//
// For the purposes of this example, this TrustManager implementation
// will accept all certificates. This is only appropriate for
// a development environment. Production code should implement certificate support.

private static class TrustAllTrustManager
        implements javax.net.ssl.TrustManager,
                   javax.net.ssl.X509TrustManager {
  public java.security.cert.X509Certificate[] getAcceptedIssuers() {
    return null;
  }

  public boolean isServerTrusted(java.security.cert.X509Certificate[] certs) {
    return true;
  }

  public boolean isClientTrusted(java.security.cert.X509Certificate[] certs) {
    return true;
```

```
    }

    public void checkServerTrusted(java.security.cert.X509Certificate[] certs,
                                   String authType)
            throws java.security.cert.CertificateException {
      return;
    }

    public void checkClientTrusted(java.security.cert.X509Certificate[] certs,
                                   String authType)
            throws java.security.cert.CertificateException {
      return;
    }
  }

  public static void main(String [] args) throws Exception {

    // arglist variables
    String serverName = args[0];
    String userName = args[1];
    String password = args[2];
    String url = "https://"+serverName+"/sdk/vimService";
    // Variables of the following types  for access to the API methods
    // and to the vSphere inventory.
    // -- ManagedObjectReference for the ServiceInstance on the Server
    // -- VimService for access to the vSphere Web service
    // -- VimPortType for access to methods
    // -- ServiceContent for access to managed object services
    ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
    VimService vimService;
    VimPortType vimPort;
    ServiceContent serviceContent;
```

```
  // Declare a host name verifier that will automatically enable
  // the connection. The host name verifier is invoked during
  // the SSL handshake.
  HostnameVerifier hv = new HostnameVerifier() {
  public boolean verify(String urlHostName, SSLSession session) {
    return true;
  }
};

// Create the trust manager.
javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
trustAllCerts[0] = tm;
// Create the SSL context
javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL");

// Create the session context
javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();

// Initialize the contexts; the session context takes the trust manager.
sslsc.setSessionTimeout(0);
sc.init(null, trustAllCerts, null);

// Use the default socket factory to create the socket for the secure connection
javax.net.ssl.HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());

// Set the default host name verifier to enable the connection.
HttpsURLConnection.setDefaultHostnameVerifier(hv);

// Set up the manufactured managed object reference for the ServiceInstance
SVC_INST_REF.setType("ServiceInstance");
SVC_INST_REF.setValue("ServiceInstance");
```

```
  // Create a VimService object to obtain a VimPort binding provider.
  // The BindingProvider provides access to the protocol fields
  // in request/response messages. Retrieve the request context
  // which will be used for processing message requests.
  vimService = new VimService();
  vimPort = vimService.getVimPort();
  Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();

  // Store the Server URL in the request context and specify true
  // to maintain the connection between the client and server.
  // The client API will include the Server's HTTP cookie in its
  // requests to maintain the session. If you do not set this to true,
  // the Server will start a new session with each request.
  ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);
  ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

  // Retrieve the ServiceContent object and login
  serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
  vimPort.login(serviceContent.getSessionManager(),
                userName,
                password,
                null);

  // retrieve data
  collectProperties( vimPort, serviceContent );

  // close the connection
  vimPort.logout(serviceContent.getSessionManager());

  }
}
```

Inventory Traversal

PropertyCollector can traverse the server inventory in arbitrary ways, enabling you to follow links between related objects.

Simple PropertyCollector Example (Java) uses a **ContainerView** to specify the objects that start the collection process. This is the simplest way to set up a filter, using a single reference to a view to provide the **PropertyCollector** with access to a set of objects. To locate objects in the inventory, a filter includes **TraversalSpec** and possibly **SelectionSpec** objects. Use these objects to direct the **PropertyCollector** to the managed objects and their properties named in the **PropertySpec** data objects of the **PropertyFilterSpec**.

TraversalSpec Traversal

Use a **TraversalSpec** object to identify a managed object type and a traversal property in that type. **TraversalSpec** contains type, path, and selection properties.

- **type** – identifies an inventory object type.
- **path** – specifies a managed object reference property in the **type** object. This property provides the traversal path extending from this object.
- **selectSet** – specifies an optional list of selection objects for additional object traversal paths. The **PropertyCollector** tries to apply the **TraversalSpec** objects in the **selectSet** array to the results of the traversal (the targets of `TraversalSpec.path`). If the type of a referenced managed object matches one of the specs in the **SelectSet** array, the **PropertyCollector** follows that **TraversalSpec** next. The **selectSet** array can also contain **SelectionSpec** objects; a **SelectionSpec** is a reference to a named **TraversalSpec**. See SelectionSpec Traversal.
- **skip** – indicates whether to collect properties for managed objects referenced by the **path** property.

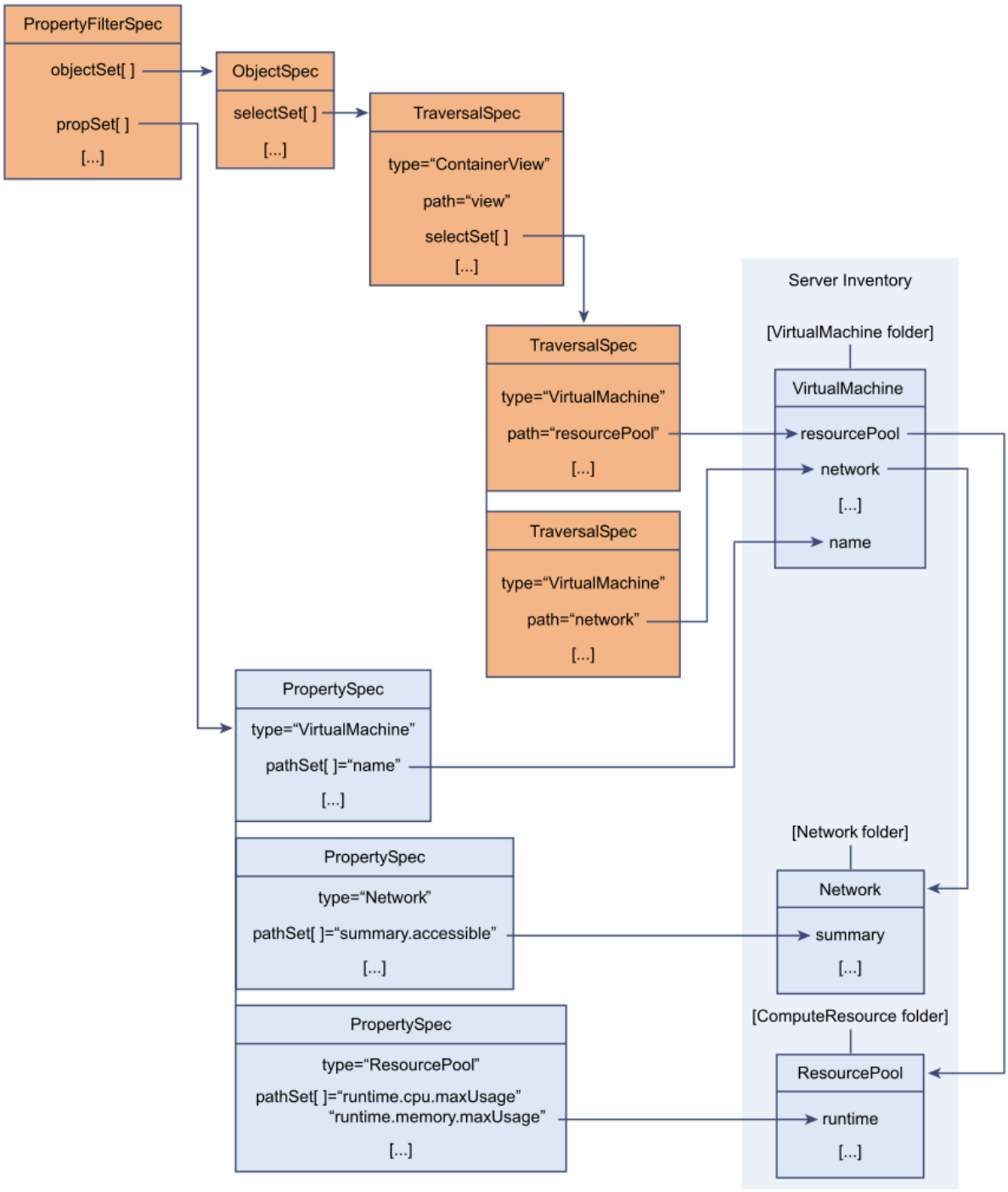
During inventory traversal, the `PropertyCollector` visits managed objects referenced by the `TraversalSpec.path` property values. When the `PropertyCollector` visits an object, it collects properties if the object's type matches a `PropertySpec.type` value.

Traversal starts with the managed object referenced by `ObjectSpec.obj`, and the `PropertyCollector` also collects its properties if it matches a `PropertySpec` and if `ObjectSpec.skip` is `false`. Then the `PropertyCollector` follows the `path` property of each applicable `TraversalSpec` in the `ObjectSpec.selectSet` array. If the `TraversalSpec.skip` property is `false`, the `PropertyCollector` can also collect properties from the objects referenced by the `pathSet` property array.

Finally, the `PropertyCollector` checks the `TraversalSpec.selectSet` array for any traversal specs that apply to the referenced objects, and the traversal continues until no more traversal specs apply.

Figure: Inventory Navigation is a representation of a **PropertyFilterSpec** that defines traversal of **VirtualMachine** objects to access related `Network` objects.. The filter uses a **ContainerView** as a starting point. The **TraversalSpec** for the **ContainerView** specifies the **view** property for access to the view's virtual machines. The figure shows **TraversalSpec** objects that extend navigation from a **VirtualMachine** object to the associated **Network** and **ResourcePool** objects. The **PropertyCollector** applies these **TraversalSpec** objects to each of the **VirtualMachine** objects in the view list. The figure also shows the **PropertySpec** objects for collecting data from **VirtualMachine**, **Network**, and **ResourcePool** objects.

Figure 16: Inventory Navigation



Traverse the Inventory By Using the Property Collector

To traverse inventory objects, you define one or more `TraversalSpec` objects. In each `TraversalSpec` object, you specify a property of one managed object that holds a reference to another managed object. The `TraversalSpec` defines a path that the PropertyCollector can follow between objects.

To define inventory traversal, use the following steps.

1. Create a **ContainerView** for virtual machines.
2. Create an **ObjectSpec** that uses the container view as the collection starting point.
3. Create a **TraversalSpec** to be applied to the **ContainerView** to select **VirtualMachine** objects.
4. Create additional **TraversalSpec** objects to select additional objects.

   The **SelectSet** list for the container view **TraversalSpec** has two **TraversalSpec** objects. Both specify a **VirtualMachine** object context. One object uses the **network** property to extend traversal to the **Network** managed object. The other uses the **resourcePool** property to extend traversal to the **ResourcePool** managed object.

5. Create **PropertySpec** objects to retrieve **VirtualMachine**, **Network**, and **ResourcePool** properties.

   To retrieve properties that are embedded in data objects, the **PropertySpec.PathSet** property uses dot notation to specify the property paths.

Inventory Traversal Example in Java

This example shows a complete Java program that collects and prints selected properties from the inventory, using traversal specs with a property collector. The example lists networks, resource pool status, and virtual machines.

Inventory Traversal in Java

```
import com.vmware.vim25.*;


import java.util.*;

import javax.net.ssl.HostnameVerifier;

import javax.net.ssl.HttpsURLConnection;

import javax.net.ssl.SSLSession;

import javax.xml.ws.BindingProvider;

import javax.xml.ws.soap.SOAPFaultException;



// PropertyCollector example

// command line input: server name, user name, password


public class PCollector_traversal {


  private static void collectProperties(VimPortType methods,

                                        ServiceContent sContent) throws Exception {


    // Get references to the ViewManager and PropertyCollector
```

```
ManagedObjectReference viewMgrRef = sContent.getViewManager();
ManagedObjectReference propColl = sContent.getPropertyCollector();

// use a container view for virtual machines to define the traversal
// - invoke the VimPortType method createContainerView (corresponds
//   to the ViewManager method) - pass the ViewManager MOR and
//   the other parameters required for the method invocation
//   (use a List<String> for the type parameter's string[])
List<String> vmList = new ArrayList<String>();
vmList.add("VirtualMachine");

ManagedObjectReference cViewRef =
      methods.createContainerView(viewMgrRef,
                                  sContent.getRootFolder(),
                                  vmList,
                                  true);

// create an object spec to define the beginning of the traversal;
// container view is the root object for this traversal
ObjectSpec oSpec = new ObjectSpec();
oSpec.setObj(cViewRef);
oSpec.setSkip(true);

// create a traversal spec to select all objects in the view
TraversalSpec tSpec = new TraversalSpec();
tSpec.setName("traverseEntities");
tSpec.setPath("view");
tSpec.setSkip(false);
tSpec.setType("ContainerView");

// add the traversal spec to the object spec;
// the accessor method (getSelectSet) returns a reference
```

```
// to the mapped XML representation of the list; using this
// reference to add the spec will update the selectSet list
oSpec.getSelectSet().add(tSpec);

// extend from virtual machine to network
TraversalSpec tSpecVmN = new TraversalSpec();
tSpecVmN.setType("VirtualMachine");
tSpecVmN.setPath("network");
tSpecVmN.setSkip(false);

// extend from virtual machine to resourcepool
TraversalSpec tSpecVmRp = new TraversalSpec();
tSpecVmRp.setType("VirtualMachine");
tSpecVmRp.setPath("resourcePool");
tSpecVmRp.setSkip(false);

// add the network and resource pool traversal specs
// to the virtual machine traversal;
// the accessor method (getSelectSet) returns a reference
// to the mapped XML representation of the list; using this
// reference to add the spec will update the selectSet list
tSpec.getSelectSet().add(tSpecVmN);
tSpec.getSelectSet().add(tSpecVmRp);

// specify the properties for retrieval
// (virtual machine name, network summary accessible, rp runtime props);
// the accessor method (getPathSet) returns a reference to the mapped
// XML representation of the list; using this reference to add the
// property names will update the pathSet list
PropertySpec pSpec = new PropertySpec();
pSpec.setType("VirtualMachine");
pSpec.getPathSet().add("name");
```

```
PropertySpec pSpecNs = new PropertySpec();
pSpecNs.setType("Network");
pSpecNs.getPathSet().add("summary.accessible");

PropertySpec pSpecRPr = new PropertySpec();
pSpecRPr.setType("ResourcePool");
pSpecRPr.getPathSet().add("runtime.cpu.maxUsage");
pSpecRPr.getPathSet().add("runtime.memory.maxUsage");
pSpecRPr.getPathSet().add("runtime.overallStatus");

// create a PropertyFilterSpec and add the object and
// property specs to it; use the getter methods to reference
// the mapped XML representation of the lists and add the specs
// directly to the objectSet and propSet lists
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);
fSpec.getPropSet().add(pSpecNs);
fSpec.getPropSet().add(pSpecRPr);

// Create a list for the filters and add the spec to it
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);

// get the data from the server
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(propColl,fSpecList,ro);

// go through the returned list and print out the data
if (props != null) {
  for (ObjectContent oc : props.getObjects()) {
```

```
        String value = null;
        String path = null;
        List<DynamicProperty> dps = oc.getPropSet();
        if (dps != null) {
          for (DynamicProperty dp : dps) {
            path = dp.getName();
            if (path.equals("name")) {
              value = (String) dp.getVal();
            }
            else if (path.equals("summary.accessible")) {
              // summary.accessible is a boolean
              value = String.valueOf( dp.getVal() );
            }
            else if (path.equals("runtime.cpu.maxUsage")) {
              // runtime.cpu.maxUsage is an xsd:long
              value = String.valueOf( dp.getVal() );
            }
            else if (path.equals("runtime.memory.maxUsage")) {
              // runtime.memory.maxUsage is an xsd:long
              value = String.valueOf( dp.getVal() );
            }
            else if (path.equals("runtime.overallStatus")) {
              // runtime.overallStatus is a ManagedEntityStatus enum
              value = String.valueOf( dp.getVal() );
            }

            System.out.println(path + " = " + value);
          }
        }
      }
    }
  }//end collectProperties()
```

```
  // Authentication is handled by using a TrustManager and supplying
  // a host name verifier method. (The host name verifier is declared
  // in the main function.)
  //
  // For the purposes of this example, this TrustManager implementation
  // will accept all certificates. This is only appropriate for
  // a development environment. Production code should implement certificate support.
  private static class TrustAllTrustManager
      implements javax.net.ssl.TrustManager,
                 javax.net.ssl.X509TrustManager {

    public java.security.cert.X509Certificate[] getAcceptedIssuers() {
      return null;
    }

    public boolean isServerTrusted(java.security.cert.X509Certificate[] certs) {
      return true;
    }

    public boolean isClientTrusted(java.security.cert.X509Certificate[] certs) {
      return true;
    }

    public void checkServerTrusted(java.security.cert.X509Certificate[] certs, String
authType)
        throws java.security.cert.CertificateException {
      return;
    }

    public void checkClientTrusted(java.security.cert.X509Certificate[] certs, String
authType)
        throws java.security.cert.CertificateException {
      return;
```

```
    }
  }

  public static void main(String [] args) throws Exception {

    // arglist variables
    String serverName = args[0];
    String userName = args[1];
    String password = args[2];
    String url = "https://"+serverName+"/sdk/vimService";

    // Variables of the following types  for access to the API methods
    // and to the vSphere inventory.
    // -- ManagedObjectReference for the ServiceInstance on the Server
    // -- VimService for access to the vSphere Web service
    // -- VimPortType for access to methods
    // -- ServiceContent for access to managed object services
    ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
    VimService vimService;
    VimPortType vimPort;
    ServiceContent serviceContent;

    // Declare a host name verifier that will automatically enable
    // the connection. The host name verifier is invoked during
    // the SSL handshake.
    HostnameVerifier hv = new HostnameVerifier() {
      public boolean verify(String urlHostName, SSLSession session) {
        return true;
      }
    };

    // Create the trust manager.
```

```
javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
trustAllCerts[0] = tm;

// Create the SSL context
javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL");

// Create the session context
javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();

// Initialize the contexts; the session context takes the trust manager.
sslsc.setSessionTimeout(0);
sc.init(null, trustAllCerts, null);

// Use the default socket factory to create the socket for the secure connection
javax.net.ssl.HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());

// Set the default host name verifier to enable the connection.
HttpsURLConnection.setDefaultHostnameVerifier(hv);

// Set up the manufactured managed object reference for the ServiceInstance
SVC_INST_REF.setType("ServiceInstance");
SVC_INST_REF.setValue("ServiceInstance");

// Create a VimService object to obtain a VimPort binding provider.
// The BindingProvider provides access to the protocol fields
// in request/response messages. Retrieve the request context
// which will be used for processing message requests.
vimService = new VimService();
vimPort = vimService.getVimPort();
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
```

```
    // Store the Server URL in the request context and specify true
    // to maintain the connection between the client and server.
    // The client API will include the Server's HTTP cookie in its
    // requests to maintain the session. If you do not set this to true,
    // the Server will start a new session with each request.
    ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);
    ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);
    // Retrieve the ServiceContent object and login
    serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
    vimPort.login(serviceContent.getSessionManager(),
                  userName,
                  password,
                  null);

    // retrieve data
    collectProperties( vimPort, serviceContent );

    // close the connection
    vimPort.logout(serviceContent.getSessionManager());

  }
}
```

SelectionSpec Traversal

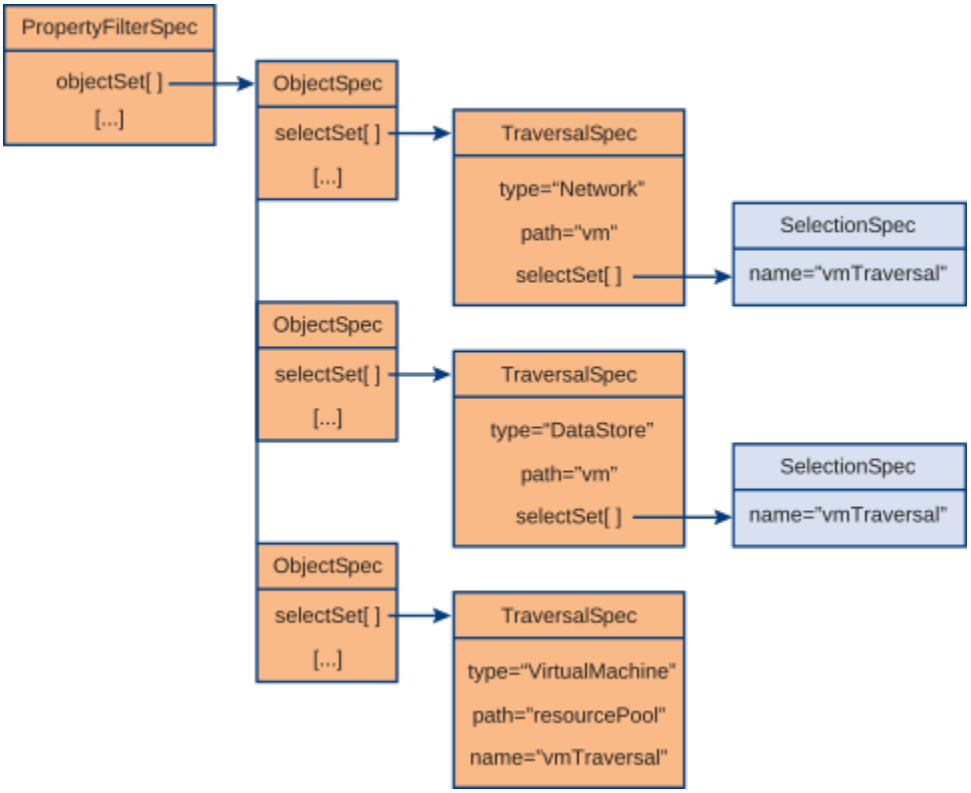
The **selectSet** array in **ObjectSpec** and **TraversalSpec** objects can include **TraversalSpec** objects and **SelectionSpec** objects. **SelectionSpec** is the base class for **TraversalSpec** objects. **SelectionSpec** defines the **name** property. You can use a **SelectionSpec** object in a **selectSet** array as a reference to a named **TraversalSpec** object. By using **SelectionSpec** references, you can reuse a **TraversalSpec** and you can define recursive traversal.

Simple Reference SelectionSpec

Use **SelectionSpec** references to avoid writing duplicate **TraversalSpec** declarations.

The **TraversalSpec** identified in a **SelectionSpec** reference must be within the same **PropertyFilterSpec**. Figure: SelectionSpec Reference shows the use of **SelectionSpec** references to a virtual machine **TraversalSpec**. The **SelectionSpec** references are associated with **Network** and **Datastore** traversals.

Figure 17: SelectionSpec Reference



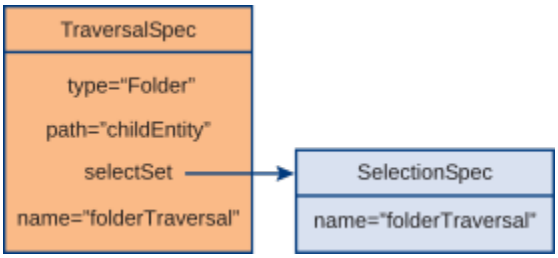
If the **ObjectSpec.selectSet** array contains a **SelectionSpec**, the referenced **TraversalSpec** must identify the same object type. `TraversalSpec.type` must match the type of the object specified in `ObjectSpec.obj`. The **PropertyCollector** applies the **TraversalSpec** to the object and uses the `TraversalSpec.path` property to extend its traversal.

Recursive Traversal

Use a **SelectionSpec** to apply a **TraversalSpec** to the results of its own traversal. To use a recursive filter construction, create a **SelectionSpec** that specifies the name of a **TraversalSpec** and add it to the named **TraversalSpec** selection set. The recursive construction extends inventory traversal beyond the paths directly represented by **TraversalSpec** objects.

You can use recursive traversal on any inventory objects that can be nested. See Inventory Hierarchies and ServiceInstance for a general representation of the structure of an inventory. For example, on a vCenter Server, folders can nest to arbitrary depths. To describe a traversal path through a succession of folders, you can add a **SelectionSpec** to the **Folder TraversalSpec**. The **SelectionSpec** must reference the **TraversalSpec**. Figure: Recursive TraversalSpec and SelectionSpec shows a representation of a **TraversalSpec** and its associated **SelectionSpec** for nested folder traversal.

Figure 18: Recursive TraversalSpec and SelectionSpec



Use Recursive TraversalSpec to Traverse Nested Inventory Folders

To traverse inventory objects that are nested to a variable depth, define one or more recursive `TraversalSpec` objects. In each `TraversalSpec` object, you specify a property of one managed object that holds a reference to another managed object. The `TraversalSpec` defines a path that the property collector can follow between objects.

To define recursive inventory traversal, use the following steps.

1. Use the **`SearchIndex`** managed object to retrieve the managed object reference for the top-level virtual machine folder.

   This folder is used as the beginning of the inventory traversal. For more information see SearchIndex.

2. Create an **`ObjectSpec`** object that references the top-level virtual machine folder.
3. Create a **`SelectionSpec`** object that references the **`Folder TraversalSpec`** by name.
4. Create a named **`TraversalSpec`** for **`Folder`** objects.

   The `TraversalSpec.path` property identifies the **`Folder.childEntity`** property for traversal to any child objects.

5. Add the **`SelectionSpec`** to the **`TraversalSpec`** to create the recursive filter.
6. Add the **`TraversalSpec`** to the **`ObjectSpec`**.
7. Create a **`PropertySpec`** for the **`Folder`** name.
8. Add the object and property specifications to the **`PropertyFilterSpec`**.
9. Call the **`RetrievePropertiesEx`** method.

Nested Folder Traversal in Java

This example shows the use of a recursive **`TraversalSpec`** with the property collector. The recursive traversal spec enables the property collector to navigate a chain of nested objects of the same type.

Nested Folder Traversal

```
import com.vmware.vim25.*;


import java.util.*;

import javax.net.ssl.HostnameVerifier;

import javax.net.ssl.HttpsURLConnection;

import javax.net.ssl.SSLSession;

import javax.xml.ws.BindingProvider;
```

```
import javax.xml.ws.soap.SOAPFaultException;

// PropertyCollector example
// command line input: server name, user name, password

public class nestedTraversal {

  private static void collectProperties(VimPortType methods,
                                        ServiceContent sContent)
                      throws Exception {

    // Get reference to the PropertyCollector
    ManagedObjectReference propColl = sContent.getPropertyCollector();

    // get the top-level vm folder mor
    ManagedObjectReference sIndex = sContent.getSearchIndex();
    ManagedObjectReference rootVmFolder =
              methods.findByInventoryPath(sIndex,"datacenter1/vm");
    // create an object spec to define the beginning of the traversal;
    // root vm folder is the root object for this traversal
    ObjectSpec oSpec = new ObjectSpec();
    oSpec.setObj(rootVmFolder);
    oSpec.setSkip(true);

    // folder traversal reference
    SelectionSpec sSpecF = new SelectionSpec();
    sSpecF.setName("traverseFolder");

    // create a folder traversal spec to select childEntity
    TraversalSpec tSpecF = new TraversalSpec();
    tSpecF.setType("Folder");
    tSpecF.setPath("childEntity");
```

```
tSpecF.setSkip(false);
tSpecF.setName("traverseFolder");

// use the SelectionSpec as a reflexive spec for the folder traversal;
// the accessor method (getSelectSet) returns a reference to the
// mapped XML representation of the list; using this reference
// to add the spec will update the list
tSpecF.getSelectSet().add(sSpecF);

// add folder traversal to object spec
oSpec.getSelectSet().add(tSpecF);

// specify the property for retrieval (folder name)
PropertySpec pSpec = new PropertySpec();
pSpec.setType("Folder");
pSpec.getPathSet().add("name");

// create a PropertyFilterSpec and add the object and
// property specs to it; use the getter method to reference
// the mapped XML representation of the lists and add the specs
// directly to the lists
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);

// Create a list for the filter and add the spec to it
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);

// get the data from the server
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(propColl,fSpecList,ro);
```

```
  // go through the returned list and print out the data
  if (props != null) {
    for (ObjectContent oc : props.getObjects()) {
      String folderName = null;
      String path = null;
      List<DynamicProperty> dps = oc.getPropSet();
      if (dps != null) {
        for (DynamicProperty dp : dps) {
          folderName = (String) dp.getVal();
          path = dp.getName();
          System.out.println(path + " = " + folderName);
        }
      }
    }
  }
}//end collectProperties()

// Authentication is handled by using a TrustManager and supplying
// a host name verifier method. (The host name verifier is declared
// in the main function.)
//
// For the purposes of this example, this TrustManager implementation
// will accept all certificates. This is only appropriate for
// a development environment. Production code should implement certificate support.
private static class TrustAllTrustManager
               implements javax.net.ssl.TrustManager,
               javax.net.ssl.X509TrustManager {

  public java.security.cert.X509Certificate[] getAcceptedIssuers() {
    return null;
  }
```

```
    public boolean isServerTrusted(java.security.cert.X509Certificate[] certs) {
      return true;
    }

    public boolean isClientTrusted(java.security.cert.X509Certificate[] certs) {
      return true;
    }

    public void checkServerTrusted(java.security.cert.X509Certificate[] certs, String
authType)
        throws java.security.cert.CertificateException {
      return;
    }

    public void checkClientTrusted(java.security.cert.X509Certificate[] certs, String
authType)
        throws java.security.cert.CertificateException {
      return;
    }
  }

  public static void main(String [] args) throws Exception {

    // arglist variables
    String serverName = args[0];
    String userName = args[1];
    String password = args[2];
                String url = "https://"+serverName+"/sdk/vimService";
    // Variables of the following types  for access to the API methods
    // and to the vSphere inventory.
    // -- ManagedObjectReference for the ServiceInstance on the Server
    // -- VimService for access to the vSphere Web service
    // -- VimPortType for access to methods
```

```
// -- ServiceContent for access to managed object services
ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
VimService vimService;
VimPortType vimPort;
ServiceContent serviceContent;

// Declare a host name verifier that will automatically enable
// the connection. The host name verifier is invoked during
// the SSL handshake.
HostnameVerifier hv = new HostnameVerifier() {
  public boolean verify(String urlHostName, SSLSession session) {
    return true;
  }
};

// Create the trust manager.
javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
trustAllCerts[0] = tm;

// Create the SSL context
javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL");

// Create the session context
javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();

// Initialize the contexts; the session context takes the trust manager.
sslsc.setSessionTimeout(0);
sc.init(null, trustAllCerts, null);

// Use the default socket factory to create the socket for the secure connection
javax.net.ssl.HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
```

```
// Set the default host name verifier to enable the connection.
HttpsURLConnection.setDefaultHostnameVerifier(hv);

// Set up the manufactured managed object reference for the ServiceInstance
SVC_INST_REF.setType("ServiceInstance");
SVC_INST_REF.setValue("ServiceInstance");

// Create a VimService object to obtain a VimPort binding provider.
// The BindingProvider provides access to the protocol fields
// in request/response messages. Retrieve the request context
// which will be used for processing message requests.
vimService = new VimService();
vimPort = vimService.getVimPort();
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();

// Store the Server URL in the request context and specify true
// to maintain the connection between the client and server.
// The client API will include the Server's HTTP cookie in its
// requests to maintain the session. If you do not set this to true,
// the Server will start a new session with each request.
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

// Retrieve the ServiceContent object and login
serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
vimPort.login(serviceContent.getSessionManager(),
              userName,
              password,
              null);

// retrieve data
```

```
    collectProperties( vimPort, serviceContent );


    // close the connection
    vimPort.logout(serviceContent.getSessionManager());


  }
}
```

Client Data Synchronization with WaitForUpdatesEx

To maintain a client-side representation of server object state (by monitoring the properties for the inventory), use the **`CreateFilter`** and **`WaitForUpdatesEx`** methods. **`WaitForUpdatesEx`** supports an incremental retrieval model.

When the **`WaitForUpdatesEx`** call returns an **`UpdateSet`**, apply the updates as patches to the properties of managed objects in your client-side copy. In some circumstances, the server can return redundant updates. If the client response includes side effects in addition to updating client data, the additional actions should be idempotent.

The **`WaitForUpdatesEx`** call cannot guarantee synchronization between client and server in the presence of rapid server data changes, but when the server state is static calls to **`WaitForUpdatesEx`** result in eventual synchronization.

> **NOTE**
> If you create additional instances of **`PropertyCollector`**, each instance uses independent version IDs for data synchronization. You should maintain a separate client copy of server data for each **`PropertyCollector`** instance when you use **`WaitForUpdatesEx`**.

> **IMPORTANT**
> The filters you use for incremental retrieval persist for the duration of the session or until you destroy them.

Property Filtering

A **`PropertyCollector`** can have one or more associated **`PropertyFilter`** objects. A **`PropertyFilter`** object has one associated **`PropertyFilterSpec`** object.

Some differences apply when filtering properties for **`RetrievePropertiesEx`** or for **`WaitForUpdatesEx`**.

When you call **`RetrievePropertiesEx`**, you supply a **`specSet`** parameter that can specify more than one **`PropertyFilterSpec`**. When you call **`CreateFilter`** you can specify only one **`PropertyFilterSpec`**.

A call to **`CreateFilter`** creates a persistent server-side filter belonging to the PropertyCollector. These filters are for use with successive calls to **`WaitForUpdatesEx`**, made for the same PropertyCollector instance. A call to **`RetrievePropertiesEx`** creates a set of nonpersistent filters that the server uses to select properties for the **`RetrievePropertiesEx`** call and for any associated **`ContinueRetrievePropertesEx`** calls made with the returned **`token`**.

To create persistent property filter specifications, use the **`CreateFilter`** method. When you call **`CreateFilter`**, you pass a **`PropertyFilterSpec`** object to the method. The method adds the new filter to the **`PropertyCollector`** associated with the method invocation and returns a reference to the new filter. After you have created the filter, you can

add additional **PropertyFilterSpec** objects. You cannot share a filter with a **PropertyCollector** in another session.

A **RetrievePropertiesEx** call returns a change set consisting of the union of results for all the filter specifications in the **specSet** parameter of the call. A **WaitForUpdatesEx** call returns a change set consisting of individual results, one result set for each persistent **PropertyFilter** attached to the PropertyCollector.

You can call **WaitForUpdatesEx** when the PropertyCollector has no **PropertyFilter** objects, but the call will block until you create a filter or until the call times out.

WaitForUpdatesEx

The **WaitForUpdatesEx** method supports a polling mechanism for property collection that is based on a specified wait time.

Specify the following parameters when you call **WaitForUpdatesEx**:

- Managed object reference to a **PropertyCollector** instance.
- **version** value that identifies a sequence value. The first time you call **WaitForUpdatesEx**, specify an empty string (“”) to retrieve a complete set of results for the specified properties. Your subsequent calls should use the version value returned in the previous call. If you don’t include the version value, the server returns everything. For more information about data versions, see Server Data Transmission.
- **options** specifying the amount of data to transmit in a single response (the **WaitOptions.maxObjectUpdates** property) and the number of seconds the **PropertyCollector** should wait for updates (the **WaitOptions.maxWaitSeconds** property).

The value of the **WaitOptions.maxWaitSeconds** property determines whether the **PropertyCollector** uses an instant retrieval or a polling model. When you call **WaitForUpdatesEx** with a wait time of 0, it checks for updates and returns immediately. When you call **WaitForUpdatesEx** with a wait time greater than 0, the method waits until the specified time or until a change. **WaitForUpdatesEx** will not return a result until updates occur or until the call times out. The time-out is affected by the **maxWaitSeconds** value, the amount of time it takes to collect updated property values, and **PropertyCollector** policy.

If the property collection operation times out, and there are no updates to the requested properties, the **PropertyCollector** returns null for the **WaitForUpdatesEx** response.

- **maxWaitSeconds** is an optional property. If you do not specify a value, the **PropertyCollector** waits as long as possible for updates. Therefore, if **maxWaitSeconds** is unset, the **waitForUpdatesEx** method will cause the server to wait if there are no more changes to report, until the server times out the communication and drops the request. You can use **maxWaitSeconds** to specify a shorter time-out interval if the client-side firewall times out sooner than the server.
- **maxWaitSeconds** set to zero specifies an immediate call and response. The **PropertyCollector** checks for updates for all properties specified by the union of all filters associated with that instance of the **PropertyCollector**. The **PropertyCollector** returns any results, or `null` if there have been no updates.
- **maxWaitSeconds** greater than zero specifies how long the server should wait before timing out the request. If there are no updates available within **maxWaitSeconds**, **WaitForUpdatesEx** returns **null** instead of a set of updates.

Comparing Usage of MaxWaitSeconds

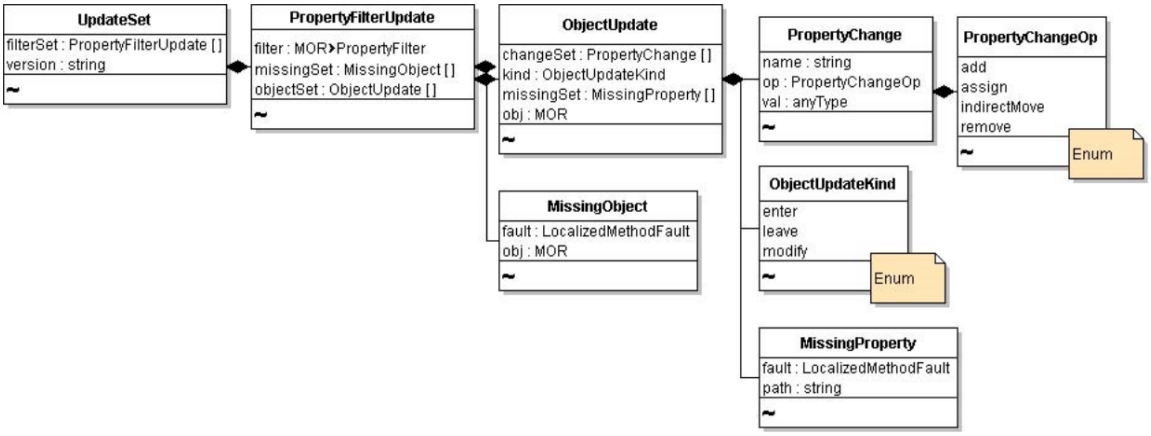
The following table lists some of the advantages and disadvantages of these two operations.

Operation	Advantages	Disadvantages
MaxWaitSeconds=0	Returns only properties that have changed since the version specified. Use this to poll for changes without waiting for changes.	Returns an empty set when nothing has changed on the server. Depending on your client application, this might be inefficient.
MaxWaitSeconds>0	Waits until an update occurs or a time-out occurs. Efficient use of network resources.	This can block a thread until updates occur. However, you can cancel this call while waiting, by using **CancelWaitForUpdates**.

UpdateSet Data Object from the Property Collector

The **WaitForUpdatesEx** method returns an **UpdateSet** data object, the composite data structure shown in the following figure.

Figure 19: UpdateSet Data Object Returned by WaitForUpdates Operations



Server Data Transmission

Property collection can involve the retrieval of large amounts of data, depending on the number of properties implied in the collection request. The server supports segmented data transmission, or chunking, when it sends collected data to a client. If the amount of collected data exceeds the chunk size, the server returns a chunk of data in a single response, and indicates additional data can be retrieved.

- The **WaitForUpdatesEx** method returns an **UpdateSet** data object. The **UpdateSet.truncated** property indicates whether the server has returned all the updates in its reply. The reply size can be limited by **options.maxObjectUpdates** in the call to **WaitForUpdatesEx**, or by server policy. On your first call to **WaitForUpdatesEx**, you can test **UpdateSet.truncated** in the response to see if all properties are present. If not, use **UpdateSet.version** in another call to **WaitForUpdatesEx**, repeating until **UpdateSet.truncated** is false. At that point, you can be sure that your client copy of the filtered properties is complete.
- The **RetrievePropertiesEx** method returns a **RetrieveResult** data object. The **RetrieveResult.token** property indicates whether you must call the **ContinueRetrievePropertiesEx** method to retrieve additional data. If the **token** property has a value, it identifies chunked data. When your client application receives an indication that additional data are available, it must send the returned token in the subsequent call to **ContinueRetrievePropertiexEx** to retrieve the next chunk of data.

Version strings and tokens are sequenced. Your client application must keep track of the sequence of values. If an error interrupts the collection operation, resume the operation by using the version string or token that was submitted before the interruption.

For information about chunk size, see descriptions of properties **`RetrieveOptions.maxObjects`** and **`WaitOptions.maxObjectUpdates`** in the *`vSphere API Reference`*.

> **NOTE**
> If you resend a version string with **`WaitForUpdatesEx`**, you might receive more updates than the first time you used that version string. When you resend, the server repeats the data collection and more changes might have happened in between calls to **`WaitForUpdatesEx`**.

PropertyCollector Performance

Various factors affect the performance of a **`PropertyCollector`** for any given session.

- Number of objects
- Number of properties
- Density of property data (composite, nested data objects)
- Frequency of changes to the objects and properties on the server
- Depth of traversal (number of properties traversed)

In addition, a vSphere server is affected by the number of **`PropertyCollector`** instances and the number of filters each instance is supporting across all sessions on the server.

To minimize **`PropertyCollector`** overhead and the amount of network traffic for your client application, use **`View`** objects with the **`PropertyCollector`**. Simple PropertyCollector Example (Java) illustrates using views with the **`PropertyCollector`**.

SearchIndex

The **`SearchIndex`** managed object provides a set of methods to retrieve references to managed objects in the vSphere inventory. You can search by managed objects inventory path, IP address, datastore path, DNS name, and various other identifying attributes.

For example, if you know the IP address of a virtual machine, you can obtain its managed object reference by using the **`SearchIndex.FindByIp`** method. You can use **`SearchIndex`** to obtain the reference to a server object, and then use that reference as the starting object for property collection. See the sample applications `SearchIndex.java` and **`SearchIndex.cs`** for more information about using **`SearchIndex`**. See the *`vSphere API Reference`* for more information about **`SearchIndex`** methods.

Authentication and Authorization

VMware vSphere implements mechanisms to ensure that only valid users can access virtual infrastructure components.Each property and method in the API has an associated privilege requirement, and only uses with corresponding privileges can access the entities. This chapter discusses approaches to securing the system and the related service interfaces. The chapter also discusses the user model, which is different in ESXi systems and vCenter Server systems.

See the *`vSphere Datacenter Administration Guide`* for a list of required privileges for common tasks and best practices for roles and permissions.

Objects for Authentication and Authorization Management

VMware vSphere includes the following interfaces for authenticating users and protecting virtual infrastructure components from unauthorized access:

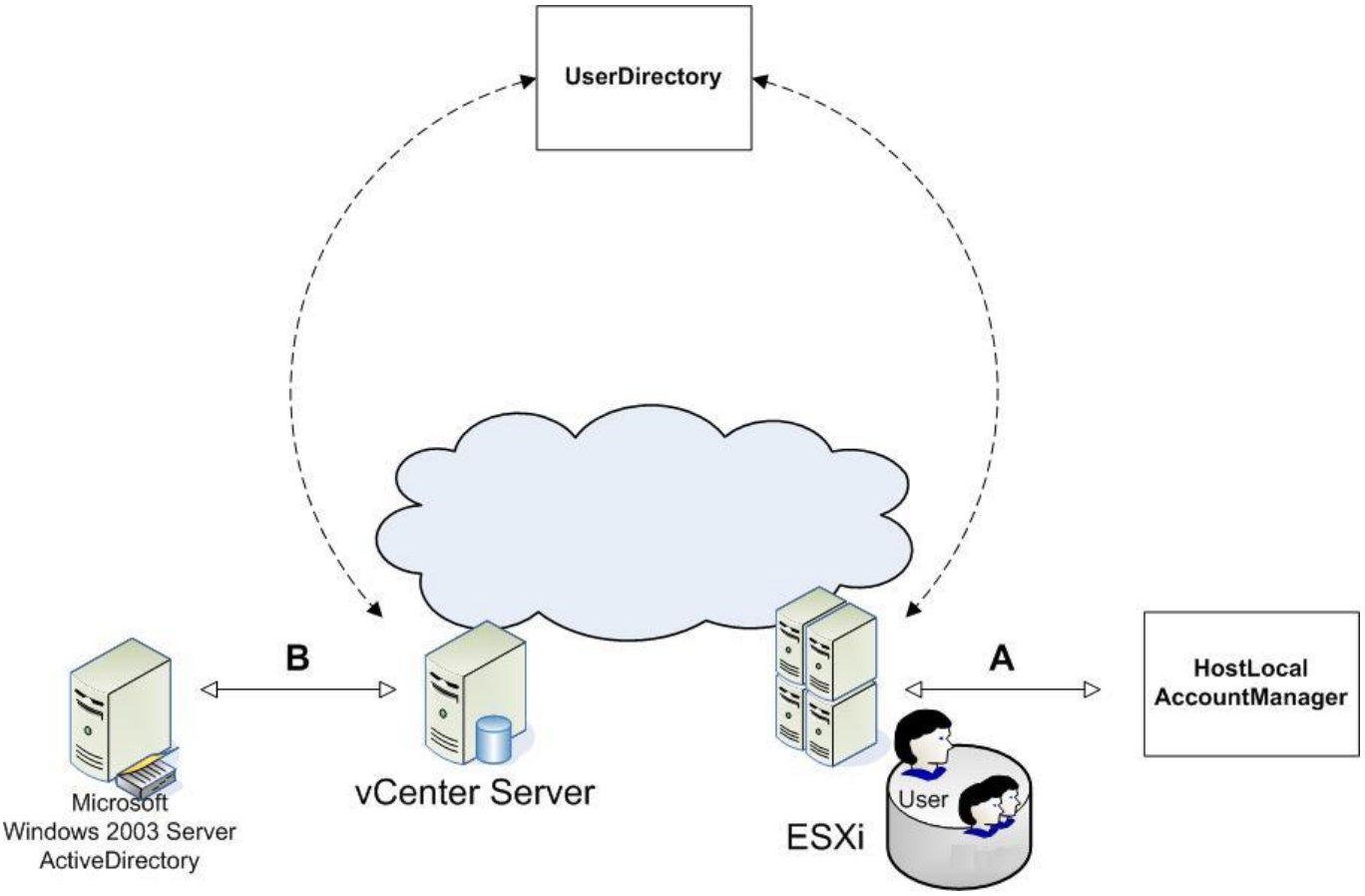
- **`HostLocalAccountManager`** is used to create and manage user accounts on ESXi systems. Authenticated users can view objects or invoke operations on the server depending on the permissions associated with their account. See Managing ESXi Users with HostLocalAccountManager.

- **AuthorizationManager** protects vSphere components from unauthorized access. Access to components is role-based: Users are assigned roles that encompass the privileges needed to view and perform operations on vSphere objects. **AuthorizationManager** has operations for creating new roles, modifying roles, setting permissions on entities, and handling the relationship between managed objects and permissions.
- **UserDirectory** provides a look-up mechanism that returns user-account information to **AuthorizationManager** or to another requestor, such as a client application. See Obtaining User and Group Information from UserDirectory.
- **SessionManager** provides an interface to the authentication infrastructure on the target server system (see Authenticating Users Through SessionManager).
  - For vCenter Server systems, **SessionManager** supports single sign-on based on SSO tokens obtained from a VMware SSO Server. See Establishing a Single Sign-On Session with a vCenter Server.
  - For ESXi systems, **SessionManager** supports authenticating user accounts as defined on the host system, such as accounts created using vSphere Client or accounts created programmatically through the **HostLocalAccountManager** API.
- Even if a user is authorized to perform operations on a vSphere object, the operation fails if the licenses for the host or the feature have not been assigned. You use **LicenseManager** and **LicenseAssignmentManager** to manage the licenses. See Managing Licenses with LicenseManager.

Authentication and Authorization for ESXi and vCenter Server

Several server-side mechanisms authenticate a human user when a client application, such as the vSphere Client or a vSphere Web Services SDK application, connects to the server. Because ESXi uses Linux-based authentication, and vCenter Server ran on Windows, the two systems can use different approaches for handling user accounts. The following illustration shows two different user management mechanisms associated with the VMware vSphere server.

Figure 20: Managed Objects for Handling User Accounts



These services work together to ensure that only authenticated users can connect to ESXi or vCenter Server systems, and that they can access only those objects—folders, virtual machines, datacenters, virtual services, and so on—for which they have the required privileges and which they are authorized to use or to view.

In addition, the vSphere Web Services SDK supports automated login through a credential store. See Using the Credential Store for Automated Login.

ESXi User Model

When users enter their user account and credential from a client application, the server consults the appropriate user account store and validates the authenticity of the user account and the associated credential. Currently, the credential consists of a password, but vSphere also supports certificates, such as X.509 certificates. Authenticated users can then access objects they are authorized to use. Authentication succeeds if a user identity exists as a user account on the target system or in a supported directory service.

ESXi leverages standard Linux infrastructure, including the Linux pluggable authentication module (PAM) mechanism for user account creation and management. The VMware authentication daemon (**`vmware-authd`**) is implemented as a PAM module. You can create and manage user accounts on an ESXi system by using **`HostLocalAccountManager`**.

vCenter Server User Model

The pre-appliance vCenter Server was a Windows-based service using native Windows facilities and the Windows user model for identification and authentication. The vCenter Server Web service is associated with the Windows user account that was logged in to the machine for vCenter Server installation. This vCenter Server administrator account was a member of the local Windows Administrator group on the machine.

VMware recommended creating a dedicated Windows user account for installing and managing the vCenter Server system. Other vCenter Server users who connect to the Web service also needed a Windows account on the local Administrator group.

More recently the vCenter Server Appliance uses the Platform Services Controller for authentication.

> **IMPORTANT**
> Even if a user with the same name exists on an ESXi host and a vCenter Server system, the two users have different accounts.

For details, see the *`Datacenter Administration Guide`* in the VMware vSphere documentation set.

Organizations that are using Microsoft Active Directory can use the user identities contained in a Windows Server domain controller or Active Directory service across their virtual infrastructure. Microsoft Active Directory identities are supported for all clients that run vSphere Web Services SDK applications from Windows-based systems.

A vCenter Server client uses a SAML token to establish a single sign on session with the Server. See Establishing a Single Sign-On Session with a vCenter Server.

vSphere Security Model

Although the details of authentication and authorization differ between ESXi and vCenter Server, the model itself is the same for both systems. It relies on privileges, roles, and permissions.

Privileges

A privilege is a system-defined requirement associated with a VMware vSphere object. Privileges are defined by VMware. Privileges are static, and do not change for a single version of a product. Each managed object has one or more privileges that a principal (user, group member) must have to invoke an operation or to view a property. For example, managed

entities such as **`Folder`** and **`VirtualMachine`** require the principal to have the `System.Read` privilege on the entity to view the values of its properties.

The *`vSphere API Reference`* includes information about privileges required to invoke operations and to view properties on the **Required Privileges** labels on the documentation page for each managed object. Privileges for vSphere components are defined as follows:

```
<group>[.<group>].privilege
```

For example:

`Datacenter.Create`

`Host.Config.Connection`

`Host.Config.Snmp`

A privilege might be specific to vCenter Server or to ESXi systems. For example, the **`Alarm.Create`** privilege is defined on vCenter Server. Setting alarms is done through the **`AlarmManager`** service interface, which requires a running vCenter Server system.

Privilege requirements apply to system objects regardless of how a given client application attempts to access server content (vSphere Client, CLI, or SDK). For example, you can use the following URL to access virtual machine datastore files:

```
https://<hostname>/folder[/<path>]/?dcPath=<datacenter_path>[&dsName=<datastore_name>]
```

The URL accesses a **`Datastore`** object in the inventory. You must have privileges to access each object in the hierarchy, corresponding to the elements of the URL.

Privileges for Datastore Objects in the Web Services API

The following table shows the privileges needed by methods that access datastore objects..

Object Associated with File	URL Element	Required Privileges
Root folder	`/folder`	`System.View`
Datacenter	**`?dcPath`**	**`Datastore.Browse`**
		`Datastore.FileManagement`
Datastore	**`&dsName`**	**`Datastore.Browse`**
		`Datastore.FileManagement`
Host	`/host`	**`Host.Config.AdvancedConfig`**
	`/tmp/`	**`Host.Config.SystemManagement`**

Roles

A role is a predefined set of privileges. Users are granted privileges to objects through roles.

When you assign user or group permissions, you pair the user or group with a role and associate that pairing with an inventory object. A single user might have different roles for different objects in the inventory.

For example, if you have two resource pools in your inventory, Pool A and Pool B, you might assign a particular user the role Virtual Machine User on Pool A and the role **`ReadOnly`** on Pool B. These assignments allow that user to turn on virtual machines in Pool A. In Pool B, the user can view the status of virtual machines, but cannot turn on virtual machines.

Permissions

In vSphere, a permission consists of a user or group and an assigned role for an inventory object, such as a virtual machine or ESXi host. Permissions grant users the right to perform the activities specified by the role on the object to which the role is assigned.

For example, to configure memory for an ESXi host, a user must be granted a role that includes the **`Host.Configuration.Memory`** privilege. By assigning different roles to users or groups for different objects, you can control the tasks that users can perform in your vSphere environment.

Many tasks require permissions on more than one object.

Setting Up Users, Groups, and Permissions

Setting up users, groups, and permissions consists of these tasks:

1. Get information about privilege requirements and privileges associated with system and sample roles.
   - Find out which operations on vSphere objects require which privileges. See the *`API Reference`*.
   - Find out which operations the system roles and sample roles can perform. See Description of Roles on vSphere Servers.
2. If necessary, create additional roles (sets of privileges). See Modifying Sample Roles to Create New Roles.
3. Retrieve information about existing users and groups (see Obtaining User and Group Information from UserDirectory) and create additional groups if needed.
4. Associate users or groups with roles using permissions. See Managing Roles and Permissions with AuthorizationManager.

At runtime, use **`SessionManager`** to log in to the server. vCenter Servers support single sign-on sessions. To establish a single sign-on session, use the **`SessionManager.LoginByToken`** method. To establish a session with a standalone ESXi host, use the **`SessionManager.Login`** method.

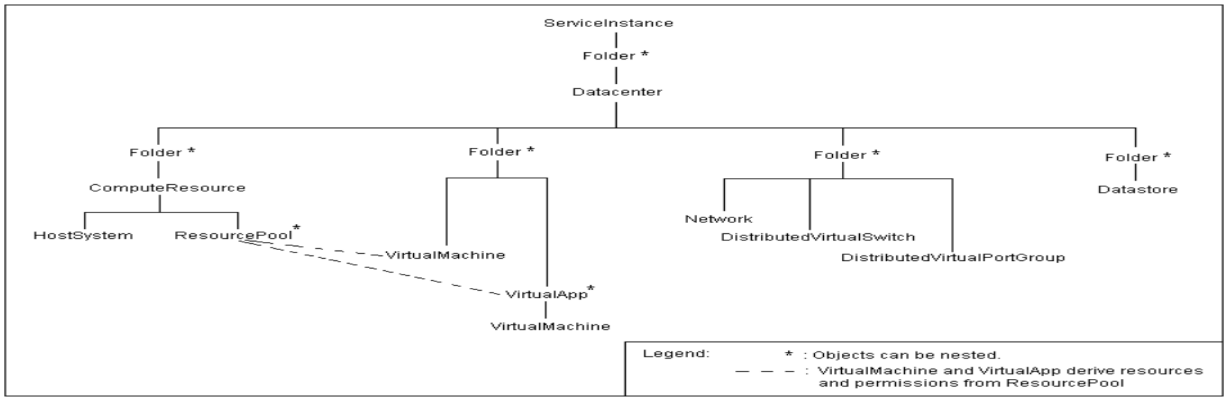
Obtaining User and Group Information from UserDirectory

The **`UserDirectory`** managed object allows a client application to obtain information about users and groups on a VMware vSphere server. Properties and results vary, depending on whether the server is a vCenter Server or an ESXi system.

- vCenter Server system. Domain controller, Active Directory, or local Windows account repository.
- ESXi host. Linux password file in `/etc/passwd` on the host.

For example, vCenter Server user accounts can be managed in a Windows Active Directory server or domain controller from which the **`domainList`** property of **`UserDirectory`** is derived. For ESXi systems, the **`domainList`** property is empty.

Figure 21: UserDirectory Managed Object



`RetrieveUserGroups` Method

`UserDirectory` allows you to obtain information about users and groups using the **`RetrieveUserGroups`** method. The method can obtain a list of all user accounts from the host, and can search for specific users or groups based on specific criteria to filter the results. You can search by user name, by group name, for an exact match, or for a partial string (substring).

- ESXi does not support local user groups, so this method will not return group information for a host. This method will return information about Active Directory groups.
- For ESXi systems, search returns all users from the **`passwd`** file. If this file contains Network Information System (NIS) or NIS+ users, **`RetrieveUserGroups`** returns these accounts as well.
- For vCenter Server, search is limited to the specified Windows domain. If the domain is omitted, the search is performed on local users and groups.

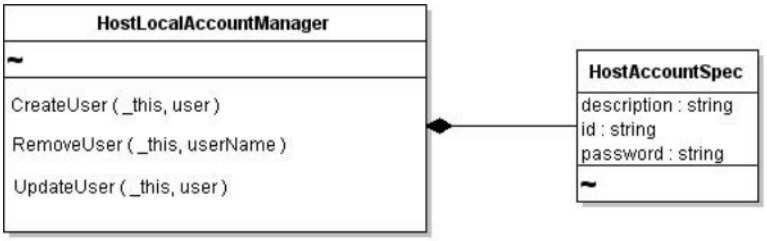
> **IMPORTANT**
> Do not configure an ESXi system to use NIS or NIS+, unless it is acceptable to have NIS (or NIS+) user information available through the **`UserDirectory.RetrieveUserGroups`** API.

Managing ESXi Users with HostLocalAccountManager

The **`HostLocalAccountManager`** managed object supports user administration tasks. **`HostLocalAccountManager`** is available only on ESXi system.

> **IMPORTANT**
> vCenter Server systems use different methods. See vCenter Server User Model.

Figure 22: HostLocalAccountManager Managed Object



Methods Available for Local Account Management

HostLocalAccountManager provides the following methods for local user account management:

- **CreateUser**
- **RemoveUser**
- **UpdateUser**

These methods accept a **HostAccountSpec** data object. Specify the object properties according to the requirements on the target system. Examples of user account requirements are password length requirements and restricted use of dictionary words.

Create a Local User Account on an ESXi System

You can use the API to create a user account on an ESXi system.

1. Obtain a managed object reference to the **HostLocalAccountManager** of the target system.
2. Create a **HostAccountSpec** data object that defines the properties of the user account, including description and password.

   Define account names and passwords according to the configuration required by your ESXi system for user account naming conventions and password requirements, such as minimum length, character set, and other requirements.

3. Call the **HostLocalAccountManager.CreateUserAccount** method, passing in the managed object reference (from step 1) and the **HostAccountSpec** data object (step 2).

   After creating user accounts on the ESXi system, you can grant these users access to virtual components by using **AuthorizationManager** methods. See Managing Roles and Permissions with AuthorizationManager.

Managing Roles and Permissions with AuthorizationManager

AuthorizationManager is the service interface for handling permissions and roles assigned to the users and groups you define with **HostLocalAccountManager**. **AuthorizationManager** methods allow you to create, modify, and manage roles and permissions, and to obtain information about the roles and permissions defined in the system. If a predefined role does not meet your needs, define a new one that contains only the minimum set of required privileges.

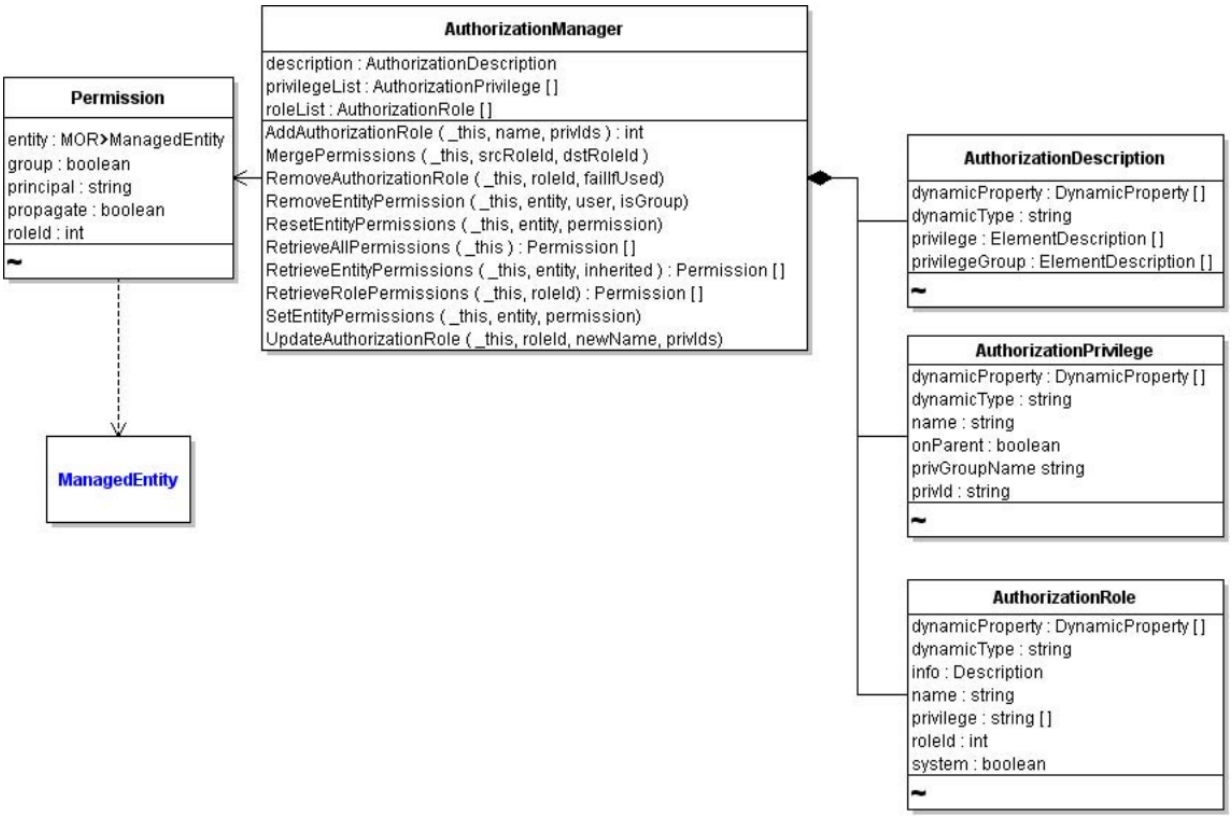
The **AuthorizationManager** also allows access and prevents access to specific server objects based on the permissions associated with the object.

AuthorizationManager includes methods for managing roles and for managing permissions:

- Roles Management. **AddAuthorizationRole**, **RemoveAuthorizationRole**, and **UpdateAuthorizationRole**. See Using Roles to Consolidate Sets of Privileges and Modifying Sample Roles to Create New Roles.
- Permissions Management. **MergePermissions**, **RemoveEntityPermission**, **ResetEntityPermissions**, **RetrieveAllPermissions**, **RetrieveEntityPermissions**, **RetrieveRolePermissions**, and **SetEntityPermissions**. See Granting Privileges Through Permissions.

The following diagram shows these methods in a UML diagram for **AuthorizationManager** and some of its associated data objects.

Figure 23: AuthorizationManager Managed Object



`AuthorizationManager` properties allow access to information. For example:

- The **`privilegeList`** property returns a list of all privileges defined on the system, as an array of **`AuthorizationPrivilege`** data objects. Privileges are defined by VMware, on the objects and properties contained in the system. These privileges are fixed and cannot be changed by client applications.
- The **`roleList`** property returns a list of all currently defined roles, including the system-defined roles, as an array of **`AuthorizationRole`** data objects.

Using Roles to Consolidate Sets of Privileges

A role is a named set of one or more privileges. A role is normally defined for a group of people who have common responsibilities in the system, for example, administrators. Each role can have zero to multiple privileges. ESXi defines system roles and user roles.

- System roles. Cannot be modified or deleted.
- User roles. Apply to different user communities or restrict access for add-on tools. Several predefined user roles are included with vCenter Server and with ESXi systems. You can create new roles using these predefined user roles as a starting point.

Description of Roles on vSphere Servers

The following table describes system roles and user roles in more detail and lists currently available roles as examples.

Type	Role name	Role ID	Description
System Roles	Administrator	-1	Superuser access. Encompasses the set of all defined privileges. This role cannot be deleted. By default, the Administrator role is granted to the user or group that owns the root node.
	Anonymous	-4	Cannot be granted. Default access role associated with any user account that has logged in.
	No Access	-5	No access. Explicitly denies access to the user or group with this role. Assigning this role to a user account prevents the user from seeing any objects. Use the **`No Access`** role to mask subobjects under a higher-level object that has propagated permissions defined.
	Read-Only	-2	Read-only access. Encompasses the set of all nonmutable privileges. (**`System.Anonymous`**, `System.Read`, and `System.View`). Equivalent to a user role with no permissions. Users with this role can read data or properties and call query methods, but cannot make changes to the system.
	View	-3	Visibility access consisting of **`System.Anonymous`** and `System.View` privileges. Cannot be granted.
Sample Roles	Virtual Machine Administrator	1	Set of privileges necessary to manage virtual machines and hosts within the system.
	Datacenter Administrator	2	Set of privileges necessary to manage resources, but not interact with virtual machines.
	Virtual Machine Provider	3	Set of privileges necessary to provision resources.
	Virtual Machine Power User	4	Set of privileges for a virtual machine user that can also make configuration changes and create new virtual machines.
	Virtual Machine User	5	Set of privileges necessary to use virtual machines only. Cannot reconfigure virtual machines.
	ResourcePool Administrator	6	Available on vCenter Server systems only.
	VMware Consolidated Backup Utility	7	Available on vCenter Server systems only. Set of privileges necessary to run the Consolidated Backup Utility.

Modifying Sample Roles to Create New Roles

The system roles cannot be modified or deleted. However, you can create new roles, or modify the sample roles.

To create new roles using the API, use the following steps.

1. Starting with the **`ServiceContent`** object in **`ServiceInstance.content`**, obtain a managed object reference to the **`AuthorizationManager`** for the server.
2. Invoke the **`AddAuthorizationRole`** method. Parameters are a reference to **`AuthorizationManager`**, a name for the role (as a string), and an array of privileges (array of strings) that should be assigned to the role.

   `AddAuthorizationRole` returns an integer (**`xsd:int`**) value for the **`roleId`** that the system assigns to the newly defined role.

3. In subsequent code, use the **`roleID`** to assign the role to specific users or groups.

Granting Privileges Through Permissions

When you use one of the **`AuthorizationManager`** objects to assign or modify permissions, you use a **`Permission`** data object. **`Permission`** associates a principal with a set of privileges. A permission identifies:

- The user or group (**principal**) to which the permission applies.
- The role containing the privileges that should be granted to the user or group.
- The managed object reference to the entity to which the permission applies.

Every managed entity has at least one **Permission** object associated with it. A managed entity can have more than one **Permission** assigned to it, effectively granting different privileges to different users or groups. Permissions are defined for managed entities either explicitly or through inheritance.

Obtaining Information About Permissions

Users with the Administrator role can obtain information about **Permission** objects at different levels of detail.

- For an array of **Permission** objects, call the **AuthorizationManager.RetrieveAllPermissions** method.
- For specific inventory objects, such as managed entities, folders, datacenters, or virtual services, call the **AuthorizationManager.RetrieveEntityPermissions** method.
- For a role defined in the system, call the **AuthorizationManager.RetrieveRolePermissions** method.

See the *vSphere API Reference*.

Setting, Changing, or Deleting Permissions

The **Permission** data object associates the privileges required to perform an action on an object with the principals (user, group). Principals have privileges through their role. To set or update permissions on an object, use the **AuthorizationManager.SetEntityPermissions** method.

To set permissions on an entity, use the following steps.

1. Obtain a reference to the **AuthorizationManager** for the server from the **ServiceContent** object associated with the **ServiceInstance**. For example:

   ```
   ManagedObjectReference hostAuthorizationManager = service.getAuthorizationManager();
   ```

2. Create a **Permission** data object that identifies the user (or group) name, the role, the entity to which the permission should apply, and whether the permission should be applied to the entity's children.

   For example, the following code fragment creates a permission on the root folder of the inventory granting a user Administrator role to the root folder and all its children.

   ```
   Permission per = new Permission();

   per.setGroup(false);

   per.setPrincipal("new_user_name");

   per.setRoleId(-1);

   per.setPropagate(true);

   per.setEntity(rootFolder);
   ```

   Permissions cannot be set directly on children in a complex entity. For complex entities, set permissions on the parent entity and set the **propagate** flag to true to apply permissions to the child entities.

   To replace existing permissions with a new set of permissions, use the **AuthorizationManager.ResetEntityPermissions** method.

Impact of Group Membership on Permissions

Users can be members of multiple groups. The system handles multigroup membership as follows:

- Permissions are applied to inventory objects from the containing object to each of its child entities.
- If a user has no explicit user-level permissions, group-level permissions apply as if granted to the user directly.
- Membership in multiple groups with permissions on the same object results in a union of permissions.
- User-level permissions always take precedence over group-level permissions.

Applying Permission to a Managed Entity

The following example shows some of the code required to create a user account and apply a permission to an entity that grants access to the user account based on a role.

The role with role ID 4, assigned in this example, is defined as a “Virtual Machine Power User.” The sample uses **`AuthorizationManager`** to grant permissions to the user and to associate the permission with the managed entity in the inventory—in this example, the **`rootFolder`**. The example uses the **`apputil`** helper classes to access the objects.

Creating a User Account

```
...

ManagedObjectReference _authManRef = _sic.getAuthorizationManager();

public class CreateUser {

private static AppUtil appUtil= null;

private void createUser() throws Exception {

ManagedObjectReference hostLocalAccountManager =
appUtil.getConnection().getServiceContent().getAccountManager();

ManagedObjectReference hostAuthorizationManager =

     appUtil.getConnection().getServiceContent().getAuthorizationManager();


// Create a user

HostAccountSpec hostAccountSpec = new HostAccountSpec();

hostAccountSpec.setId(userName);

hostAccountSpec.setPassword(password);

hostAccountSpec.setDescription("my delegated admin auto-agent software");

appUtil.getConnection().getService().createUser(hostLocalAccountManager, hostAccountSpec);

ManagedObjectReference rootFolder =
appUtil.getConnection().getServiceContent().getRootFolder();

Permission permission = new Permission();

permission.setGroup(false);

permission.setPrincipal(userName);
```

```
// Assign the Virtual Machine Power User role

permission.setRoleId(4);

permission.setPropagate(true);

permission.setEntity(rootFolder);

appUtil.getConnection().getService().setEntityPermissions(hostAuthorizationManager,
rootFolder,

     new Permission [] {permission});

...
```

Authenticating Users Through SessionManager

The **SessionManager** managed object controls user access to the server. **SessionManager** includes methods for logging in to the server, obtaining a session, and logging out. The **SessionManager** defines the lifetime and visibility of many objects. Session-specific objects are not visible outside the session in which they are created.

> **IMPORTANT**
> Each user session uses system resources and creates locks on the server side. Too many concurrent sessions can slow down the server. By default, vCenter Server terminates a session after 30 minutes.

Upon successful authentication of a user account, **SessionManager** returns a **UserSession** data object to the client application. The session is associated with that user account for the duration of the session. The client application can save the session locally, to a secure file, and reuse the session later to reconnect to the server. You can also configure an ESXi or vCenter Server system to support local sessions, which enable users with credentials on the host to log in based on those privileges.

The **SessionManager** provides these capabilities:

- Log in and log out. Basic operations to log in to ESXi or vCenter Server system, obtain a session, and log out. When a session terminates, all session-specific objects are destroyed.
- Impersonation. One user session adopts the authorization level of another user session. Impersonation is common in Web based scenarios in which a middle-tier application functions as a central account that interacts with other back-end servers or processes. Windows services impersonate a client when accessing resources on behalf of the client. **SesssionManager** supports impersonation through its **ImpersonateUser** method.
- Delegation. A client application that is running on behalf of a local user can call the **SessionManager.AcquireLocalTicket** method to obtain a one-time user name and password for logging in. Delegation is useful for host-based utilities that run in the local console.

If the user account associated with the session does not have the permissions required to perform an action, the **AuthorizationManager** returns a **NoPermission** fault to the client application.

Using VMware Single Sign On for vCenter Server Sessions

vSphere supports single sign on for a single point of authentication for vCenter Server clients. To use VMware Single Sign On, your vSphere Web Services SDK client connects to the VMware SSO Server to obtain an SSO token. Your client includes the token in the **SessionManager.LoginByToken** request to start a vSphere session.

For more information about creating a session, see Establishing a Single Sign-On Session with a vCenter Server.

Using the Credential Store for Automated Login

To facilitate automated login for unattended applications, the vSphere Web Services SDK includes client-side credential store libraries and tools for automating the login process in a more secure manner. The libraries eliminate the need for system administrators to keep passwords in local scripts.

IMPORTANT
These libraries are built on top of the vSphere Web Services SDK.

The credential store has the following components:

- A persistence file (credential store backing file) that stores authentication credentials. Currently, only passwords are supported. The persistence file maps a remote user account from an ESXi host to the password for that user on the server.
- Java and Perl libraries for managing the credential store programmatically. See Credential Store Methods for available methods.
- Java and Microsoft PowerShell-based command-line utilities for managing the credential store.

In addition to the libraries listed in Credential Store Libraries, the vSphere Web Services SDK includes the **`CredentialStoreAdmin`** tool for creating, examining, and managing the credential store. You can use the tool to examine the contents of the credential store, for example, the generated user accounts and passwords.

If you use these credential store client libraries in an application, you must set up the credential store on all client machines that run your application.

Credential Store Libraries

The following table shows the credential management libraries for Java.

Package com.vmware.security.credstore (Java)
CredentialStore.java
CredentialStoreFactory.java

Several of the helper classes provided with the sample applications use the credential store mechanism.

Credential Store Methods

The following table shows the methods used in Java clients for managing credentials.

Java	Description
`addPassword(hostname, username, password)`	Stores the password for the specified host and user. Overwrites any existing password for that user in the credential store. Creates the default credential store backing file in the default location (if it does not exist).
`removePassword(hostname, username)`	Deletes the password for the specified user from the credential store.
`clearPasswords()`	Deletes all passwords from the credential store.
`getPassword(hostname, username)`	Returns the password for the specified host and user from the credential store.
`getHosts()`	Returns the set of hosts contained in the credential store.
`getUsernames(hostname)`	Returns the collection of all user names that have passwords stored for the specified **`hostname`**.

Table continued on next page

Continued from previous page

Java	Description
`close()`	Closes the credential store, preventing further method invocations. Releases associated resources.

Credential Store Backing File

The credential store backing file is an XML file that is saved locally on the client machine for access at runtime. Unless otherwise specified, the backing file is located in the following location:

- Linux. `$HOME/.vmware/credstore/vicredentials.xml`
- Windows Vista and later. `C:\Users\[user_name]\AppData\Roaming\VMware\credstore\vicredentials.xml`
- Windows XP and Windows 2000. `C:\Documents and Settings\[user_name]\Application Data\VMware\credstore\vicredentials.xml`

The credential store persists locally on a per-user basis—each user has his or her own credential store backing file.

> **CAUTION**
> The credential store backing files use filesystem-level permissions to ensure that passwords remain confidential. Protect the credential store backing file with appropriate file permissions.

Credential Store File Format shows the XML elements that are read and written to the file.

Credential Store File Format

```
<?xml version="1.0" encoding="UTF-8"?>

     <viCredentials>

          <version>1.0</version>

          <passwordEntry>

              <host>mi6.vmware.com</host>

              <username>agent007</username>

              <password>IhWS1saIhtsw2FbIh0w2F2...</password>

          </passwordEntry>

          <passwordEntry>

              ...

          </passwordEntry>

          ...

     </viCredentials>
```

Credential Store Samples

The **`CreateUser`** and **`SimpleAgent`** sample applications demonstrate how to use the credential store client libraries.

- The **CreateUser** sample creates a user account and password for the server based on random-number-generation scheme. The sample populates the local credential store backing file with this information. If the backing file does not exist, it is created in the default location.
  When you run **CreateUser**, specify the name of an ESXi system, and an administrator user name and password. A user account name and password are created on the server. Specify **--ignorecert** unless your system has a secure connection to the target. Do not use **--ignorecert** in a production environment.

  ```
  java com.vmware.samples.simpleagent.CreateUser --server <servername> --url

  https://<servername>/sdk --username <adminuser> --password <pwd> --ignorecert
  ignorecert
  ```

  CAUTION
  The **CreateUser** sample application is for demonstration purposes only and should not be used as a model for production code. The sample breaks the principle of least privilege by granting the user account the Administrator role (-1). Never do this in a production environment.

- The **SimpleAgent** sample application demonstrates how to use credential store libraries to extract the user account and password at runtime to authenticate a user noninteractively.
  ```
  java com.vmware.samples.simpleagent.SimpleAgent <servername>
  ```

Specifying Roles and Users with the Credential Store

VMware recommends that you apply the principle of least privilege to any agent-like software or automated application that uses the credential store in a production environment. Give user accounts the minimal number of privileges on the system that they require to do their jobs.

Specify roles and users as follows:

1. For each SDK-based application, use one specific role, newly created or predefined, that has appropriate privileges.

   For example, if you are developing an agent-like application to automatically start the VMware Consolidated Backup utility, you might use the “VMware Consolidated Backup Utility” role (roleID 7).

   If no predefined user role that meets the needs of your application exists, create a role with only those privileges needed for the application. See Using Roles to Consolidate Sets of Privileges for more information about roles.

2. Create a user account for use with the agent or application.
3. Apply the role created in Step 1 to the user account created in Step 2.
4. Store the user account and password in the credential store, using the **CredentialStoreAdministration** tool.

   Never grant administrator privileges to a user account associated with an automated script or software agent, especially one that uses the credential store.

Managing Licenses with LicenseManager

When you want to perform tasks in the vSphere environment, you must have licenses to do so. Licensing applies to ESXi hosts, vCenter Server, and special features such as VMware HA or VMware vMotion.

The *vSphere Datacenter Administration Guide* explains how to manage ESXi and vCenter Server licenses using the vSphere Client, and gives background information about license keys, license inventory, and related topics.

You can also manage licenses using the **LicenseManager** and **LicenseAssignmentManager** managed objects. You use **LicenseManager** to explicitly manage the pool of available licenses on ESXi systems released before vSphere 4.0. You use **LicenseAssignmentManager**, available through the **LicenseManager.licenseAssignmentManager**

property, to manage assignment of licenses to entities in the vCenter Server inventory. You can retrieve information, add licenses, and remove licenses.

Retrieve Information

- Retrieve the **`LicenseManager.evaluation`** and **`LicenseManager.licenses`** properties to obtain information on evaluation licenses and full licenses.
- Call **`LicenseManager.DecodeLicense`** to decode license information. The call returns a **`LicenseManagerLicenseInfo`** data object, which encapsulates information about the license.
- Call **`LicenseAssignmentManager.QueryAssignedLicenses`** for information about assigned licenses.

Add Licenses

- Call **`LicenseManager.AddLicense`**, passing in a license key, to add a license to the inventory of available licenses.
- Call **`LicenseAssignmentManager.UpdateAssignedLicense`**, passing in a license key, to update the licenses for an entity, for example, a host system.

Remove Licenses

- Call **`LicenseAssignmentManager.RemoveAssignedLicense`** to remove all licenses from an entity, passing in an entity to remove licenses from. You can then assign those licenses to other entities.
- Call **`LicenseManager.RemoveLicense`**, passing in a license key, to remove a license from the inventory of available licenses.

Hosts

Many of the operations in your vSphere environment involve setting up the ESXi hosts on which the virtualization layer runs. You can set up storage and networking, and those settings directly affect the virtual machine. You must also manage other aspects of the host, as discussed in this chapter.

> **IMPORTANT**
> See the *`ESX Configuration Guide`* and the *`ESXi Configuration Guide`* for important information on security considerations, not included here.

Host Management Objects

The vSphere Web Services SDK includes several objects for host management.

The central object is **`HostSystem`**. Each property of **`HostSystem`** is a data object that encapsulates some information about the host. For example, the **`capability`** property is a **`HostCapability`** object, the **`runtime`** property is a **`HostRuntimeInfo`** object. See the *`API Reference`* for a list of the properties and the corresponding data objects.

`HostSystem` methods allow you to perform certain tasks on ESX/ESXi hosts. However, many tasks are not performed through **`HostSystem`** methods, but through methods in managed objects related to **`HostSystem`**. For example, you manage the host time using the **`HostDateTimeSystem`** and you manage kernel modules using **`HostKernelModuleSystem`**.

Retrieving Host Information

You retrieve information about the host by accessing data objects defined for the **`HostSystem`**.

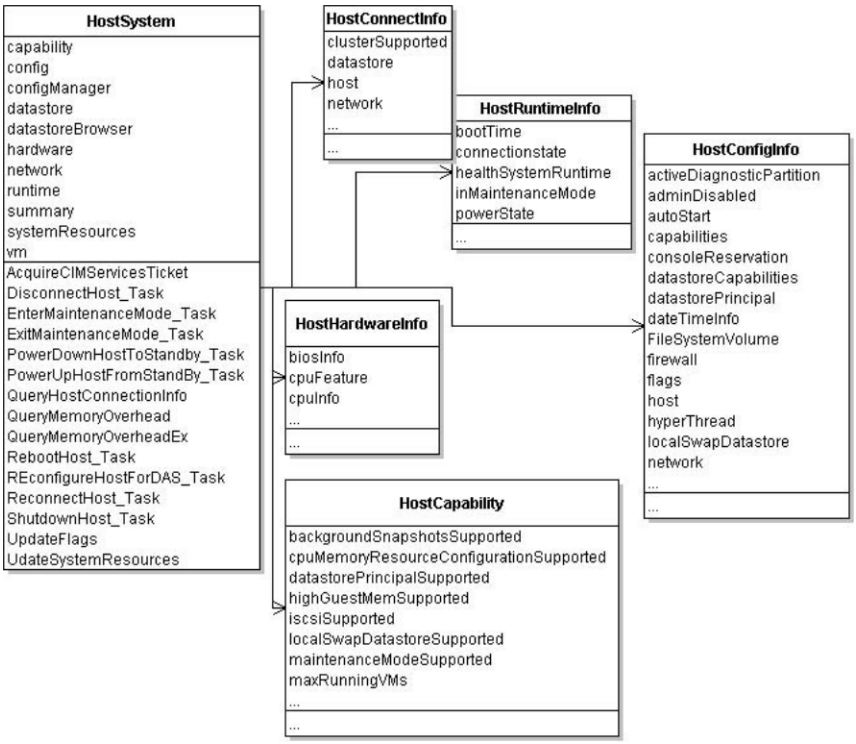
- **`HostSystem.capability`** is a **`HostCapability`** object. The **`HostCapability`** properties indicate the features that are supported by the host, for example, **`maintenanceModeSupported`** or **`recursiveResourcePoolsSupported`**.

- **HostSystem.runtimeInfo** is a **HostRuntimeInfo** object that contains several data objects with detailed information about the current state of the host. You can, for example, extract the health status as a **HealthSystemRuntime** object or the power state as a **HostPowerState** object.
- **HostSystem.hardware** is a **HostHardwareInfo** object that allows you to retrieve the host's hardware configuration including CPU and NUMA information and memory size.
- **HostSystem.config** is a **HostConfigInfo** object. This data object type encapsulates a typical set of host configuration information that is useful for displaying and configuring a host. You can access the **HostConfigInfo** object only on managed hosts, and only if the host is connected.

HostSystem has several additional properties that allow you to directly access the virtual machines, datastores, and networks associated with that system.

The **QueryHostConnectionInfo**, **QueryMemoryOverhead**, and **QueryMemoryOverheadEx** methods are available for information retrieval.

Figure 24: HostSystem and Information Properties



Configuring and Reconfiguring Hosts

When you configure or reconfigure an ESX/ESXi host, you usually do not use the methods in **HostSystem** directly, but work with managed objects available for configuration of that part of the system. For example, **HostNetworkSystem** allows you to configure the network, and **HostAuthorizationManager** is for managing users, groups, and permissions on a host. The objects and related methods are discussed in the corresponding chapters of this guide.

Some methods are defined locally in **HostSystem**. See the *vSphere API Reference* for details on each method.

- CIM Management – **AcquireCimServicesTicket**. For additional information on using vSphere with CIM, see the VMware CIM APIs documentation.
- Host Lifecycle – **RebootHost_Task**, **ShutdownHost_Task, PowerDownHostToStandBy_Task**, **PowerUpHostFromStandBy_Task**, **DisconnectHost_Task**, **ReconnectHost_Task**. See Managing the Host Lifecycle.

- Maintenance Mode – **EnterMaintenanceMode_Task**, **ExitMaintenanceMode_Task**.
- Updates – **UpdateFlags**, **UpdateIpmi**, **UpdateSystemResources**.

Managing the Host Lifecycle

A host's lifecycle depends in part on whether the host is a standalone host or managed by a vCenter Server system.

Reboot and Shutdown

You can reboot and shut down managed and standalone hosts. The **ShutdownHost_Task** method is not supported on all hosts. Check the host capability **shutdownSupported**.

You can call both methods with a **force** parameter, which specifies whether to reboot hosts even when virtual machines are running or other operations are in progress on the host. If you set the parameter to **false**, hosts are rebooted only when they are in maintenance mode.

- **ShutdownHost_Task** – Shuts down a host. If connected directly to the host, the client never receives an indicator of success in the returned task, but temporarily loses connection to the host. If the method does not succeed, an error is returned.
- **RebootHost_Task** – Reboots a host. If the command is successful, then the host has been rebooted. Clients connected directly to the host do not receive an indication of success in the returned task, but temporarily lose connection to the host. If the method does not succeed, an error is returned.

Using Standby Mode

Standby is a power state in which the host does not support provisioning or power on of virtual machines. VMware power management module might evacuate and put a host in standby mode to save power. The host can be powered up remotely by using **PowerUpHostFromStandBy_Task**.

The following methods support standby mode. Both methods are cancelable.

- **PowerDownHostToStandBy_Task** – Puts the host in standby mode, a mode in which the host is in a standby state from which it can be powered up remotely. The command is only supported on hosts on which the host capability **standbySupported** is **true**.
  While this task is running, no virtual machines can be powered on and no provisioning operations can be performed on the host.

  Calling this method does not directly initiate any operations to evacuate or power down powered-on virtual machines. However, if VMware DRS is enabled, the vCenter Server migrates powered-off virtual machines or recommends migration to a different host, depending on the automation level. If the host is part of a cluster and the task is issued with a vCenter Server target with the method's **evacuatePoweredOffVms** parameter set to **true**, the task does not succeed unless all the powered-off virtual machines are reregistered to other hosts.
- **PowerUpHostFromStandBy_Task** – Takes the host out of standby mode. If the command is successful, the host wakes up and starts sending heartbeats. This method might be called automatically by VMware DRS to add capacity to a cluster, if the host is not in maintenance mode.

Disconnecting and Reconnecting Hosts

You can make a host a managed host by adding it to the vCenter Server system. You can later disconnect and reconnect the host, for example, to refresh the agents.

You can use the following methods, which are only supported if you access the host through a vCenter Server system.

- **QueryHostConnectionInfo** – Returns a **HostConnectInfo** object, which is the same object that the **Datacenter.QueryConnectionInfo** returns. The information in this object can be used by a connection wizard, like the wizard used in the vSphere Client.
- **DisconnectHost_Task** – Disconnects from a host and instructs the vCenter Server system to stop sending heartbeats to the host.
- **ReconnectHost_Task** – Reconnects a host to the vCenter Server system. This process reinstalls agents and reconfigures the host, if it has gotten out of sync with the server. The reconnection process checks for the correct set of licenses and for the number of CPUs on the host, ensures the correct set of agents is installed, and ensures that networks and datastores are discovered and registered with the vCenter Server system.
  Client applications can change the IP address and port of the host when doing a reconnect operation. This can be useful if the client wants to preserve existing metadata, such as statistics, alarms, and privileges, even though the host is changing its IP address.

Querying and Changing the Host Time

The **HostDateTimeSystem** supports date and time related configuration on a host and supports both Network Time Protocol (NTP) and Precision Time Protocol (PTP) configuration.

The **HostDateTimeSystem.dateTimeInfo** property allows you to retrieve and set date and time information. The **HostDateTimeInfo** data object's properties contain these data objects for date and time management:

- **HostNTPConfig** contains a list of NTP servers for use by the host.
- `HostPTPConfig` contains device and networking information.
- **HostDateTimeSystemTimeZone** specifies the time zone including the GMT offset, identifier for the time zone, and name.

You can also query the host's time information by calling one of the **HostDateTimeSystem** methods.

- **QueryAvailableTimeZones** – Retrieves the list of available timezones on the host. The method uses the public domain **tz** timezone database. The method returns an array of **HostDateTimeSystemTimeZone** objects.
- **QueryDateTime** – Returns the current date and time on the host.

You can modify the host's date and time information by calling one of the following **HostDateTimeSystem** methods:

- **RefreshDateTimeSystem** – Refreshes the date and time related settings to pick up any changes that might have occurred.
- **UpdateDateTime** – Updates the date and time on the host using the date and time passed into the method. Use with caution. Network delays or execution delays can result in time skews.
- **UpdateDateTimeConfig** – Updates the date and time configuration of the host. You call this method with a **HostDateTimeConfig** parameter, which allows you to specify NTP configuration, PTP configuration, and the time zone.

Querying Virtual Machine Memory Overhead

Each virtual machine you power on requires a certain amount of memory for its use. In addition, the host must have some memory overhead available for each virtual machine. To find out about memory overheat, call the **HostSystem.QueryMemoryOverheadEx** method. The method takes a **virtualMachineConfigInfo** data object as an argument, and determines the amount of overhead necessary to power on a virtual machine with those characteristics.

The methods returns the amount of memory required, in bytes.

Storage

A virtual machine uses a virtual disk to store its operating system, program files, and other data. A virtual disk is a large physical file, or a set of files, that can be copied, moved, archived, and backed up like other files. To store and manipulate

virtual disk files, a host requires dedicated storage space. ESX/ESXi supports storage in multiple ways. Hosts that are managed by a vCenter Server system can share storage.

Any type of network-attached storage requires complete configuration of networking in the VMkernel to support network-based access to the storage media. The VMkernel requires its own IP address. See vSphere Networks.

Storage Management Objects

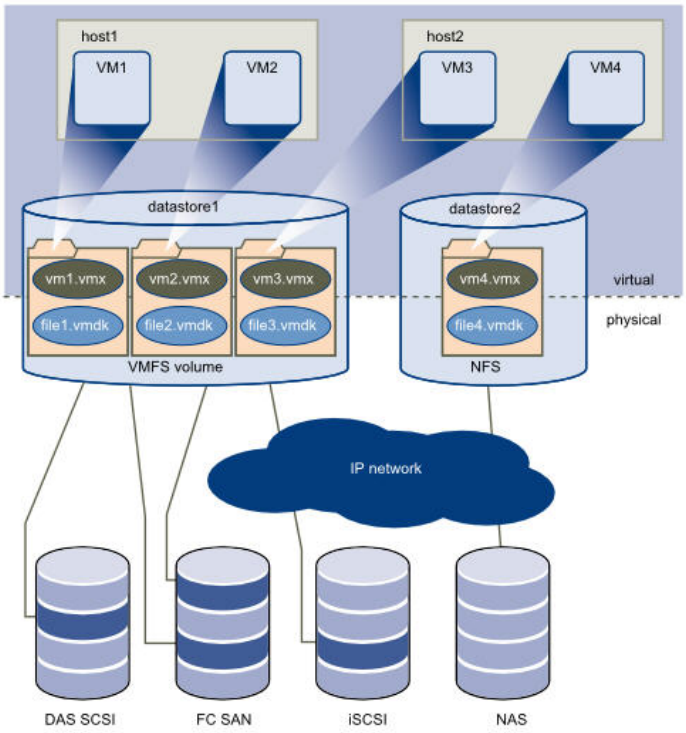
You can access the objects that support storage management through the **`HostSystem`** managed object.

- **`HostStorageSystem`** – The **`HostSystem.storageSystem`** property is a managed object reference to the **`HostStorageSystem`** of the ESX/ESXi system. **`HostStorageSystem`** is a low-level interface that is used mainly for configuring the physical storage. See Configuring Disk Partitions.
- **`HostDatastoreSystem`** – The **`HostSystem.datastoreSystem`** property is a managed object reference to a **`HostDatastoreSystem`** managed object. **`HostDatastoreSystem`** methods allow you to create, configure, extend, and remove datastores. While **`HostStorageSystem`** supports access and configuration of physical storage, **`HostDatastoreSystem`** supports access and configuration of logical storage through the volumes (**`Datastore`** managed objects) the host can use for virtual machines. See Creating and Managing Datastores.
- **`HostDatastoreBrowser`** – Provides access to the contents of one or more datastores. The items in a datastore are files that contain configuration, virtual disk, and other data associated with a virtual machine.
- **`Datastore`** – The **`Datastore`** managed entity provides methods for mounting datastores, browsing datastores, and obtaining information about the datastores associated with a virtual machine. See Creating and Managing Datastores.
- **`HostDiagnosticPartition`** – Supports creating and querying diagnostic partitions for your ESX/ESXi host. See Managing Diagnostic Partitions.

Introduction to Storage

The VMware vSphere storage architecture consists of layers of abstraction that hide and manage the complexity and differences of physical storage subsystems, shown in the following illustration.

Figure 25: Storage Architecture



How Virtual Machines Access Storage

Virtual machines use virtual disks for their operating system, application software, and other data files. A virtual disk is stored as a VMDK file on a datastore. The virtual disk hides the physical storage layer from the virtual machine's operating system. Regardless of the type of storage device that your host uses, the virtual disk always appears to the virtual machine as a local SCSI device. As a result, you can run operating systems that are not certified for specific storage equipment, such as SAN, in the virtual machine.

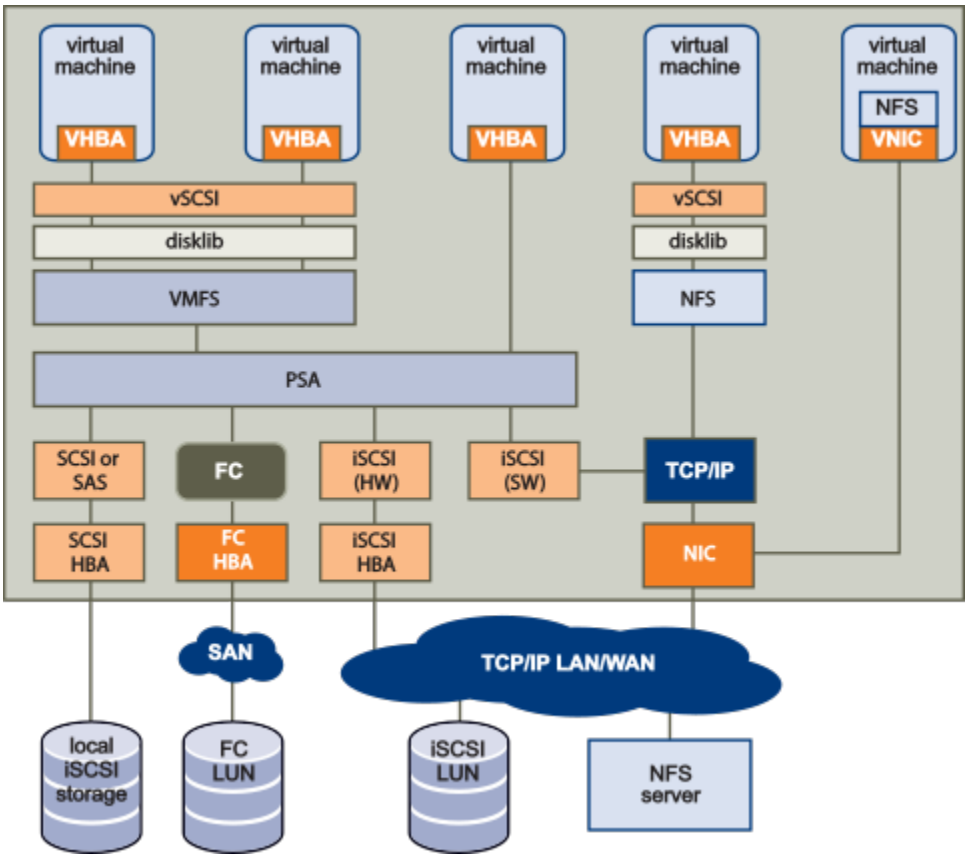
When a virtual machine communicates with its virtual disk stored on a datastore, it issues SCSI commands. Because datastores can exist on different types of physical storage, these commands are encapsulated into other forms, depending on the protocol that the ESX/ESXi host uses to connect to the physical storage device.

To the applications and guest operating systems running on each virtual machine, the storage subsystem appears as a virtual SCSI controller connected to one or more virtual SCSI disks as shown in the top half of Figure: Storage Architecture. These controllers are the only types of SCSI controllers that a virtual machine can see and access, and include the objects that extend **`VirtualSCSIController`**:

- **`ParaVirtualSCSIController`**
- **`VirtualBusLogicController`**
- **`VirtualLsiLogicController`**
- **`VirtualLsiLogicSASController`**

How precisely a virtual machine accesses storage depends on the setup of the host. Figure: Storage API Architecture gives an overview of the different possibilities.

Figure 26: Storage API Architecture



Datastores

A datastore is a manageable storage entity, usually used as a repository for virtual machine files including log files, scripts, configuration files, virtual disks, and so on. vSphere supports two types of datastores, VMFS and NAS.

- If you want to use a NAS volume, mount it using **`CreateNasDatastore`** and unmount it using **`RemoveDatastore`**. The two commands are host specific, you must invoke the create and remove methods on each host on which you want to mount or unmount the datastore.
- To create a VMFS datastore, call **`CreateVmfsDatastore`**, passing in any existing disk. As a result of the call, the disk is formatted with VMFS and the datastore is automounted on all ESX/ESXi hosts on which the disk is visible the next time you perform a rescan. When you call **`RemoveDatastore`** on a VMFS datastore, the datastore is destroyed. After a rescan, the datastore is no longer available to any ESX/ESXi systems. In contrast to NAS datastores, you do not have to invoke methods for creation and removal of the datastore on each host.

An ESX/ESXi host automatically discovers the VMFS volume on attached Logical Unit Numbers (LUNs) on startup and after re-scanning the host bus adapter. When you create a VMFS datastore, the datastore label is based on the VMFS volume label. If there is a conflict with an existing datastore, the label is made unique by appending a suffix. The VMFS volume label remains unchanged.

Destroying a VMFS datastore removes the partitions that compose the VMFS volume.

Datastores can span multiple physical storage devices. A single VMFS volume can contain one or more LUNs from a local SCSI disk array on a physical host, a Fibre Channel SAN disk farm, or iSCSI SAN disk farm. The ESX/ESXi system detects new LUNS that are added to any of the physical storage subsystems. When the user queries for a list of available devices, the newly discovered devices are included. You can extend storage capacity on an existing VMFS volume without powering down physical hosts or storage subsystems.

If any of the LUNs within a VMFS volume fails or becomes unavailable, only virtual machines with data on that LUN are affected. An exception is the LUN that has the first extent of the spanned volume (multi-extent volume). All other virtual machines with virtual disks residing on other LUNs continue to function normally.

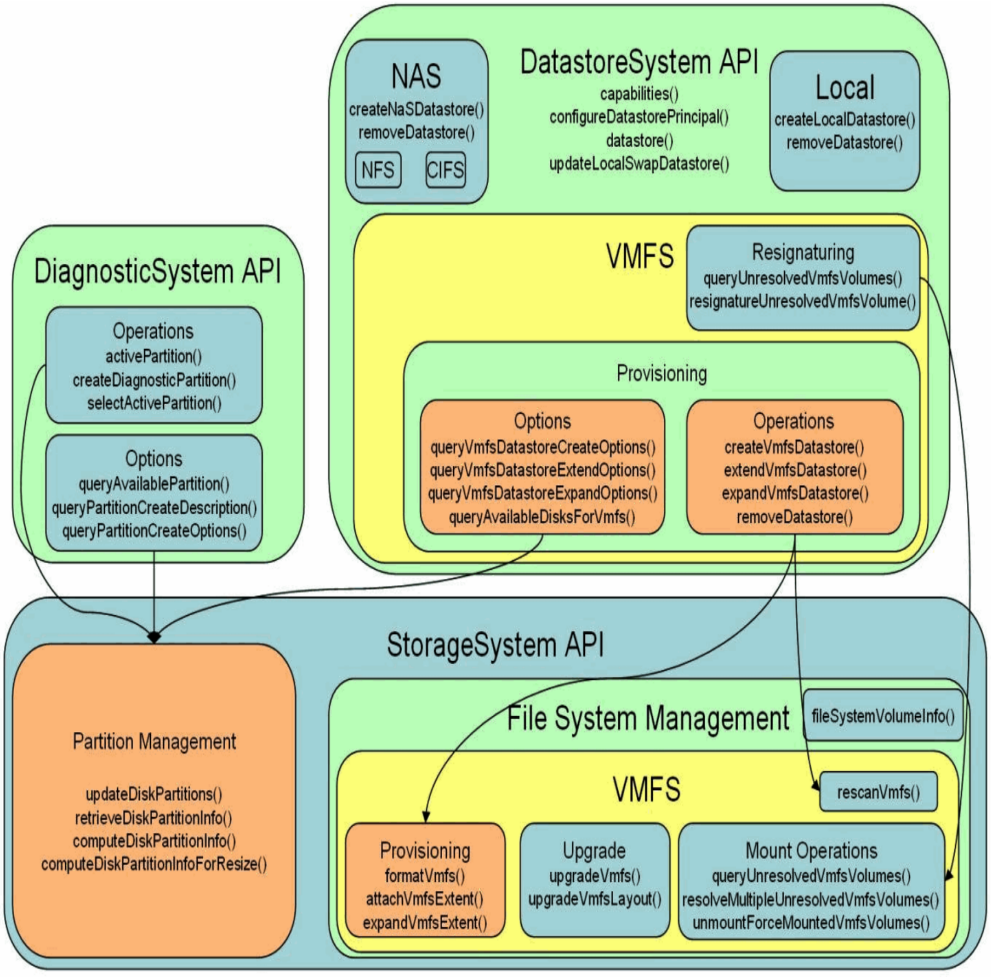
Choosing the Storage API to Use

The **`HostStorageSystem`** APIs are low-level enough for performing VMFS provisioning operations. They require a knowledge of partitioning details and VMFS extent composition. They do not enforce VMFS best practices like partition alignment and optimum VMFS block sizes, and they allow you to mix extents from different datastores on the same LUN and to add extents even though expansion is preferable in most cases.

The **`HostDatastoreSystem`** APIs are primarily used for managing VMFS volumes. They don't require an in-depth knowledge of storage systems, and do enforce best practices.

Figure: Storage APIs gives an overview of the different APIs. See Managed Objects for Working with Storage to determine which tasks are commonly performed with which API.

Figure 27: Storage APIs



Managed Objects for Working with Storage

The Web Services API provides several managed objects that are designed to work with ESXi storage devices.

Managed Object	Task	See
HostStorageSystem	Low-level operations associated with individual hosts, such as resizing or updating disk partitions.	Configuring Disk Partitions
HostStorageSystem	Multipath management.	Multipath Management
HostStorageSystem	iSCSI Storage setup and configuration.	Configuring iSCSI Storage
HostDatastoreSystem	Creating and managing VMFS datastores and remote datastores.	Creating and Managing Datastores
HostDatastoreSystem **HostStorageSystem**	Managing VMFS volume copies (resignature or force mount).	Managing VMFS Volume Copies (Resignaturing)
HostDiagnosticSystem	Creating and managing diagnostic partitions.	Managing Diagnostic Partitions

Configuring Disk Partitions

HostStorageSystem manages low-level storage components including HBAs, SCSI LUNs, file system volumes, and so on. You can use this API to set up the partitions before creating, extending, or expanding a VMFS file system.

- **ComputeDiskPartitionInfo** – Computes the disk partition information based on the specified disk layout. The server computes a new **HostDiskPartitionInfo** object for a specific disk using the layout that is specified by the **HostDiskPartitionLayout** object. Inside the **HostDiskPartitionLayout** object, you specify the list of block ranges for that partition, and optionally the total number and size of the blocks. You can then use that information inside the **HostDiskPartitionSpec** when updating a disk partition.
- **ComputeDiskPartitionInfoForResize** – Computes the disk partition information to support resizing a given partition. Returns the resized disk partition information as a **HostDiskPartitionInfo** object. You can then use that information inside the **HostDiskPartitionSpec** when resizing the disk partition.
- **RetrieveDiskPartitionInfo** – Allows you to specify an array of device path names that identify disks and returns an array of **HostPartitionInfo** objects for each of those disks.
- **UpdateDiskPartitions** – Changes the partitions on a disk by supplying a partition specification (**HostDiskPartitionSpec**) and device name.

After you have updated the disk partitions for the host, you must perform a rescan by using one of the following methods. Complete rescans might take a long time.

- **RefreshStorageSystem** – Refreshes the storage information and settings to pick up changes, but does not explicitly issue commands to discover new devices.
- **RescanAllHba** – Rescans all host bus adapters for new storage devices. This method might take a long time.
- **RescanHba** – Rescans a specific host bus adapter for new devices.

HostStorageSystem methods are also used for setting up iSCSI storage. See Configuring iSCSI Storage.

Multipath Management

The *vSphere Storage* documentation includes information about using multipathing for failover and load balancing. You can manage multipathing using the vSphere Client, the **esxcli** command, or using the following commands. Use the **HostStorageSystem.multipathStateInfo** property to access the **HostMultipathStateInfo** data object that describes runtime information about the state of multipathing on a given host.

- **EnableMultipathPath** – Enables a disabled path for a device. Use the pathname from **HostMultipathStateInfoPath** or **HostMultipathInfoPath**.
- **QueryPathSelectionPolicyOptions** – Obtains the set of path-selection-policy options. These options determine the path that can be used by a device that is managed by native multipathing. A **HostMultipathInfo** data object identifies the devices that are managed through native multipathing.
- **QueryStorageArrayTypePolicyOptions** – Obtains the set of storage-array-type policy options. These options determine the storage-array-type policies that a device that is managed by native multipathing might use. A **HostMultipathInfo** data object identifies the devices that are managed through native multipathing.
- **SetMultipathLunPolicy** – Updates the path selection policy for a LUN. Specify the LUN using the LUN UUID from the **HostMultipathInfoLogicalUnit** object.
- **DisableMultipathPath** – Disables an enabled path for a device. Use the pathname from **HostMultipathStateInfoPath** or **HostMultipathInfoPath**.

Configuring iSCSI Storage

vSphere supports software iSCSI, dependent hardware iSCSI, and independent hardware iSCSI. See *Configuring iSCSI Adapters and Storage* in the *vSphere Storage* documentation for a detailed discussion.

The following **HostStorageSystem** methods are available for iSCSI storage management.

- Add a dynamic or static target.
  - **AddInternetScsiSendTarget** – Adds send target entries to the host bus adapter discovery list if the **DiscoveryProperties.sendTargetsDiscoveryEnabled** flag is set to true.
  - **AddInternetScsiStaticTargets** – Adds static target entries to the host bus adapter discovery list. The **DiscoveryProperty.staticTargetDiscoveryEnabled** flag must be set to true.
- Configure targets.
  - **UpdateInternetScsiAdvancedOptions** – Updates the advanced options that the iSCSI host bus adapter or the discovery addresses and targets associated with it.
  - **UpdateInternetScsiAlias** – Updates the alias of an iSCSI host bus adapter.
  - **UpdateInternetScsiAuthenticationProperties** – Updates the authentication properties for one or more targets or discovery addresses associated with an iSCSI host bus adapter.
  - **UpdateInternetScsiDigestProperties** – Updates the digest properties for the iSCSI host bus adapter or the discovery addresses and targets associated with it.
  - **UpdateInternetScsiDiscoveryProperties** – Updates the discovery properties for an iSCSI host bus adapter.
  - **UpdateInternetScsiIPProperties** – Updates the IP properties for an iSCSI host bus adapter.
  - **UpdateInternetScsiName** – Updates the name of an iSCSI host bus adapter.
  - **UpdateSoftwareInternetScsiEnabled** – Enables and disables software iSCSI in the VMkernel.
- Remove a dynamic or static target.
  - **RemoveInternetScsiSendTargets** – Removes send target entries from the host bus adapter discovery list. The **DiscoveryProperty.sendTargetsDiscoveryEnabled** must be set to true. If any of the targets provided as parameters are not found in the existing list, the other targets are removed and an exception is thrown.
  - **RemoveInternetScsiStaticTargets** – Remove static target entries from the host bus adapter discovery list. The **DiscoveryProperty.staticTargetDiscoveryEnabled** must be set to true. If any of the targets

provided as parameters are not found in the existing list, the other targets are removed and an exception is thrown.

iSCSI initiators and targets have unique, permanent iSCSI names and addresses. An iSCSI name correctly identifies a specific iSCSI initiator or target, regardless of physical location. Names must be in EUI or IQN format, as specified by the storage vendor's hardware.

Configure the VMkernel To Support Software iSCSI

Before you can set up iSCSI on a system, you must create a dedicated VMkernel network interface. See. You can then enable the VMkernel to support iSCSI and configure the initiator.

To enable the VMkernel to support software iSCSI, use the following steps.

1. Obtain a managed object reference to the host system's **`HostStorageSystem`**.
2. Invoke the **`UpdateSoftwareInternetScsiEnabled`** method, passing the reference to the **`HostStorageSystem`** and the value **`true`**.

Configure iSCSI Initiators

After you have configured the VMkernel to support software iSCSI, you can configure the storage initiators for iSCSI.

To configure iSCSI initiators, use the following steps.

1. Access the list of available HBAs on the host system.

   You can do this by creating a property collector with **`HostSystem`** as the starting point. See Property Collector. From the **`HostSystem.config`** property, you can obtain the list (array) of host bus adapters by specifying this property path:

   ```
   config.storageDevice.hostBusAdapter
   ```

   The property path returns an array of host bus adapters. For example:

   ```
   hostBusAdapter["key-vim.host.BlockHba-vmhba32"]

   hostBusAdapter["key-vim.host.BlockHba-vmhba33"]

   hostBusAdapter["key-vim.host.BlockHba-vmhba34"]

   hostBusAdapter["key-vim.host.BlockHba-vmhba35"]

   hostBusAdapter["key-vim.host.BlockHba-vmhba1"]

   ...
   ```

2. From the array, select the host bus adapter (instance of **`HostHostBusAdapter`**) that you want to configure and obtain its **`key`** property, which is the device name of the host bus adapter as a string.
3. Determine the capabilities of the adapter by retrieving the properties of the **`HostHostBusAdapter`** object.
4. Configure the initiator.
   - For an independent hardware initiator, configure the IP address.
   - For a software initiator, enable the software initiator in the VMkernel.
5. Configure the iSCSI name by calling **`HostStorageSystem.UpdateInternetScisiName`** and the alias by running **`HostStorageSystem.UpdateInternetScisiAlias`**.
6. Configure target discovery by calling **`HostStorageSystem.UpdateInternetScisiHbaDiscoveryProperties`**.

   The method takes a **`HostInternetScisiHbaDiscoveryProperties`** data object that you can configure.

7. (Optional) Set the authentication information by calling **HostStorageSystem.UpdateInternetScisiAuthenticationProperties**.

   The **HostInternetScsiHbaAuthenticationProperties** object you pass into that method includes properties for configuring CHAP and Mutual CHAP. See the *vSphere Storage* documentation for information about securing your iSCSI storage array.

8. Configure access to the targets.
9. Rescan the HBAs.

   Rescan enables the HBAs to discover the new storage devices. You can either rescan a single HBA with **HostStorageSystem.RescanHba**, specifying the HBA ID as a parameter, or rescan all HBAs using **HostStorageSystem.RescanAllHba**.

Creating and Managing Datastores

Each datastore is a logical container, analogous to a file system on a logical volume, where the host places virtual disk files and other virtual machine files. Datastores hide specifics of the physical storage device and provide a uniform model for storing virtual machine files.

The **HostDatastoreSystem** managed object provides methods for creating and managing datastores. All **HostDatastoreSystem** methods require a managed object reference to **HostDatastoreSystem**, and return a reference to the **Datastore** object after it is created.

HostDatastoreSystem allows you to create and expand, query, and remove or update datastores. **HostDatastoreSystem** also allows you to configure a datastore principal for a host by calling **ConfigureDatastorePrincipal.** All virtual machine-related file I/O is performed under this user.

VMFS provisioning tasks are often performed as follows:

1. Call **QueryAvailableDisksForVmfs** to get the subset of disks that are well suited for holding VMFS datastores. **QueryAvailableDisksForVmfs** obtains a list of disks that can be used to contain VMFS datastore extents. You can provide a datastore name to obtain the list of disks that can contain extents for the specified VMFS datastore. The operation does not return disks currently used by the VMFS datastore, nor does it return management LUNs and disks that are referenced by RDMs. RDM disks are not usable for VMFS datastores.
2. Get information about provisioning options by calling one of the following methods, passing in the selected disk:
   - **QueryVmfsDatastoreCreateOptions** – Obtains information about options for creating a new VMFS datastore on a disk. The method returns an array of **VmfsDatastoreOption** data objects.
   - **QueryVmfsDatastoreExpandOptions** – Obtains information about options for expanding the extents of an existing VMFS datastore.
   - **QueryVmfsDatastoreExtendOptions** – Obtains information about options for extending an existing VMFS datastore for a disk.
3. If required, change the layout by calling **HostStorageSystem.ComputeDiskPartitionInfo** and then **HostStorageSystem.UpdateDiskPartition** to resize the partition.
4. Call **CreateVmfsDatastore**, **ExtendVmfsDatastore**, or **ExpandVmfsDatastore** to complete the VMFS provisioning operation.

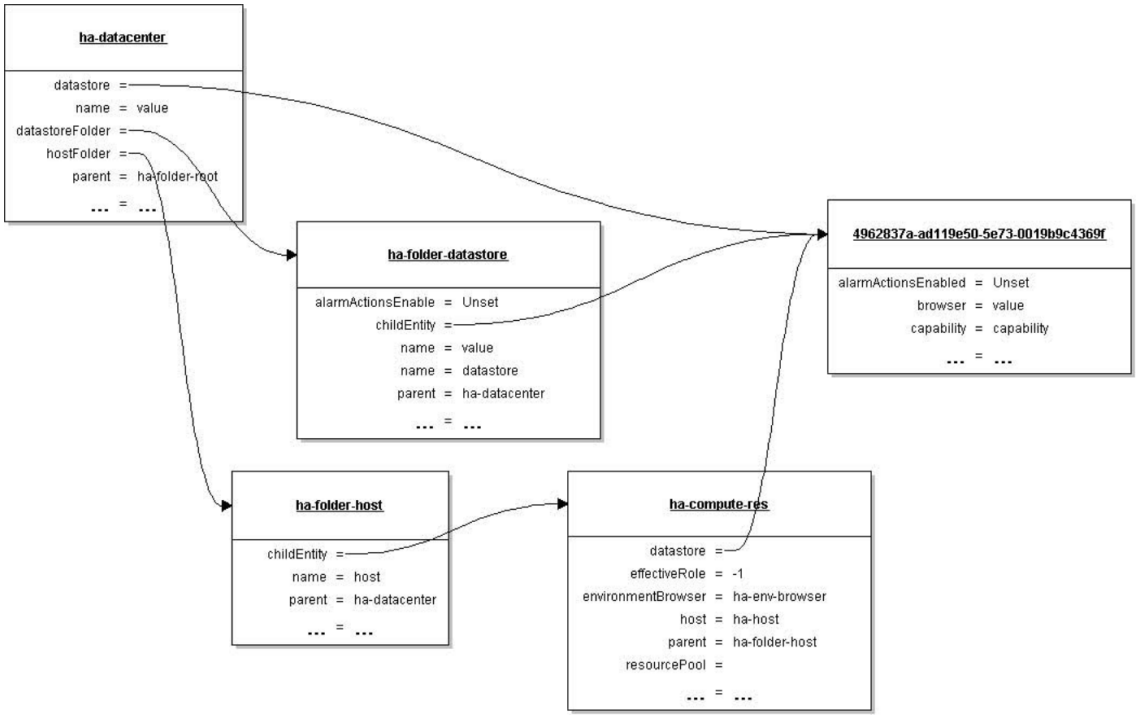
Accessing Datastores

The following diagram illustrates how you can access or specify datastores.

- Each **Datacenter** managed object has a **datastore** property that contains an array of datastores.
- Each **Datacenter** managed object has a **datastoreFolder** property that is a reference to the folder (or folder hierarchy) that contains the datastores for this datacenter.

- Each **`Datacenter`** managed object has a **`hostFolder`** property that is a reference to the folder (or folder hierarchy) that contains the compute resources, including hosts and clusters, for this datacenter. Each **`HostSystem`** or **`ComputeResource`** has a **`datastore`** property that is an array of **`Datastore`** managed objects.

Figure 28: Datastore Managed Object



For more information about the hierarchy of managed objects, see Datacenter Inventory.

Creating and Modifying a VMFS Datastore

A datastore is a manageable storage entity, usually used as a repository for virtual machine files including log files, scripts, configuration files, virtual disks, and so on.

VMFS is a proprietary file system VMware designed for virtual machines. VMFS is well suited for storing a small number of large data files like virtual disks. These files are mostly used by a single host. VMFS differs from other filesystem formats like FAT16/FAT32 and so on in that it can be accessed by multiple hosts connected to the same SAN LUN.

You can set up a VMFS datastore on any SCSI-based storage device that the host can access. VMFS volume creation, extension, and expansion requires first partitioning operations and the VMFS volume operations.

Set Up Disk Partitions

The Web Services API enables you to manage disk partitions on ESXi host storage devices.

To set up the disk partitions, use the following steps.

1. Call **`HostStorageSystem.RetrieveDiskPartitionInfo`** to retrieve information about existing partitions.
2. Call **`HostStorageSystem.ComputeDiskPartition`**, passing in the desired disk layout. The server computes a new partition information object for a specific disk representing the desired layout and returns a **`HostDiskPartitionInfo`** object that you can use in the **`HostDiskPartitionSpec`** you pass into **`UpdateDiskPartitions`**.
3. Call **`HostStorageSystem.UpdateDiskPartitions`** to update partitions by passing in a **`HostDiskPartitionSpec`**.

Create the VMFS Datastore

After you create disk partitions on host storage, you can configure VMFS datastores.

To create a VMFS datastore, use the following steps.

1. Configure and install any third-party adapter your storage requires and rescan the adapters by calling **`HostStorageSystem.RescanAllHba`**.
2. Call **`HostDatastoreSystem.QueryAvailableDisksForVmfs`** for information about disks that can be used to contain VMFS datastore.

   This method filters out disks that are currently in use by an existing VMFS unless the VMFS using the disk is one being extended. It will also filter out management LUNs and disks that are referenced by RDMs. These disk LUNs are also unsuited for use by a VMFS. The method returns an array of **`HostScisiDisk`** objects.

3. Call **`HostDatastoreSystem.QueryVmfsDatastoreCreateOptions`** for information about options for for creating a new VMFS datastore. The call returns an array of **`VmfsDatastoreCreateOption`** data objects that allow you to access the UUIDs of suitable data stores.
4. (Optional) If no suitable partitions for your VMFS volume exist, you might have to create them. Use the **`ComputeDiskPartitionInfo`** and **`UpdateDiskPartitions`** methods in **`HostStorageSystem`**.
5. Create the datastore.
   - Call **`HostDatastoreSystem.CreateVmfsDatastore`** to create a VMFS datastore. The method takes a **`VmfsDatastoreCreateSpec`** data object that consists of a a **`partition`**, a **`HostVmfsSpec`**, and an optional **`extent`**. The **`HostVmfsSpec`** allows you to specify the block size, extent, major version, and volume name for the VMFS.
   - Call **`HostDatastoreSystem.CreateNasDatastore`** to create a network-attached storage based datastore.

You can later expand and extend the VMFS datastore by calling one of the following methods.

- Call first **`QueryVmfsDatastoreExpandOptions`** and then **`ExpandVmfsDatastore`** to expand an existing VMFS datastore using the specification provided in the **`VmfsDatastoreExpandSpec`** data object (which contains the name of the extent and partition information). **`ExpandVmfsDatastore`** increases the size of the datastore up to the full size provisioned for the datastore, if necessary.
- Call first **`QueryVmfsDatastoreExtendOptions`** and then **`ExtendVmfsDatastore`** to extend an existing VMFS datastore using the specification provided in the **`VmfsDatastoreExtendSpec`** data object.

Removing and Updating Datastores

The Web Services API has methods to update and remove datastores.

- **`RemoveDatastore`** – Removes a datastore from a host.
- **`UpdateLocalSwapDatastore –`** Choose the **`localSwapDatastore`** for this host. Any change to this setting affects virtual machines that subsequently power on or resume from a suspended state at this host, or that migrate to this host while powered on. Virtual machines that are currently powered on at this host are not affected.

See the *`vSphere API Reference`* for more information about the **`HostDatastoreSystem`** operations, including constraints and limitations.

Managing VMFS Datastores with HostStorageSystem

In most cases, the **`Datastore`** methods are appropriate for creating and managing VMFS datastores. However, in some cases the following **`HostStorageSystem`** commands are used instead:

- **AttachVmfsExtent** – Extends a VMFS by attaching a disk partition as an extent.
- **ExpandVmfsExtent** – Expands a VMFS extent as specified by the disk partition specification.
- **FormatVmfs** – Formats a new VMFS on a disk partition based on the **HostVmfsSpec** that you pass in. Returns a **HostVmfsVolume** that represents the new VMFS file system. The **HostVmfsVolume** includes the block size, list of partition names of the disk's VMFS extents, and other information including the UUID.
  This command is a low-level API you can use to partition disks explicitly. In most cases, the **Datastore** VMFS commands are more suitable.
- **RescanVmfs** – Rescans for new VMFS instances.
- **UpgradeVmfs** – Upgrades the VMFS to the current VMFS version.

Update and Upgrade with HostStorageSystem

You can use the HostStorageSystem managed object to alter characteristics of storage units.

- **HostStorageSystem.UpdateScsiLunDisplayName** – Update the mutable display name associated with a SCSI LUN. The SCSI LUN to be updated is identified using the LUN UUID.
- **HostStorageSystem.UpgradeVmLayout** – Iterates over all registered virtual machines. For each virtual machine, upgrades the layout and logs an event. After the method has been called, the information in the **VirtualMachineFileLayout** data object data object is correct.

Managing VMFS Volume Copies (Resignaturing)

By default, ESX/ESXi hosts mount all VMFS datastores. Each VMFS datastore that is created in a partition on a LUN has a unique UUID that is stored in the file system superblock. In addition, the LUN ID of the source LUN is unique and is stored in the VMFS metadata.

When a LUN is replicated or a copy is made, the resulting LUN copy is identical, byte-for-byte, with the original LUN. As a result, if the original LUN contains a VMFS datastore with UUID X, the LUN copy appears to contain an identical VMFS datastore, or a VMFS datastore copy, with exactly the same UUID X. ESX/ESXi can determine whether a LUN contains the VMFS datastore copy, and considers the copy unresolved and does not mount it automatically.

To make the data on the LUN copy available, you can either force mount the copy if you are sure the original is not in use, or you can resignature the copy. When you perform datastore resignaturing, consider the following points:

- Datastore resignaturing is irreversible because it overwrites the original VMFS UUID.
- The LUN copy that contains the VMFS datastore that you resignature is no longer treated as a LUN copy, but instead appears as an independent datastore with no relation to the source of the copy.
- A spanned datastore can be resignatured only if all its extents are online.
- The resignaturing process is crash and fault tolerant. If the process is interrupted, you can resume it later.
- You can mount the new VMFS datastore without a risk of its UUID colliding with UUIDs of any other datastore, such as an ancestor or child in a hierarchy of LUN snapshots.

See the *vSphere Storage* documentation for additional information.

Resignaturing Volumes with ResignatureUnresolvedVmfsVolume_Task

The easiest way to resignature unresolved volumes is by using the **HostDatastoreSystem.ResignatureUnresolvedVmfsVolume_Task** method. The method assigns a new **DiskUuid** to a VMFS volume, but keep its contents intact. The method supports safe volume sharing across hosts and is appropriate in most cases.

You can instead use the low-level **HostStorageSystem** methods to find, force mount, or unmount unresolved volumes:

- **HostStorageSystem.QueryUnresolvedVmfsVolume** – Obtains the list of unbound VMFS volumes. For sharing a volume across hosts, a VMFS volume is bound to its underlying block device storage. When a low-level block copy is performed to copy or move the VMFS volume, the copied volume is unbound.
- **HostStorageSystem.ResolveMultipleUnresolvedVmfsVolumes** – Resignatures or force mounts unbound VMFS volumes. This method takes a **HostUnresolvedVmfsResolutionSpec** data object as input. The **HostUnresolvedVmfsResolutionSpec.resolutionSpec** property is an array of **HostUnresolvedVmfsResolutionSpec** data objects that contain a **HostUnresolvedVmfsResolutionSpecVmfsUuidResolution** enumeration. The enumeration is either **forceMount** or **resignature**.
- **UnmountForceMountedVmfsVolume** – Unmounts a force mounted VMFS volume. When a low-level block copy is performed to copy or move the VMFS volume, the copied volume is unresolved. For the VMFS volume to be usable, a resolution operation is applied. As part of resolution operation, you might decide to keep the original VMFS UUID. Once the resolution is applied, the VMFS volume is mounted on the host for its use. This method allows you to unmount the VMFS volume if it is not used by any registered virtual machines.

Managing Diagnostic Partitions

Your host must have a diagnostic partition (dump partition) to store core dumps for debugging and for use by VMware technical support.

For more information about diagnostic bundles, see Generating Diagnostic Bundles. For information about how to collect diagnostic partitions for a purple screen fault in ESXi, see the Broadcom knowledge base article at KB 1006796 .

A 100MB diagnostic partition for each host is recommended. If more than one ESX/ESXi host uses the same LUN as the diagnostic partition, that LUN must be zoned so that all the ESX/ESXi host can access it. Each host needs 100MB of space, so the size of the LUN determines how many servers can share it. Each ESX/ESXi host is mapped to a diagnostic slot. VMware recommends at least 16 slots (1600MB) of disk space if servers share a diagnostic partition. You can set up a SAN LUN with FibreChannel or hardware iSCSI. SAN LUNs accessed through a software iSCSI initiator are not supported.

> **CAUTION**
> If two hosts that share a diagnostic partition fail and save core dumps to the same slot, the core dumps might be lost. To collect core dump data, reboot a host and extract log files immediately after the host fails. If another host fails before you collect the diagnostic data of the first host, the second host does not save the core dump.

Retrieving Diagnostic Partition Information

The **HostDiagnosticSystem** managed object allows you to retrieve information in several ways.

- Retrieve the **HostDiagnosticPartition** object from the **HostDiagnosticSystem.activePartition** property to examine the properties of the active partition.
- Call the **HostDiagnosticPartition.QueryAvailablePartition** method to retrieve a list of available diagnostic partitions, in order of suitability.
- Call the **HostDiagnosticPartition.QueryPartitionCreateOptions** method to retrieve a list of disks with sufficient space to contain a diagnostic partition of the specified type. The choices are returned in order of suitability.

Create a Diagnostic Partition

Creating a diagnostic partition requires that you find a suitable partition using one of the query methods. You can then retrieve a creation specification, and perform the actual creation.

To create a diagnostic partition, use the following steps.

1. Find a suitable partition by calling **`HostDiagnosticPartition.QueryAvailablePartition`** or **`HostDiagnosticPartition.QueryPartitionCreateOptions`**.
2. Call **`HostDiagnosticPartition.CreateDiagnosticPartition`**, passing in a **`HostDiagnosticPartitionCreateSpec`**, which includes information about the diagnostic type, id, storage type, and so on.

On success, this method creates the partition and makes the partition the active partition if specified in the **`active`** parameter. On failure, the diagnostic partition might exist, but will not be active even if the partition was supposed to be made active.

Sample Code for Storage

The following table lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

Java (SDK\vsphere-ws\java\JAXWS\samples\com\vmware\)
scsilun\SCSILunName.java
httpfileaccess\GetVMFiles.java

vSphere Networks

Before you add storage and virtual machines to an ESXi system, you should have completed networking setup. This chapter describes how to set up virtual switches in the vSphere environment.

Virtual Switches

vSphere supports the use of virtual switches to manage network traffic to and from virtual machines.

- vCenter Server supports a distributed network model in which a distributed virtual switch manages ESXi host proxy switch configuration. In the distributed network model, a host proxy switch reflects the distributed virtual switch port settings, describes how physical network adapters are bridged to the switch, and performs network I/O.
- On a standalone ESXi host, you can use a VMware standard virtual switch to support network traffic to and from virtual machines on the host.

To configure a vSphere network you perform the following operations:

- Set up virtual switches
- Define portgroups
- Configure physical network adapters

Port Groups

Port groups aggregate multiple ports under a common configuration. Each port can connect to a network adapter of a virtual machine, or an uplink adapter on the physical machine.

Each port group is identified by a network label, which is unique to the current host. Network labels make virtual machine configuration portable across hosts. All port groups in a datacenter that are physically connected to the same network (in the sense that each can receive broadcasts from the others) are given the same label. Conversely, if two port groups cannot receive broadcasts from each other, they have distinct labels.

You can use a VLAN ID to restrict port group traffic to a logical Ethernet segment within the physical network. For a port group to reach port groups located on other VLANs, the VLAN ID must be set to 4095. If you use VLAN IDs, you must change the port group labels and VLAN IDs together so that the labels properly represent connectivity.

Virtual Machine Network Interfaces

When you create a virtual machine, you include a **`VirtualMachineConfigSpec`**, which, in turn, includes a **`VirtualDeviceConfigSpec`**. The **`device`** property of **`VirtualDeviceConfigSpec`** is a **`VirtualDevice`** data object.

One of the available virtual devices is **`VirtualEthernetCard`**. You can use one of the subtypes of **`VirtualEthernetCard`** to specify the virtual card to use and to specify the MAC address and whether wake-on-LAN is enabled for this virtual card. See Adding Devices to Virtual Machines. A limited number of adapters is supported. KB article 1001805 (http://kb.vmware.com/kb/1001805) discusses available network adapters and which adapter is appropriate in which situation.

VMkernel Network Interfaces

The network services that the VMkernel provides (iSCSI, NFS, and VMotion) use a TCP/IP stack in the VMkernel. This stack accesses various networks by attaching to one or more port groups on one or more virtual switches.

The VMware VMkernel TCP/IP networking stack handles iSCSI, NFS, and VMotion in the following ways.

- iSCSI as a virtual machine datastore
- iSCSI for the direct mounting of ISO files, which are presented as CD-ROMs to virtual machines
- NFS as a virtual machine datastore
- NFS for the direct mounting of ISO files, which are presented as CD-ROMs to virtual machines
- Migration with VMotion

If you have two or more physical NICs for iSCSI, you can create multiple paths for the software iSCSI by using port binding. For more information on port binding, see the *`iSCSI SAN Configuration Guide`*.

A freshly installed ESX/ESXi system does not include VMkernel network interfaces. When you wish to migrate a virtual machine with VMotion, your VMkernel networking stack must be set up properly. When you want to use storage types that use TCP/IP network communications, such as iSCSI, you must provide a separate VMkernel network interface for that storage device. You must create any VMkernel ports you might need (see Adding a VMkernel Network Interface).

Physical Network Adapter (pnic)

The term pnic refers to the physical network adapters as seen by the primary operating system. When using the vSphere Web Services SDK, you can manipulate the adapter directly. When using the vSphere Client GUI, you manipulate instead the uplink adapter. On an ESXi host, each pnic has one associated uplink adapter.

In a vDS environment, you use a DVS uplink instead of an uplink adapter.

Using a Distributed Virtual Switch

A **`DistributedVirtualSwitch`** managed object is a virtual network switch that is located on a vCenter Server. A distributed virtual switch manages configuration for proxy switches (**`HostProxySwitch`**). A proxy switch is located on an ESXi host that is managed by the vCenter Server and is a member of the switch. A distributed switch also provides virtual port state management so that port state is maintained when vCenter Server operations move a virtual machine from one host to another.

A proxy switch performs network I/O to support the following network traffic and operations:

- Network traffic between virtual machines on any hosts that are members of the distributed virtual switch.
- Network traffic between a virtual machine that uses a distributed virtual switch and a virtual machine that uses a VMware standard virtual switch.
- Network traffic between a virtual machine and a remote system on a physical network connected to the ESXi host.
- vSphere system operations to support capabilities such as VMotion or High Availability.

A **DistributedVirtualSwitch** is the base distributed switch implementation. It supports a VMware distributed virtual switch implementation and it supports third party distributed switch implementations. The base implementation provides the following capabilities (defined in the **DVSFeatureCapability** object):

- NIC teaming
- Network I/O control
- Network resource allocation
- Quality of service tag support
- User-defined resource pools
- I/O passthrough (VMDirectPath Gen2)

A **VmwareDistributedVirtualSwitch** supports the following additional capabilities (defined in the **DVSFeatureCapability** and **VMwareDVSFeatureCapability** objects):

- Backup, restore, and rollback for a VMware distributed virtual switch and its associated portgroups.
- Maximum Transmission Unit (MTU) configuration.
- Health check operations for NIC teaming and VLAN/MTU support.
- Monitoring switch traffic using Internet Protocol Flow Information Export (IPFIX).
- Link Layer Discovery Protocol (LLDP).
- Virtual network segmentation using a Private VLAN (PVLAN).
- VLAN-based SPAN (VSPAN) for virtual distributed port mirroring.
- Link Aggregation Control Protocol (LACP) defined for uplink portgroups.

Distributed Virtual Switch Configuration

To use a distributed virtual switch, you create a switch and portgroups on a vCenter Server, and add hosts as members of the switch.

1. Use the **Folder.CreateDVS_Task** method to create a distributed virtual switch. Use a **DVSConfigSpec** to create a switch for a third-party implementation. Use a **VMwareDVSConfigSpec** to create a VMware distributed virtual switch.
2. Use the **CreateDVPortgroup_Task** method to create portgroups for host and virtual machine network connections and for the connection between proxy switches and physical NICs. A **DistributedVirtualPortgroup** specifies how virtual ports (**DistributedVirtualPort**) will be used. When you create a distributed virtual switch, the vCenter Server automatically creates one uplink portgroup (**config.uplinkPortgroup**). Uplink portgroups are distributed virtual portgroups that support the connection between proxy switches and physical NICs.

   Port creation on a distributed switch is determined by the portgroup type (`DVPortgroupConfigSpec.type`):

   - If a portgroup is early binding (static), then **DVPortgroupConfigSpec.numPorts** determines the number of ports that get created when the portgroup is created. This number can be increased if **DVPortgroupConfigSpec.autoExpand** is true.
   - If a portgroup is ephemeral (dynamic), then `numPorts` is ignored and ports are created as needed. You can also specify standalone ports that are not associated with a port group and uplink ports that are created on ESXi hosts (**DVSConfigSpec.numStandalonePorts**).

     The **DVPortgroupConfigInfo.numPorts** property is the total number of ports for a distributed virtual switch. This total includes the ports generated by the static and dynamic portgroups and the standalone ports.
3. If you have created additional uplink portgroups, use the `ReconfigureDvs_Task` method to add the portgroup(s) to the **DVSConfigSpec.uplinkPortgroup** array.
4. Retrieve physical NIC device names from the host (**HostSystem.config.network.pnic[].device**).
5. Add host member(s) to the distributed virtual switch. To configure host members:
   - Specify hosts (**DVSConfigSpec.host[]**).

- For each host, specify one or more physical NIC device names to identify the pNIC(s) for the host proxy connection to the network (**`DistributedVirtualSwitchHostMemberConfigSpec.backing.pnicSpec[].pnicDevice`**)
- Use the `DistributedVirtualSwitch.ReconfigureDvs_Task` method to update the switch configuration.
  When you add a host to a distributed virtual switch (`DistributedVirtualSwitch.config.host`), the host automatically creates a proxy switch. The proxy switch is removed automatically when the host is removed from the distributed virtual switch.

6. Connect hosts and virtual machines to the distributed virtual switch.

Host connection	Specify port or portgroup connections in the host virtual NIC spec (**`HostVirtualNicSpec.distributedVirtualPort`** or **`HostVirtualNicSpec.portgroup`**).
Virtual machine connection	Specify port or portgroup connections in the distributed virtual port backing (**`VirtualEthernetCardDistributedVirtualPortBackingInfo`**) for the virtual Ethernet cards on the virtual machine (**`VirtualEthernetCard.backing`**).

Backup, Rollback, and Query Operations

If you are using a **`VmwareDistributedVirtualSwitch`**, you can perform backup and rollback operations on the switch and its associated distributed virtual portgroups.

When you reconfigure a VMware distributed virtual switch (**`ReconfigureDvs_Task`**), the server saves the current switch configuration before applying the configuration updates. The saved switch configuration includes portgroup configuration data. The server uses the saved switch configuration as a checkpoint for rollback operations. You can rollback the switch or portgroup configuration to the saved configuration, or you can rollback to a backup configuration (**`EntityBackupConfig`**).

- To backup the switch and portgroup configuration, use the **`DistributedVirtualSwitchManager.DVSManagerExportEntity_Task`** method. The export method produces a **`EntityBackupConfig`** object. The backup configuration contains the switch and/or portgroups specified in the `SelectionSet` parameter. To backup the complete configuration you must select the distributed virtual switch and all of its portgroups.
- To rollback the switch configuration, use the **`DVSRollback_Task`** method to determine if the switch configuration has changed. If it has changed, use the **`ReconfigureDvs_Task`** method to complete the rollback operation.
- To rollback the portgroup configuration, use the **`DistributedVirtualPortgroup.DVPortgroupRollback_Task`** method to determine if the portgroup configuration has changed. If it has changed, use the **`ReconfigureDVPortgroup_Task`** method to complete the rollback operation.

To perform query operations on a distributed virtual switch, use the **`DistributedVirtualSwitchManager`** methods.

VMware Standard Virtual Switch

Network setup for ESXi hosts can consist of several parts:

- Setting up one or more virtual switches. Virtual switches provide the connectivity between virtual machines on the same host or on different hosts. Virtual switches also support VMkernel network access for VMotion, iSCSI, and NFS. You set up virtual switches independently on each host. See Adding a Standard Virtual Switch.
- Adding virtual machine port groups. A virtual machine always accesses the network through a port group. See Adding a Virtual Port Group.
- Specifying the adapter for the virtual machine. This adapter is specified as a virtual device, configured as part of virtual machine setup, and discussed in Configuring a Virtual Machine.

- Adding VMkernel network interfaces, for example, to support iSCSI storage or VMotion. See Adding a VMkernel Network Interface.
- Configuring a physical adapter (pnic), the actual connection from the host to the network. You can configure the pnic through the `HostNetworkSystem.pnic` property, which is a **`PhysicalNic`** data object. You can specify the set of pnics associated with a virtual switch through the `VirtualSwitch.pnic` property, which takes an array of physical network adapters.
- Network configuration for the host (IP routing, DNS, SNMP). See Adding Networking Services.

Configuring a Standard Virtual Switch

To use a VMware standard virtual switch, you use the following elements to configure the switch on an ESXi host.

- **`HostNetworkSystem`** – Managed object that represents the host's networking configuration. This object's properties point to the networking data objects you can use for network management, including **`HostDnsConfig`** and **`HostIpRouteConfig`**.
  `HostNetworkSystem` properties allow you to access **`HostNetCapabilities`** and **`HostNetworkInfo`** data objects, and access and modify the **`HostNetworkConfig`** data object.

  `HostNetworkSystem` includes methods for retrieving and changing the network configuration. See the *`API Reference`* for a complete list of methods and the permissions required to run them.
- **`HostNetworkConfig`** – Allows you to specify the network configuration for the host. You can apply the configuration by running the **`HostNetworkSystem.UpdateNetworkConfig`** method.
- **`Network`** – Represents a network accessible by either hosts or virtual machines. This can be a physical network or a logical network, such as a VLAN.
  When you add a host to a vCenter Server system, or when you add a virtual machine to an ESX/ESXi host, a **`Network`** is added automatically.
- **`HostSystem.QueryHostConnectionInfo`** and **`Datacenter.QueryConnectionInfo`** both return a **`HostConnectInfo`** data object, which describes the current network configuration.

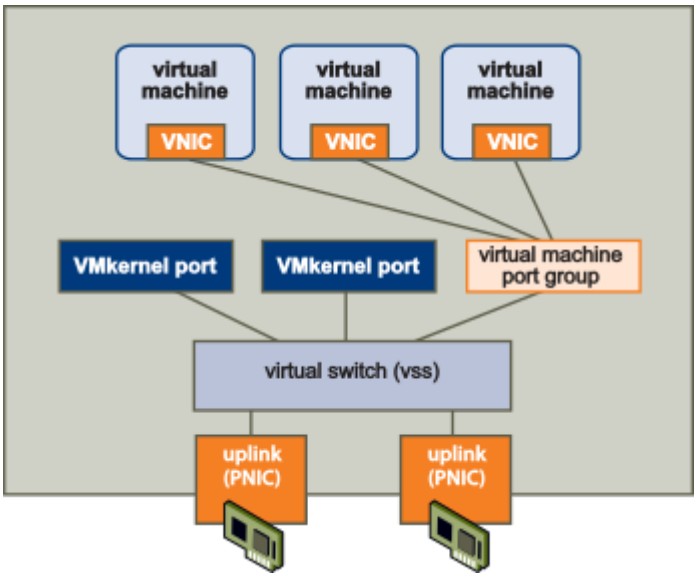
`HostSNMPSystem` – Supports SNMP setup. See Setting Up SNMP.

vNetwork Standard Switch Environment

A vNetwork Standard Switch (vSS) can route traffic internally between virtual machines and can link virtual machines to external networks.

The following diagram shows the elements of a vSS environment.

Figure 29: vSS Environment



Virtual Switches

At the center of networking with vSS is the virtual switch itself. The vSS can send network traffic between virtual machines on the same host (private network) or network traffic to an external network (public network). The public network uses the Ethernet adapter associated with the physical host (uplink adapter).

When two or more virtual machines are connected to the same vSS, network traffic between them is routed locally. If an uplink adapter is attached to the vSS, each virtual machine can access the external network that the adapter is connected to.

Setting Up Networking with vSS

You can use the **`HostNetworkSystem`** managed object to access and manipulate the elements of an ESXi network.

Retrieving Information About the Network Configuration

You can retrieve information about the network configuration as follows:

- The properties of the **`HostNetworkConfig`** object, which you access through **`HostNetworkSystem.networkConfig`**, allow you to retrieve configuration information. This information is comprehensive and includes the physical adapters, virtual switches, virtual network interfaces, and so on. You can also use **`HostNetworkConfig`** to make changes to the configuration.
- The properties of the **`HostNetworkInfo`** object, which you access through **`HostNetworkSystem.networkInfo`**, allow you to retrieve runtime information.

Adding a Standard Virtual Switch

You call the **`HostNetworkSystem.AddVirtualSwitch`** method to add one or more virtual switches. Pass in the name of the virtual switch and a **`HostVirtualSwitchSpec`** data object as parameters.

Inside **`HostVirtualSwitchSpec`** you can specify the MTU, number of ports, network policy, and bridge specification. The bridge specifies how the virtual switch connects to the physical adapter. The currently supported bond bridge provides network adapter (NIC) teaming capabilities through the use of a list of physical devices and, optionally, a beacon probe to test connectivity with physical adapters.

After you have created the virtual switch, you can connect it to a pnic for connection to the outside, and to a VMkernel port or a port group.
To add a virtual switch, use the following steps.

1. Obtain information about the current networking configuration.

   You can use a property collector to retrieve the **`HostNetworkSystem`** managed object and several of its properties, such as **`networkInfo`**.

2. Define a **`HostVirtualSwitchSpec`** that specifies the attributes of the virtual switch. You can specify the number of ports (56 to 4088 on ESXi systems) and the **`HostNetworkPolicy`**. See “Defining the Host Network Policies” on page 122.
3. Call **`HostNetworkSystem.AddVirtualSwitch`** to add a virtual switch. Specify a unique name and a **`HostVirtualSwitchSpec`** that defines the switch attributes.

   The following fragment from `AddVirtualSwitch.java` illustrates this.

Adding a Virtual Switch

```
vswitchId = vSwitch42;

...

ManagedObjectReference nwSystem = configMgr.getNetworkSystem();

HostVirtualSwitchSpec spec = new HostVirtualSwitchSpec();

spec.setNumPorts(8);

service.addVirtualSwitch(nwSystem, vswitchId, spec);

System.out.println( " : Successful creating : " + vswitchId);
```

Adding a Virtual Port Group

Port groups allow you to differentiate between different kinds of traffic passing through a virtual switch. You can also use port groups as a boundary for communication or for security policy configuration. The default port groups for ESXi systems are **`Management Network`** and **`VM Network`**.

When you create a port group, you can specify a VLAN ID for it. VLANs are an important part of ESXi networking because they allow you to group traffic. For example, you could create separate network segments for VMotion, for management and for development. Using VLANS, you only need to have a separate uplink adapter for each network segment and a single virtual switch connecting to that adapter. That setup can greatly reduce the number of switches you need.
To add a virtual port group, use the following steps.

1. Define a **`HostPortgroupSpec`**. For each port group, you can specify the network policy, the VLAN ID, and the virtual switch to which the port group belongs.
2. Call **`HostNetworkSystem.AddPortGroup`**, passing in the **`PortGroupSpec`**.

Adding a VMkernel Network Interface

VMkernel network interfaces provide the network access for the VMkernel TCP/IP stack. You must create new VMkernel ports for your ESXi system if you plan on using VMotion, VMware FT, or iSCSI and NAS storage. A VMkernel port consists of a port on the virtual switch and a VMkernel interface.

To add a VMkernel Network Interface to your ESXi system, use the following steps.

1. Create a **HostVirtualNicSpec** data object. Inside the object, you can specify the IP configuration in a **HostIpConfig** data object. For vSS, specify the **portgroup** property. For vDS, specify the **distributedVirtualPort** property.
2. Call **HostNetworkSystem.AddVirtualNic**, passing in the **HostVirtualNicSpec**.
3. You can then use the VMkernel network interface for software iSCSI or NAS, or call the **HostVmotionSystem.SelectVnic** method to use this VMkernel NIC for VMotion.

   Adding a VMkernel Network Interface, a code fragment from the **AddVirtualNic** example, illustrates this. The sample retrieves the IP address from the command line using the **cb.get_option** call.

Adding a VMkernel Network Interface

```
private HostVirtualNicSpec createVNicSpecification() {

     HostVirtualNicSpec vNicSpec = new HostVirtualNicSpec();

     HostIpConfig ipConfig = new HostIpConfig();

     ipConfig.setDhcp(false);

     ipAddr = cb.get_option("ipaddress");

     ipConfig.setIpAddress(ipAddr);

     ipConfig.setSubnetMask("255.255.255.0");

     vNicSpec.setIp(ipConfig);

     return vNicSpec;

}

     ....


HostVirtualNicSpec vNicSpec = createVNicSpecification();

service.addVirtualNic(nwSystem, portGroup, vNicSpec);
```

Defining the Host Network Policies

When you configure host networks, you can define specific policies for the network. The **HostNetworkPolicy** data object type describes network policies for both virtual switches and port groups. If the settings are not specified for the port group explicitly, the port group inherits policy settings from the virtual switch with which it is associated.

The policies are defined by the following data objects available as properties of **HostNetworkPolicy**.

- **HostNicTeamingPolicy** – Defines the connection to the physical network. This includes failure criteria, active and standby NICs, and failover and load balancing information. See NIC Teaming.
- **HostNetworkSecurityPolicy** – Defines the security policies for the network. See the *ESXi Configuration Guide*.
- **HostNetworkTrafficShapingPolicy** – Establishes parameters for three traffic characteristics: average bandwidth, peak bandwidth, and maximum burst size.

You can also specify the VLAN policy by assigning an integer to the **HostPortgroupSpec.vlanid** property. The VMkernel takes care of tagging and untagging the packets as they pass through the virtual switch. See the **HostPortgroupSpec** and **HostNetworkPolicy** data objects in the *API Reference*.

NIC Teaming

Virtual machines connect to the public network through a virtual switch, which, in turn, connects to the physical network interface (pnic). When the physical adapter or the adapter's network connection fails, connectivity for the associated virtual switch and all port groups and virtual machines is lost.

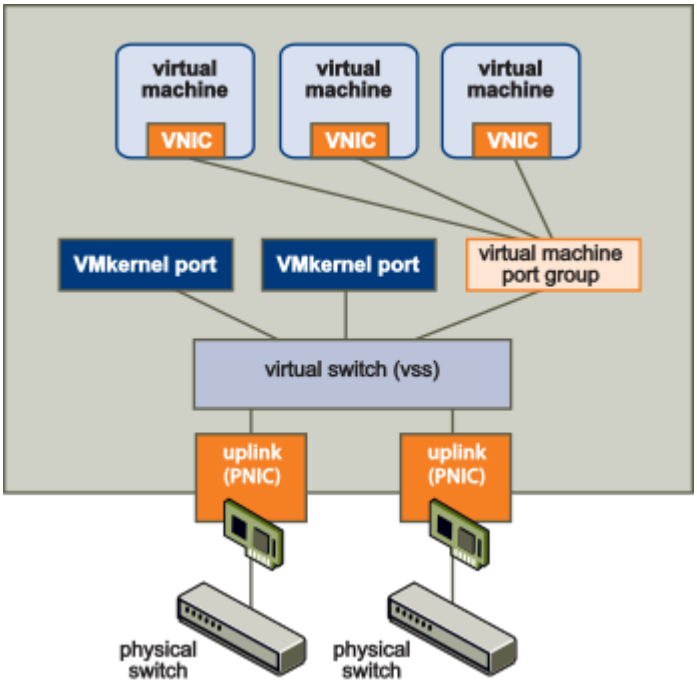
To resolve this issue, you can set up your environment so each virtual switch connects to two uplink adapters. Each uplink adapter connects to two different physical switches. The teams can then either share the load of traffic between physical and virtual networks among some or all of its members, or provide passive failover in the event of a hardware failure or a network outage.

You set up NIC teaming by setting the **`HostNetworkPolicy`**. The path to the **`HostNicTeamingPolicy`** is:

`HostConfigSpec.network.vswitch[].spec.policy.nicTeaming`

If you specify NIC teaming for a virtual switch, the **`HostVirtualSwitchSpec.bridge`** property must be set to **`HostVirtualSwitchBondBridge`**.

Figure 30: NIC Teaming



Setting Up IPv6 Networking

vSphere supports both Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6) environments. With IPv6, you can use vSphere features such as NFS in an IPv6 environment.

An IPv6-specific configuration in vSphere involves providing IPv6 addresses, either by entering static addresses or by using DHCP for all relevant vSphere networking interfaces. IPv6 addresses can also be configured using stateless autoconfiguration sent by router advertisement.

You can set up IPv6 networking for a host by changing the **`HostIpConfig.ipV6Config`** property, which is a **`HostIpConfigIpV6AddressConfiguration`** data object. **`HostIpConfigIpV6AddressConfiguration`** allows you to specify whether auto-configuration is enabled, whether DHCP for ipV6 addresses is enabled, and an array of IPv6 addresses (**`HostIpConfigIpV6Address`** data objects).

HostIpConfigIpV6Address allows you to specify all aspects of the IPv6 address including the state of the address, the address (unless DHCP is enabled), life time, operation, origin, and prefix length. See the *API Reference*. The following code fragment illustrates setting the VMkernel NIC to get an automatic IPv6 address from router advertisements and through DHCP. The user provides the IP address on the command line when calling the program from which the fragment is taken. The sample retrieves the address using the **cb.get_option** utility applications call.

IPv6 Setup

```
private HostVirtualNicSpec createVNicSpecification() {

     HostVirtualNicSpec vNicSpec = new HostVirtualNicSpec();

     HostIpConfig ipConfig = new HostIpConfig();


     //setting the vnic to get an automatic ipv6 address from router advertisements

     // and through dhcp


     ipV6Config = new HostIpConfigIpV6AddressConfiguration();

     ipV6Config.setAutoConfigurationEnabled(true);

     ipV6Config.setDhcpV6Enabled(true);

     ipConfig.setIpV6Config(ipV6Config);

     vNicSpec.setIp(ipConfig);

     return vNicSpec;

....
```

Adding Networking Services

You can set up network services for your ESXi system by using **HostConfigManager** properties and methods.

Adding an NTP Service

The **HostConfigManager.dateTimeSystem** property contains a **HostDateTimeSystem** data object. This object allows you to perform NTP and date and time related configuration.

- Query and update the date and time information by using one of the methods defined in **HostDateTimeSystem**.
- Modify the **HostDateTimeSystem.dateTimeInfo** property, which contains a **HostDateTimeInfo** object, to set up NTP. The NTP information is stored in the **HostDateTimeInfo.ntpConfig** property, which is a **HostNtpConfig** object. The **HostNtpConfig** objects's **server** property contains a list of time servers, specified by IP address or fully qualified domain name.

  IMPORTANT
  You can start and stop the NTP daemon and retrieve information about it by using the **HostServiceSystem** object.

Setting Up the IP Route Configuration

You can use the **HostNetworkSystem.UpdateIPRouteConfig** method to specify the IP route configuration for an ESXi system. The method takes a **HostIPRouteConfig** data object as an argument. In this object, you can specify the default gateway address and the IPv6 gateway address.

Setting Up SNMP

Simple Network Management Protocol (SNMP) allows management programs to monitor and control networked devices. vCenter Server and ESXi systems include different SNMP agents:

- The SNMP agent included with vCenter Server can send traps when the vCenter Server system is started or when an alarm is triggered on vCenter Server. The vCenter Server SNMP agent functions only as a trap emitter and does not support other SNMP operations such as **GET**.
- ESXi 4.0 includes an SNMP agent embedded in the host daemon (**hostd**) that can send traps and receive polling requests such as **GET** requests.

Versions of ESX released before ESX/ESXi 4.0 included a Net-SNMP-based agent. You can continue to use this Net-SNMP-based agent in ESX 4.x with MIBs supplied by your hardware vendor and other third-party management applications. However, to use the VMware MIB files, you must use the embedded SNMP agent. To use the NET-SNMP based agent and embedded SNMP agent at the same time, make one of the agents listen on a nondefault port. By default, both agents use the same port.

The SDK supports SNMP agent configuration through the **HostSnmpSystem** managed object. This object includes two methods, **ReconfigureSnmpAgent** and **SendTestNotification**.

- **HostSnmpSystem.ReconfigureSnmpAgent** allows you to specify agent properties through a **HostSnmpConfigSpec**. That data object allows you to specify the SNMP port, read only communities, and the trap targets in an **HostSnmpDestination** object. The **HostSnmpDestination** object allows you to specify the community, and a host and port listening for notification.
- **HostSnmpSystem.SendTestNotification** allows you to test your configuration.

A **HostSnmpSystemAgentLimits** data object in the **HostSnmpSystem.limits** property specifies limits of the agent.

Sample Code for Networks

The following table lists the sample applications included with the vSphere SDK that demonstrate how to use some of the managed objects discussed in this chapter.

Java
(SDK\vsphere-ws\java\JAXWS\samples\com\vmware\host)
AddVirtualNic.java
AddVirtualSwitch.java
AddVirtualSwitchPortGroup.java
RemoveVirtualNic.java
RemoveVirtualSwitch.java
RemoveVirtualSwitchPortGroup.java

Virtual Machine Configuration

A virtual machine is a software computer that, like a physical computer, runs an operating system and applications. Virtual machines are compatible with all standard x86 computers. Each virtual machine encapsulates a complete computing environment and runs independently of underlying hardware.

VirtualMachine Management Objects and Methods

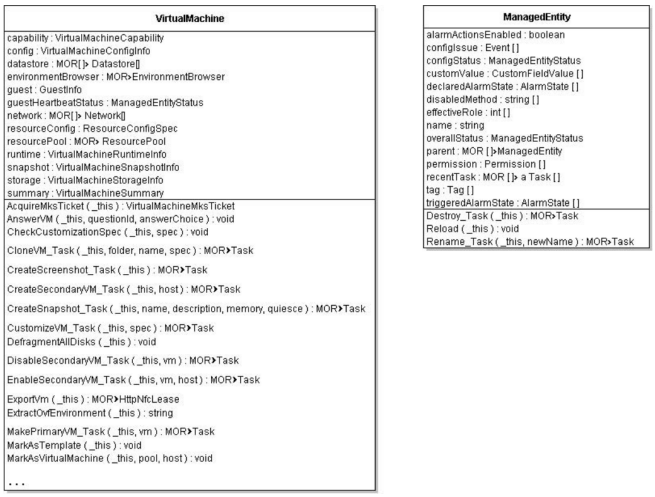
Virtual machines are the central elements of your vSphere environment.

You create a virtual machine by calling **`Folder.CreateVM_Task`**, and configure the virtual machine by using properties and methods of the **`VirtualMachine`** managed object. Most of the properties point to data objects that the methods use as input. Figure: VirtualMachine Managed Object with Some Properties and Methods shows some of the properties and methods.

Client applications commonly access and manipulate the following virtual machine related objects:

- **`VirtualMachine`** – Managed object used for most virtual machine manipulation. Includes methods that create templates, clones, or snapshots of a virtual machine, perform power operations and guest OS management, and install VMware Tools.
- **`VirtualMachineConfigInfo`** – Data object which allows you to retrieve configuration-specific information from a virtual machine.
- **`VirtualMachineCloneSpec`** – Data object which allows you to specify virtual machine properties for a clone operation. Argument to **`VirtualMachine.CloneVM_Task`**.

Figure 31: VirtualMachine Managed Object with Some Properties and Methods



Creating Virtual Machines and Virtual Machine Templates

To create a virtual machine, you use the **`Folder.CreateVM_Task`** method. The method takes a **`VirtualMachineConfigSpec`** data object as input argument. **`VirtualMachineConfigSpec`** allows you to specify the attributes of the virtual machine you are creating.

If you need several identical virtual machines, you can convert an existing virtual machine to a template and create multiple copies (clones) from the template. You can also create multiple virtual machines by cloning an existing virtual machine directly.

Creating a Virtual Machine Using VirtualMachineConfigSpec

Use the **`Folder.CreateVM_Task`** method to create a virtual machine by specifying its attributes. You must specify either a host or a resource pool (or both). The virtual machine uses the CPU and memory resources from the host or resource pool.

Calling the CreateVM_Task Method

Create a virtual machine by calling the **`Folder.CreateVM_Task`** method with the following arguments:

- **`_this`** —Folder where you want to place the virtual machine.
- **`config`**— **`VirtualMachineConfigSpec`** data object that specifies CPU, memory, networking, and so on. See Specifying Virtual Machine Attributes with VirtualMachineConfigSpec)
- **`pool`**— Resource pool for the virtual machine to draw resources from.
- **`host`**— **`HostSystem`** managed object that represents the target host on which to run the virtual machine. If you invoke this method on a standalone host, omit this parameter. If the target host is part of a VMware DRS cluster, this parameter is optional; if no host is specified, the system selects one.

  IMPORTANT
  All objects must be located in the same datacenter.

Specifying Virtual Machine Attributes with VirtualMachineConfigSpec

The actual customization of the virtual machine happens through the properties of the **`VirtualMachineConfigSpec`** that is passed in as an argument to **`Folder.CreateVM_Task`**. For example, you can specify the name, boot options, number of CPUs, and memory for the virtual machine. All properties of **`VirtualMachineConfigSpec`** are optional to support incremental changes. See the *API Reference*.

The following example fragment from the **`VMCreate`** sample program illustrates how to define a **`VirtualMachineConfigSpec`**.

Defining a VirtualMachineConfigSpec Data Object

```
VirtualMachineConfigSpec vmConfigSpec = new VirtualMachineConfigSpec();

     ...

     vmConfigSpec.setName("MyVM");

     vmConfigSpec.setMemoryMB(new Long(Integer.parseInt 500));

     vmConfigSpec.setNumCPUs(Integer.parseInt 4);

     vmConfigSpec.setGuestId(cb.get_option("guestosid"));

     ...
```

The VMware SDK `SDK/samples/Axis/java/com/vmware/apputils/vim/VMUtils.java` sample defines a more comprehensive virtual machine that also includes a Floppy, CD-ROM, disk, and virtual NIC. See Configuring a Virtual Machine for a discussion of commonly set properties.

When you create a virtual machine, the virtual machine files are added at the virtual machine's storage location. See Table : Default Files.

Additional Configuration Information

The **`VirtualMachineConfigInfo`** and **`VirtualMachineConfigSpec`** objects provide the **`extraConfig`** property for additional configuration information. The **`extraConfig`** property is an array of key/value pairs that identify configuration options. The Server stores the **`extraConfig`** options in the `.vmx` file for the virtual machine. As the vSphere API evolves from version to version, an extraConfig option may become a standard configuration property that is part of the defined inventory data model. In this case, you must use the standard data model property for access; you cannot use the **`extraConfig`** property.

Cloning a Virtual Machine

A clone is a copy of a virtual machine. The main difference between a virtual machine and a clone is that the **`VirtualMachine.config.template`** property is set to **`true`**.

You can create a clone in one of the following ways:

- If you no longer need a specific instance of a virtual machine, but you want to use the virtual machine's configuration as a template, use the **`VirtualMachine.MarkAsTemplate`** method. This method sets the **`config.template`** property to **`true`**, and disables the virtual machine.
- If you want to use an existing virtual machine as a template, but keep the virtual machine, call the **`VirtualMachine.CloneVM_Task`** method to create a duplicate of the virtual machine.

If you use the **`VirtualMachine.CloneVM_Task`** method, you can customize certain attributes of the clone by specifying them in the **`VirtualMachineCloneSpec`** data object you pass in when you call the method.

The following code fragment from `VMClone.java` illustrates how you can customize a clone and specify a new location for it.

Cloning a Virtual Machine

```
VirtualMachineCloneSpec cloneSpec = new VirtualMachineCloneSpec();

VirtualMachineRelocateSpec relocSpec = new VirtualMachineRelocateSpec();

cloneSpec.setLocation(relocSpec);

cloneSpec.setPowerOn(false);

cloneSpec.setTemplate(false);


String clonedName = cloneName;


ManagedObjectReference cloneTask

            = service.cloneVM_Task(vmRef, vmFolderRef, clonedName, cloneSpec);
```

The **`VirtualMachine.CloneVM_Task`** method takes the source virtual machine, target folder, name, and **`VirtualMachineCloneSpec`** as arguments.

The **`VirtualMachineCloneSpec`** data object includes the location, power state, and whether the clone should be a template. The location, in turn, is a **`VirtualMachineRelocateSpec`** data object that specifies the target location (datastore, disk, and host or resource pool) and any transformation to be performed on the disk.

Continuous Virtual Machine Clone

Clone is one of the long-running virtual machine operations. It can interfere with or be interrupted by server reboot. As of vSphere 8.0 U3, continuous VM clone promises to streamline maintenance mode operations.

Continuous virtual machine clone ensures that:

- VM operation automatically restarts without customer intervention after vCenter restarts.
- From a user perspective, restarted operation is the same as initially requested. Customers can use existing scripts and interfaces to manage the operation.
- Incoming operations are not affected while the system is recovering interrupted operation.

Going forward, continuous virtual machine clone is the default, but can be deactivated with a `vpxd` configuration option. If instructed by technical support personnel for example, administrators can set `config.provisioning.continuousop.enabled` to `false` in the `vpxd.cfg` configuration file. This can be done with the vSphere Client; see "Configure Advanced Settings" in VMware docs.

Here are the steps for continuous operation. Pre-workflow (1-3) operation was interrupted before the workflow was executed. Mid-workflow (4-5): operation was interrupted in the middle of workflow execution. Post-workflow (6): operation was interrupted when workflow finished but before task was updated.

1. Session preservator saves user session. Task management associates the task with returned UUID corresponding to the preserved session.
2. Session preservator acquires a persistent token (or other equivalents) to preserve the identity.
3. Workflow is journalled in `VPX_JOURNAL` and associated tables. The first action in workflow is `RedoableAction`, which is responsible for restarting the operation during recovery. Transaction manager loads workflow journals from database. For each pending continuous operation, it flags corresponding managed entities in recovery so that the entities do not participate in conflicting operations while recovery is in progress.
4. Restart begins. Recovery long-running operation revives persisted session. The session becomes the current user session for later privilege checks. Recovery cleans up previous operation that was interrupted, publishing any cleanup issues encountered in events or tasks.
5. `RedoableAction` checks if the operation should be restarted. If ye,s it will continue to the rest of recovery, otherwise it sets the task error status. Conditions when an operation should be restarted are an implementation detail, for example, it might not restart if the maximum number of retries was exceeded.
6. If the interruption occurs right before task status is updated, the situation is handled by performing the workflow commit and task update in a single transaction.

As implemented, managed object `RedoableAction` extends `UndoableAction`. In the vSphere Web Services API Reference, see data object `TaskInfo` for details about continuous operations and `VM_CLONE_CONTINUOUS`.

Creating Virtual Machine Templates

Templates allow you to create multiple virtual machines with the same characteristics, such as resources allocated to CPU and memory, or type of virtual hardware. A virtual machine template is a virtual machine that cannot be powered on and that is not associated with a resource pool.

You can convert any powered off virtual machine to a template by calling **`VirtualMachine.MarkAsTemplate`**. After the conversion, the original virtual machine no longer exists. You can use the template to create multiple clones of the same configuration.

Converting a Template to a Virtual Machine

You can change a template back to an operational virtual machine.

- To convert the template to a virtual machine, call the **`MarkAsVirtualMachine`** method on the template. You must specify a resource pool and, optionally, a host for the virtual machine. Host and resource pool must be under the same **`ComputeResource`**. When the operation completes, the template no longer exists.
- To keep the template, clone the template by calling the **`CloneVM_Task`** method on the template. In the **`VirtualMachineCloneSpec`** (the **`spec`** parameter), set the **`template`** property to **`false`**.

Accessing Information About a Virtual Machine

After you have created a virtual machine, you can retrieve information about the virtual machine through the **`VirtualMachineConfigInfo`** properties. See the *`API Reference`* for a complete list.

Checking Default Files

After you have created a virtual machine, several files are generated and placed in the directory specified in the **`VirtualMachineConfigSpec.files`** property.

Table 4: Default Files

File	Usage	File Description	File Format
.vmx	*. vmname*.vmx	Virtual machine configuration file.	ASCII
.vmxf	*vmname*.**vmxf**	Additional virtual machine configuration files, available, for example, with teamed virtual machines.	ASCII
.vmdk	*vmname* .vmdk	Virtual disk file.	ASCII
.flat.vmdk	*vmname* .flat.vmdk	Preallocated virtual disk in binary format.	Binary
.vswp	*vmname* .vswp	Swap file.	
.nvram	*vmname*.**nvram** or **nvram**	Non-volatile RAM. Stores virtual machine BIOS information.	
.vmss	*vmname* .vmss	Virtual machine suspend file.	
.log	vmware.log	Virtual machine log file.	ASCII
#.log	vmware-#.log	Old virtual machine log files. # is a number starting with 1.	ASCII
.vmtx	*vmname*.vmtx	Virtual machine template file.	ASCII

If you are using snapshots, the following additional files might be available. See Snapshots.

File Extension	Usage	File Description
.vmsd	*vmname*.**vmsd**	Virtual machine snapshot file.
.vmsn	*vmname*.**vmsn**	Virtual machine snapshot data file.
**.delta.vmdk		Snapshot difference file. A number preceding the extension increases with more snapshots.
**.vmdk		Metadata about a snapshot.
-Snapshot#.vmsn		Snapshot of virtual machine memory. Snapshot size is equal to the size of you virtual machine's maximum memory.

Checking Default Devices

When you create a virtual machine, you are also creating a set of default devices, based on the hardware version associated with your SDK. You can see these devices using the **`EnvironmentBrowser.QueryConfigOption`** method. For example, the IDE controllers are created by default. Many of these default devices contain properties that you cannot change.

However, you can add the following optional devices to the default set: **`VirtualSerialPort`**, **`VirtualParallelPort`**, **`VirtualFloppy`**, **`VirtualCdrom`**, **`VirtualUSB`**, **`VirtualEthernetCard`**, **`VirtualDisk`**, and **`VirtualSCSIPassthrough`**. See the VirtualDevice Data Object in the *API Reference* for more information about each of these optional devices.

CAUTION
Do not try to change default device properties using the **`VirtualMachineConfigSpec.deviceChange`** method discussed in Adding Devices to Virtual Machines, because the **`deviceChange`** method is not applicable to default device properties.

Configuring a Virtual Machine

You can configure a virtual machine during creation (**`Folder.CreateVM_Task`**) or cloning (**`VirtualMachine.CloneVM_Task`**). You can also reconfigure a virtual machine using **`VirtualMachine.ReconfigVM_Task`**.

In vSphere 5.5 and later, the **`ReconfigVM_Task`** method will throw an error when it attempts to change certain virtual machine properties while the virtual machine is powered on. In previous releases, the server would modify the properties in the configuration specification and the changes would take effect after the virtual machine resets, reboots, or performs a fast suspend and resume.

Since vSphere 5.5, a reconfigure operation modifies the virtual machine in real time, so that the virtual machine properties have been changed by the time the method returns control to the client.

You cannot reconfigure a powered on virtual machine successfully if you specify any of the following properties in the **`VirtualMachineConfigSpec`** when you call the **`ReconfigVM_Task`** method.

Property	VirtualMachineConfigSpec Path
VirtualDevice.unitNumber	deviceChange.device.unitNumber
VirtualDevicePciBusSlotInfo.pciSlotNumber	deviceChange.device.slotInfo.pciSlotNumber
VirtualDiskFlatVer1BackingInfo.diskMode VirtualDiskFlatVer2BackingInfo.diskMode VirtualDiskRawDiskMappingVer1BackingInfo.diskMode VirtualDiskSeSparseBackingInfo.diskMode VirtualDiskSparseVer1BackingInfo.diskMode VirtualDiskSparseVer2BackingInfo.diskMode	deviceChange.device.backing.diskMode
VirtualDiskFlatVer2BackingInfo.digestEnabled VirtualDiskSeSparseBackingInfo.digestEnabled	deviceChange.device.backing.digestEnabled
VirtualMachineConfigSpec.changeTrackingEnabled	changeTrackingEnabled
VirtualEthernetCard.addressType	deviceChange.device.addressType
VirtualEthernetCard.macAddress	deviceChange.device.macAddress
VirtualEthernetCard.wakeOnLanEnabled	deviceChange.device.wakeOnLanEnabled
VirtualSCSIController.sharedBus	deviceChange.device.sharedBus
VirtualSerialPort.yieldOnPoll	deviceChange.device.yieldOnPoll
VirtualUSBController.autoConnectDevices VirtualUSBXHCIController.autoConnectDevices	deviceChange.device.autoConnectDevices
VirtualUSBController.ehciEnabled	deviceChange.device.ehciEnabled
VirtualMachineVideoCard.useAutoDetect	deviceChange.device.useAutoDetect
VirtualMachineVideoCard.videoRamSizeInKB	deviceChange.device.videoRamSizeInKB
VirtualMachineVideoCard.numDisplays	deviceChange.device.numDisplays

Table continued on next page

Continued from previous page

Property	VirtualMachineConfigSpec Path
VirtualMachineVideoCard.use3dRendererSupported	deviceChange.device.use3dRendererSupported

The *API Reference* lists all properties and includes information about required permissions for these configuration methods. The following sections describe some commonly specified attributes.

Name and Location

You can specify the display name for the virtual machine by setting the `VirtualMachineConfigSpec.name` property. Any % (percent) character used in this **`name`** parameter must be escaped, unless it is used to start an escape sequence. Clients can also escape any other characters in this name parameter.

Use the **`annotation`** field to provide a description of the virtual machine. To remove an existing description, specify the empty string as the value of **`annotation`**.

The location of the virtual machine is determined implicitly during creation because you call a **`Folder.CreateVM_Task`** method and specify resource pool and optional target host the virtual machine should belong to. See Resource Management for a discussion of resource pools and virtual machine location.

Hardware Version

The hardware version of a virtual machine indicates the lower-level virtual hardware features a virtual machine supports, such as BIOS, number of virtual slots, maximum number of CPUs, maximum memory configuration, and other hardware characteristics.

For a newly created virtual machine, the default hardware version is the most recent version available on the host where the virtual machine is created. To increase compatibility, you might want to create a virtual machine with a hardware version older than the highest supported version. You can do so by specifying the **`VirtualMachineConfigSpec.version`** property during virtual machine creation. For existing virtual machines, call the **`VirtualMachine.UpgradeVM_Task`** method.

The hardware version of a virtual machine can be lower than the highest version supported by the ESXi host it is running on under the following conditions:

- You migrate a virtual machine to a newer version of ESXi after it was created on a host that was running an earlier version of ESXi.
- You create a virtual machine on a newer version of ESXi by using an existing virtual disk that was created on a host that was running an earlier version of ESXi.
- You add a virtual disk created on a host that was running an earlier version of ESXi to a virtual machine created on a newer version of ESXi.

Virtual machines with hardware versions lower than 4 can run on ESX/ESXi 4.x hosts but have reduced performance and capabilities. In particular, you cannot add or remove virtual devices on virtual machines with hardware versions lower than 4 when they reside on an ESX/ESXi 4.x host. To make full use of these virtual machines, upgrade the virtual hardware.

Boot Options

You can control a virtual machine's boot behavior by setting the **`VirtualMachineConfigSpec.bootOptions`** property. The **`VirtualMachineBootOptions`** data object in that property allows you to specify the following properties:

- **`bootDelay`** – Delay before starting the boot sequence, in milliseconds.
- **`bootRetryDelay`** – Delay before a boot retry, in milliseconds. This property is only considered if the **`bootRetryEnabled`** property is set to **`true`**.

- **bootRetryEnabled** – If set to **true**, a virtual machine that fails to boot tries again after the **bootRetryDelay** time period has elapsed.
- **enterBIOSSetup** – If set to **true**, the virtual machine enters BIOS setup the next time it boots. The virtual machine resets this flag to **false** so subsequent boots proceed normally.

Operating System

The guest operating system that you specify affects the supported devices and available number of virtual CPUs. You specify the guest operating system in the following two properties:

- **guestosid** – Specify one of the constants in the **VirtualMachineGuestOsIdentifier** as a string.
- **alternateGuestName** – Full name for the guest operating system. Use this property if **guestosid** is one of the values of **VirtualMachineGuestOsIdentifier** starting with **other*.**

CPU and Memory Information

To manage compute and memory resources at the virtual machine level, you use several properties of the **VirtualMachineConfigSpec** data object. These properties specify the CPU and memory configuration for the virtual machine. You can also specify the operational boundaries for resource allocation, and the priority of the virtual machine in case of resource contention.

Configuring Virtual CPUs and Memory

To configure the capabilities of a virtual machine, you use properties of the **VirtualMachineConfigSpec** data object to specify the machine characteristics available to the guest operating system. The guest uses these capabilities in the same way as resources on a physical machine.

Configuring Virtual CPUs

Set the number of virtual CPU cores for the virtual machine with the `VirtualMachineConfigSpec.numCPUs` property. Legal values for this property change depending on the **guestosid** value you specify. If you use `VirtualMachineConfigSpec` to update the virtual machine properties, you can omit this property to leave it unchanged.

The guest operating system acts as if it had `numCPUs` cores available at all times, but the host's physical resources are shared by all its virtual machines. The host allocates physical cores in time slices aa backing for virtual cores. For information about how to specify guidance for resource allocation, see Configuring Resource Allocation Constraints for Virtual Machines.

Configuring Multi-Core CPUs

Set the number of cores per CPU chip with the `VirtualMachineConfigSpec.numCoresPerSocket` property. The value must be an integral divisor of `VirtualMachineConfigSpec.numCPUs`. The default value is `1` if the property is omitted for virtual machine creation. If you use `VirtualMachineConfigSpec` to update the virtual machine properties, you can omit this property to leave it unchanged.

Configuring Memory

Set the RAM size for a virtual machine with the `VirtualMachineConfigSpec.memoryMB` property. If you use `VirtualMachineConfigSpec` to update the virtual machine properties, you can omit this property to leave it unchanged.

The guest operating system acts as if it had `memoryMB` available at all times, but the host's physical resources are shared by all its virtual machines. The amount of physical memory available as backing for virtual machines can vary over time, and it can affect virtual machine performance. For information about how to specify guidance for resource allocation, see Configuring Resource Allocation Constraints for Virtual Machines.

CPU Processors and Memory Affinity

If your virtual machine is on an ESXi system, and if you have a license that supports Symmetric Multiprocessors (SMP), you can configure the virtual machine to have multiple virtual CPUs by setting **`cpuAffinity`** and **`memoryAffinity`**. You define a set of integers that represents the processors (for CPU) and NUMA nodes (for memory). If you are reconfiguring the affinity setting and leave the array empty, any existing affinity is removed. See the *`Resource Management Guide`* for a discussion of NUMA nodes and affinity.

CPU Features

You can use the `VirtualMachineConfigSpec.cpuFeatureMask[].info` property to represent the CPU features requirements for a virtual machine or guest operating system. See the **`HostCpuIdInfo`** data object discussion in the *`API Reference`* for a detailed discussion.

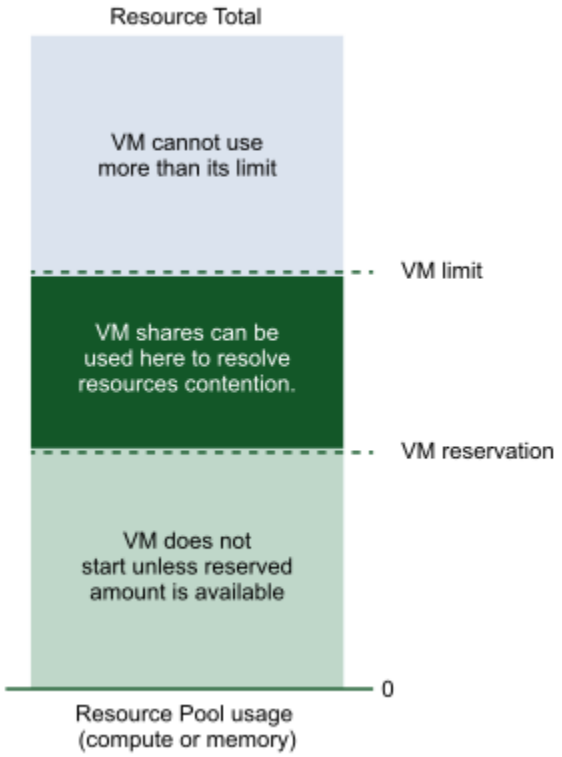
CPU and Memory Modification for Running Virtual Machines

Set **`CpuHotAddEnabled`** and **`CpuHotRemoveEnabled`** to specify whether virtual processors can be added to or removed from a virtual machine while the virtual machine is running. Set **`MemoryHotAddEnabled`** to specify whether memory can be added while the virtual machine is running.

Configuring Resource Allocation Constraints for Virtual Machines

The host hypervisor allocates physical resources as backing for the virtual resources needed by guest operating systems. You can specify resource allocation preferences that guide and constrain the hypervisor when allocating resources to running virtual machines.

To specify allocation preferences for CPU and memory, use the **`cpuAllocation`** and **`memoryAllocation`** properties of **`VirtualMachineConfigSpec`**. Both properties are data objects of type **`ResourceAllocationInfo`** objects. The **`ResourceAllocationInfo`** object has properties to specify upper and lower allocation limits, and properties to specify relative priorities when virtual machines contend for resources.



To specify boundaries for resource allocation, use these properties of **`ResourceAllocationInfo`**:

- **`reservation`** – Lower limit for resource allocation to the virtual machine. If the hypervisor cannot reserve this amount, it does not start the virtual machine. If the running virtual machine uses less than its reserved amount, other running virtual machines can use a part of the reserved resources temporarily.
- **`limit`** – Upper limit for CPU or memory resources assigned to this virtual machine. The virtual machine does not exceed this limit, even if unused resources are available. This property is typically used to limit the impact of the virtual machine on other running virtual machines. If the property is set to -1, the configured virtual memory size or the configured number of virtual cores limits the resource allocation.

  NOTE
  Although the number of virtual CPU cores limits the compute resource allocated, the virtual bandwidth allocated also depends on the speed of the physical CPU cores assigned as backing.

To specify preferences for resource allocation in case of contention, use the **`shares`** property of **`ResourceAllocationInfo`**. The **`shares`** property is a nested data object of type **`SharesInfo`**, which specifies a relative priority for resolving resource contention between virtual machines. The **`SharesInfo`** data object has two properties, **`level`** and **`shares`**:

- **`level`** – An enum type with four potential values: **`high`**, **`normal`**, **`low`**, and **`custom`**. If you specify the value **`custom`**, you must also specify the **`shares`** property. If you specify any other value for **`level`**, the **`shares`** property is ignored.
- **`shares`** – In a situation of resource contention, the **`shares`** value is compared against the **`shares`** values of other virtual machines, and resources are allocated first to the virtual machine with the highest **`shares`** value.

The most direct way to specify resource priority is to set the **`shares`** values manually. This is most useful in environments where all virtual machines use custom **`shares`** values. In a mixed environment, the configuration settings do not compare directly, which can lead to unexpected results.

A more sophisticated way of specifying priority is to specify **`high`**, **`normal`**, or **`low`** for the **`level`** property. These settings cause the host to calculate the shares of a virtual machine in a way that factors in the configured memory size or the

configured virtual CPU cores of the virtual machine. The result gives more priority shares to virtual machines with larger configurations.

> **NOTE**
> Both the custom shares values and the calculated shares values prioritize virtual machines only among their siblings running in the same resource pool. For a system of virtual machines running only in the default root resource pool, determining priority is fairly simple. For data centers with nested resource pools, a virtual machine's priority in the larger context of all running virtual machines depends also on the priority of its parent resource pool.

For a detailed examination of resource pools and how they affect priority shares calculations, see Resource Management. The *Resource Management Guide* includes a detailed discussion of resource allocation in the vSphere environment.

Networks

You configure network settings so that a virtual machine can communicate with the host and with other virtual machines.

Virtual Network Interfaces

You can add a virtual network interface to a virtual machine using a subclass of **VirtualEthernetCard**, you can set the **addressType** to **Manual**, **Generated**, or **Assigned**. If you choose **Assigned**, you can specify a MAC address explicitly.

For more information about virtual devices in general, see Adding Devices to Virtual Machines.

The number of virtual network interfaces depends on the hardware version you specify for a virtual machine. Hardware version 7 virtual machines support up to ten virtual NICs. Hardware version 4 virtual machines support up to four virtual NICs.

Virtual Machine MAC Address

Upon virtual machine creation, ESXi or vCenter Server systems assign each virtual network interface its own unique MAC address. The first three bytes of the MAC address that is generated for each virtual network adapter consists of a manufacturer-specific Organizationally Unique Identifier (OUI). The MAC address-generation algorithm produces the other three bytes. vSphere generates MAC addresses that are checked for conflicts. After the MAC address has been generated, it does not change unless the virtual machine is moved to a different location.

All MAC addresses that have been assigned to virtual network interfaces of running and suspended virtual machines on a given physical machine are tracked. The MAC address of a powered off virtual machine is not checked against those of running or suspended virtual machines. It is possible that a virtual machine acquires a different MAC address after a move.

The *ESXi Configuration Guide* discusses virtual machine MAC addresses in detail.

Fibre Channel NPIV Settings

N-port ID virtualization (NPIV) supports sharing a single physical FC HBA port among multiple virtual ports, each with unique identifiers. This capability lets you control virtual machine access to LUNs on a per-virtual machine basis.

Each virtual port is identified by a pair of world wide names (WWNs): a world wide port name (WWPN) and a world wide node name (WWNN). These WWNs are assigned by vCenter Server. For detailed information on how to configure NPIV for a virtual machine, see the *Fibre Channel SAN Configuration Guide*.

NPIV support is subject to the following limitations:

- NPIV must be enabled on the SAN switch. Contact switch vendors for information about enabling NPIV on their devices.
- NPIV is supported only for virtual machines with RDM disks. Virtual machines with regular virtual disks continue to use the WWNs of the host's physical HBAs.
- Virtual machines on a host have access to a LUN using their NPIV WWNs if the physical HBAs on the ESXi host have access to a LUN using its WWNs. Ensure that access is provided to both the host and the virtual machines

You can set up NPIV with the **`VirtualMachineConfigSpec`** properties that start with **`npiv`**.

File Locations

File locations for a virtual machine are specified in the following properties:

- **`VirtualMachineConfigSpec.files`** is a **`VirtualMachineFileInfo`** data object that allows you to specify the log directory, snapshot directory, suspend directory, and configuration file location. Most locations have a default that you can change as needed.
- **`VirtualMachineConfigSpec.locationID`** is a 128-bit hash based on the virtual machine's configuration file location and the UUID of the host the virtual machine is assigned to. This property is not usually set by developers; however, clearing this property by setting it to an empty string is recommended if you move the virtual machine.

If a virtual machine's **`VirtualMachineCapability.swapPlacementSupported`** property is **`true`** for a virtual machine, you can specify a value for the **`VirtualMachineConfigSpec.swapPlacement`** property. The value must be one of the values of the **`VirtualMachineConfigInfoSwapPlacementType`** enumeration, as a string.

Adding Devices to Virtual Machines

You can add devices to a virtual machine during creation using the **`VirtualMachineConfigSpec.deviceChange`** property, which is a **`VirtualDeviceSpec`**. You specify the host device that the virtual device should map to by using a backing object. A backing object represents the host device associated with a virtual device.

- Backing option objects – You can find out which devices the host supports by extracting the relevant backing option object.
- Backing information object – The backing information object allows you to supply data for virtual device configuration. You access a **`VirtualDeviceBackinInfo`** object as follows:
  `VirtualMachineConfigSpec.deviceChange[].device.backing`

To add a device to a virtual machine, you must first find out which devices are supported on the corresponding ESXi host, and then specify a **`VirtualDevice`** object. Perform these tasks to add a device to a virtual machine:

1. Find out which devices your ESXi system supports by calling the **`QueryConfigOption`** method, which you can access through the **`VirtualMachine.environmentBrowser`** property. The method returns a **`VirtualMachineConfigOption`** data object that specifies what the ESXi supports. For example, **`VirtualMachineConfigOption.hardwareOptions`** includes information about supported CPU and memory and an array of **`VirtualDeviceOption`** data objects.

   NOTE
   You cannot use the **`QueryConfigOption`** method to create another instance of a default device. If you attempt to add a default device, such as an IDE controller, the server ignores the operation.

2. Specify the backing information object for the device. The actual process for defining the object differs for different objects. For example, for a CD-ROM passthrough device, you use a **`VirtualCdromPassthroughBackingInfo`** device. The **`VirtualDevice.backing`** property is a **`VirtualDeviceBackingInfo`** object which is extended by devices.

The following code fragment adds a CD-ROM passthrough device:

```
VirtualCdromPassthroughBackingInfo vcpbi = new VirtualCdromPassthroughBackingInfo();

// Does the virtual device have exclusive access to the CD-ROM device?

vcpbi.setExclusive(false);

// Specifies the device name.

vcpbi.setDeviceName('cdrom0');
```

3. Specify connection information for the device.

   The **`VirtualDevice.connectable`** property is a **`VirtualDeviceConnectInfo`** data object. This object provides information about restrictions on removing the device while a virtual machine is running. This property is **`null`** if the device is not removable.

   ```
   VirtualDeviceConnectInfo vdci = new VirtualDeviceConnectInfo();

   // Allow the guest to control whether the virtual device is connected?

   vdci.setAllowGuestControl(false);

   // Is the device currently connected?

   vdci.setConnected(true);

   // Connect the device when the virtual machine starts?

   vdci.setStartConnected(true);
   ```

4. Define the controller key, the virtual device key, and the unit number.

   You define these items with the integer properties: **`controllerKey`**, **`key`**, and **`unitNumber`**. See the **`VirtualDevice`** data object in the *API Reference*.

5. Specify device Information.

   The **`deviceInfo`** property is a **`Description`** data object that has a **`name`** property and a **`summary`** property. You can supply a string value for each, describing the device.

   ```
   Description vddesc = new Description();

   vddesc.setLabel('CD-ROM Device cdrom0');

   vddesc.setSummary('The CD-ROM device for this virtual machine.');
   ```

6. Specify the virtual device as the **`device`** property of a **`VirtualDeviceConfigSpec.`**
7. Specify the **`VirtualDeviceConfigSpec`** as the **`deviceChange`** property to the **`VirtualMachineConfigSpec`** that you pass in to a **`Folder.CreateVM_Task`** or **`VirtualMachine.ReconfigVM_Task`** method.

Here's the complete code fragment for a CD-ROM passthrough device:

```
VirtualDevice vd = new VirtualDevice();

vd.setBacking(vcpbi);

vd.setConnectable(vdci);
```

```
vd.setControllerKey(257);

vd.setDeviceInfo(vddesc);

vd.setKey(2);

vd.setUnitNumber(25);
```

Performing Virtual Machine Power Operations

Just like physical machines, virtual machines have power states.

- Powered on – The virtual machine is running. If no OS has been installed, you can perform OS installation as you would for a physical machine.
- Powered off – The virtual machine is not running. You can still update the software on the virtual machine's physical disk, which is impossible for physical machines.
- Suspended – The virtual machine is paused and can be resumed; like a physical machine in standby or hibernate state.

> **IMPORTANT**
> Before you power on a virtual machine, you must make sure that the host has sufficient resources. You must have enough memory for the virtual machine, and some memory overhead. See Querying Virtual Machine Memory Overhead.

`VirtualMachine` power operations allow you to change the power state. Each operation is sensitive to the current power state, for example, powering on a powered off virtual machine has the desired result while powering on a powered on virtual machine results in an error. You must check the current state before you run one of these tasks.

- **`PowerOnVM_Task`** – Powers on a virtual machine. If the virtual machine is suspended, this method resumes execution from the suspend point.
- **`PowerOffVM_Task`** – Powers off a virtual machine.
- **`ResetVM_Task`** – Resets power on this virtual machine. If the current state is **`poweredOn`**, **`ResetVM_Task`** first performs a hard **`powerOff`** operation. After the power state is poweredOff, **`ResetVM_Task`** performs a **`powerOn`** operation.
  Although this method functions as a **`powerOff`** followed by a **`powerOn`**, the two operations are atomic with respect to other clients, meaning that other power operations cannot be performed until the reset method completes.
- **`SuspendVM_Task`** – Suspends the virtual machine. You can later power on the suspended virtual machine to the same state.

Virtual machines are often configured to start up the guest operating system when they are started, and try to shut down the guest operating system when being shut down. However, starting and stopping a virtual machine differs from starting and stopping the guest operating system (see Customizing the Guest Operating System).

> **IMPORTANT**
> Power operations might affect other virtual machines that are participating in a DRS cluster or VMware HA. See Resource Management for information about DRS clusters and VMware HA.

You can use the **`Datacenter.PowerOnMultiVM_Task`** to power on multiple virtual machines in a datacenter. Pass an array of **`VirtualMachine`** managed object references and an array of option values to the method. If any of the virtual machines in the list is manually managed by VMware DRS, the system generates a DRS recommendation that the user needs to apply manually. Standalone or DRS disabled virtual machines are powered on for the current host. Virtual machines managed by DRS, to be placed by DRS, are powered on for the recommended host.

Registering and Unregistering Virtual Machines

When you create a virtual machine, it becomes part of the inventory (inside the folder from which you called the creation method by default), and it is registered. If you copy virtual machine files to relocate the virtual machine, or if you remove the files from the inventory using the vSphere Client, it becomes unregistered and unusable. You cannot power on a virtual machine that is not part of the inventory.

To restore the virtual machine to the inventory, and make it usable again, you can use the **`RegisterVM_Task`** method, defined in the **`Folder`** managed object. You can register the virtual machine to a host or to a resource pool. You can register the virtual machine as a template if you want to use it to clone other virtual machines from.

The `ColdMigration.java` sample illustrates both registering and reconfiguring a virtual machine. At the heart of the sample is the following call, which registers the virtual machine. Arguments include the virtual machine's current folder, datastore path, and name, whether to register as a template, and the resource pool or host to register the machine in.

```
ManagedObjectReference taskmor = cb.getConnection().getService().registerVM_Task(
                    vmFolderMor,vmxPath,getVmName(),false,resourcePool,host);
```

After registration, the virtual machine takes its resources (CPU, memory, and so on) from the resource pool or host to which it is registered.

The `RemoveManagedObject.java` sample illustrates unregistering a virtual machine.

Customizing the Guest Operating System

You install the guest operating system on the virtual machine just as you would install it on a physical machine. Afterwards, you can use the Web Services API to retrieve information and perform some customization if VMware Tools is installed on top of the guest operating system.

`VirtualMachine` includes the following methods for managing the guest operating system:

- **`ShutdownGuest`** and **`RebootGuest`** shut down and reboot the guest OS, and **`StandbyGuest`** puts the guest in hybernate mode. In each case, you perform the action on the guest OS. For example, you might shut down Windows but leave the virtual machine running.
- **`ResetGuestInformation`** clears cached guest information. Guest information can be cleared only if the virtual machine is powered off. Use this method if stale information is cached, preventing reuse of an IP address or MAC address.
- **`SetScreenResolution`** sets the console screen size of the guest operating system. When you call this method, the change is reflected immediately the virtual machine console you can access in the vSphere Client.

You can customize the identity and network settings of the guest OS with the **`CustomizationSpec`** data object that is a parameter to **`VirtualMachine.CustomizeVM_Task`**. The **`CustomizationSpec`** is also a property of the **`VirtualMachineCloneSpec`** you pass in when cloning a virtual machine.

The settings you customize with this method are primarily virtual machine settings, but because the virtual machine and the guest OS share the information, you are also customizing the guest OS with this method.

The **`CustomizationSpec`** allows you to set the following properties:

- **`encryptionKey`** – Array of bytes that can be used as the public key for encrypting passwords of administrators.
- **`globalIPSettings`** – Contains a **`CustomizationGlobalIPSettings`** data object which specifies a list of DNS servers and a list of name resolution suffixes for the virtual network adapter.
- **`identity`** – Allows you to specify the network identity and settings, similar to the Microsoft Sysprep tool.
- **`nicSettingMap`** – Custom IP settings that are specific to a particular virtual network adapter.
- **`options`** – Optional operations (either LinuxOptions or WinOptions).

Installing VMware Tools

VMware Tools is a suite of utilities that enhances the performance of a virtual machine's guest operating system and improves virtual machine management. For each guest OS, VMware provides a specific binary-compatible version of VMware Tools. The SDK requires that you install VMware Tools, or some operations related to the guest operating system fail.

IMPORTANT
You must install the guest operating system before you install VMware Tools.

With VMware Tools installed on the guest OS, the virtual machine obtains its DNS (domain name server) name and an IP address and is therefore reachable over the network.

`VirtualMachine` includes three methods for automating installation and upgrade of VMware Tools.

- **`MountToolsInstaller`** – Mounts the VMware Tools CD installer as a CD-ROM for the guest operating system. To monitor the status of the tools installallation, check **`GuestInfo.toolsStatus`**. Check **`GuestInfo.toolsVersionStatus`** and **`GuestInfo.toolsRunningStatus`** for related information.
- **`UnmountToolsInstaller`** – Unmounts the VMware Tools installer CD.
- **`UpgradeToolsTask`** – Performs an upgrade of VMware Tools. This method assumes VMware Tools has been installed and is running. The method takes one argument, **`InstallerOptions`**, which allows you to specify command-line options passed to the installer to modify the installation procedure for tools.

Use the**`ToolsConfigInfo`** data object in **`VirtualMachineConfigSpec.toolsInfo`** property to specify the settings for the VMware Tools software running on the guest operating system.

Upgrading a Virtual Machine

You can upgrade virtual machine hardware by running the **`VirtualMachine.UpgradeVM_Task`** method. The method upgrades this virtual machine's virtual hardware to the latest revision that is supported by the virtual machine's current host. You can specify the version number as an argument. This method is useful if you want to run your virtual machine on a newer hypervisor that supports newer versions of the hardware.

Virtual Machine Management

Virtual machines can perform like physical computers and can be configured like physical computers. Virtual machines also support special features that physical computers do not support. This chapter discusses some of these features: migrating virtual machines, using snapshots, and using linked virtual machines.

For more information about configuring virtual machines, see the chapter Configuring a Virtual Machine.

Virtual Machine Migration

Migration is the process of moving a virtual machine from one host or storage location to another. Copying a virtual machine creates a new virtual machine. It is not a form of migration. vSphere supports the following migration types:

Migration Type	Description
Cold migration	Moves a powered-off virtual machine to a new host. Optionally, you can relocate configuration and disk files to new storage locations.
Migration of a suspended virtual machine	Moves a suspended virtual machine to a new host. Optionally, you can relocate configuration and disk files to new storage location.
Migration with vMotion	Moves a powered-on virtual machine to a new host. Migration with vMotion allows you to move a virtual machine to a new host without interruption in the availability of the virtual machine.

Table continued on next page

Continued from previous page

Migration Type	Description
Migration with Storage vMotion	Moves the virtual disks or configuration file of a powered-on virtual machine to a new datastore. Migration with Storage vMotion allows you to move a virtual machine's storage without interruption in the availability of the virtual machine.

Migration of a suspended virtual machine and migration with vMotion are both sometimes called hot migration, because they allow migration of a virtual machine without powering it off.

You can move virtual machines manually or set up a scheduled task to perform the cold migration.

Cold Migration

If a virtual machine is shut down, you can move it to a different cluster, resource pool, or host by copying all virtual machine files to a different directory. The **`ColdMigration`** example illustrates this.

Migration with vMotion

VMware vMotion supports the live migration of running virtual machines from one physical server to another with no downtime. The source and destination physical servers may be in the same datacenter or in different datacenters.

When calling the **`VirtualMachine`** object's **`MigrateVM_Task`** method, you can specify a host or resource pool to migrate to, and optionally the task priority and power state of the virtual machine. The **`VMotion`** example makes the following queries, and performs the migration if possible:

- Uses **`QueryVMotionCompatibility_Task`** to check that two hosts are compatible.
- Uses **`CheckMigrate_Task`** to check whether migration is feasible. If two hosts are not compatible, virtual machines cannot be migrated from one to the other.
- Uses **`CheckRelocation_Task`** to check whether relocation is possible.

`MigrateVM_Task` was deprecated in vSphere 6.5 and can be replaced with **`RelocateVM_Task`**.

Using Storage vMotion

Storage vMotion allows you to move a running virtual machine from one storage cluster to another. Taking the virtual machine or its associated storage offline is not required. All datastore types are supported, including local storage, VMFS, NAS (network attached storage), and VVols (virtual volumes).

You can place the virtual machine and all its disks in a single location, or select separate locations for the virtual machine configuration file and each virtual disk. The virtual machine remains on the same host during Storage vMotion.

To perform Storage vMotion, call the **`VirtualMachine`** object's **`RelocateVM_Task`** method. The **`RelocateVMSpec`** passed in to the method allows you to specify the target datastore and target host or resource pool.

As of vSphere 6.5, you should call **`RelocateVM_Task`** for all types of vMotion:

- cold relocate, whether VM storage moves or not
- vMotion within a cluster
- Storage vMotion
- cross-datacenter vMotion (XvMotion)
- Folder moves not involving VM moves

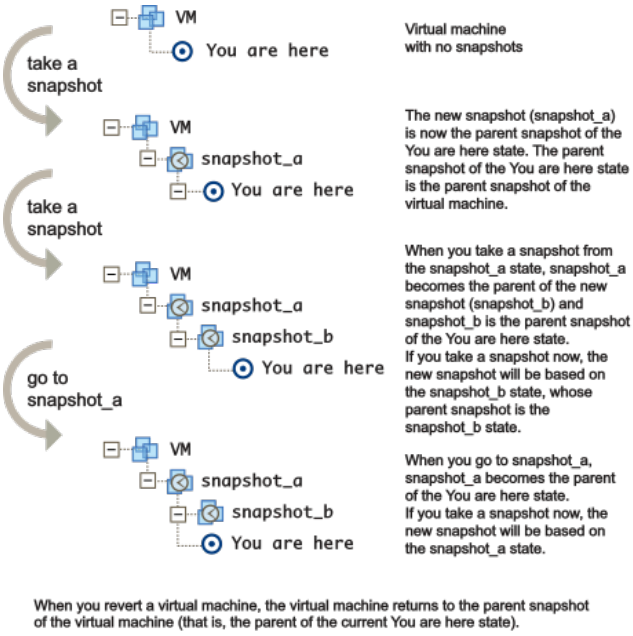
Snapshots

A snapshot is reproduction of the virtual machine just as it was when you took the snapshot. The snapshot includes the state of the data on all virtual machine disks and the virtual machine power state (on, off, or suspended). You can take a snapshot when a virtual machine is powered on, powered off, or suspended.

When you create a snapshot, the system creates a delta disk file for that snapshot in the datastore and writes any changes to that delta disk. You can later revert to the previous state of the virtual machine.

The **`VirtualMachine`** object has methods for creating snapshots, reverting to any snapshot in the tree, and removing snapshots.

Figure 32: Virtual Machine Snapshots



Snapshot hierarchies can become fairly complex. For example, assume that, in the example in Figure: Virtual Machine Snapshots, you revert to **`snapshot_a`**. You might then work with and make changes to the **`snapshot_a`** virtual machine, and create a new snapshot, creating, in effect, a branching tree.

Creating a Snapshot

The **`VirtualMachine.CreateSnapshot_Task`** method creates a new snapshot of a virtual machine. As a side effect, the current snapshot becomes the parent of the new snapshot.

The method allows you to specify a name for the snapshot and also requires you set the **`memory`** and **`quiesce`** properties:

- **`memory`** – If **`true`**, a dump of the internal state of the virtual machine (basically a memory dump) is included in the snapshot. Memory snapshots consume time and resources, and take a while to create. When set to **`false`**, the power state of the snapshot is set to powered off.
- **`quiesce`** – If **`true`** and the virtual machine is powered on when the snapshot is taken, VMware Tools is used to quiesce the file system in the virtual machine. This ensures that a disk snapshot represents a consistent state of the guest file systems. If the virtual machine is powered off or VMware Tools is not available, the quiesce flag is ignored.

The `VMSnapshot.java` example calls this method as follows:

```
ManagedObjectReference taskMor = service.createSnapshot_Task(
                                        vmMor, snapshotName, desc, false, false);
```

The method returns MOR to a **`Task`** object with which to monitor the operation. The **`info.result`** property in the **`Task`** contains the newly created **`VirtualMachineSnapshot`** upon success.

Reverting to a Snapshot

When you revert to a snapshot, you restore a virtual machine to the state it was in when the snapshot was taken. The **`VirtualMachine.RevertToSnapshot_Task`** allows you to specify a target host and whether the virtual machine should be powered on.

If the virtual machine was running when the snapshot was taken, and you restore it, you must either specify the host to restore the snapshot to, or set the **`SupressPowerOn`** flag to **`true`**.

Deleting a Snapshot

You can delete all snapshots by calling **`VirtualMachine.RemoveAllSnapshots`** or by calling the **`VirtualMachineSnapshot.RemoveSnapshot_Task`** method. The **`VirtualMachineSnapshot`** object was previously returned in the task returned by the **`CreateSnapshot_Task`** method.

Linked Virtual Machines

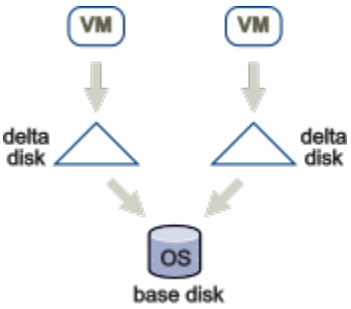
Linked virtual machines are two or more virtual machines that share storage and support efficient sharing of duplicated data.

Linked Virtual Machines and Disk Backings

In its simplest form, shared storage is achieved through the use of delta disk backings. A delta disk backing is a virtual disk file that sits on top of a standard virtual disk backing file. Each time the guest operating system on a virtual machine writes to disk, the data is written to the delta disk. Each time the guest operating system on a virtual machine reads from disk, the virtual machine first targets the disk block in the delta disk. If the data is not on the delta disk, the virtual machine looks for it on the base disk.

Linked virtual machines can be created from a snapshot or from the current running point. After you create a set of linked virtual machines, they share the base disk backing and each virtual machine has its own delta disk backing, as shown in Figure: Linked Virtual Machines with Shared Base Disk Backing and Separate Delta Disk Backing.

Figure 33: Linked Virtual Machines with Shared Base Disk Backing and Separate Delta Disk Backing



CAUTION

We recommend a limit of up to eight host virtual machines accessing the same base disk in a linked virtual machine group. However, you can have an unlimited number of linked virtual machines within each host virtual machine in the group.

Limitation for HA Clusters

Virtual machines in a linked clone group can be part of a VMware HA (high availability) cluster. The number of hosts in a cluster might affect HA's ability to restart a failed virtual machine.

- Clusters that contain ESXi 5.0 or earlier hosts – If a cluster has eight or fewer hosts, then linked virtual machines restart properly. However, if the cluster has more than eight hosts and any of the hosts are ESXi 5.0 or earlier, HA might not be able to restart a virtual machine after it fails. HA is not aware that virtual machines in the linked clone group are subject to the eight host limit. In this case, when HA responds to a failure, it might try to restart the virtual machine on a host that cannot participate in the group due to the maximum host limit. HA will attempt failover five times to different hosts. Thus, in clusters with 13 or more hosts, it is possible that HA will never try a host that is associated with the linked clone group.
- Clusters that contain only ESXi 5.1 or later hosts – The maximum host limit for a linked clone group is the maximum number of hosts allowed in a cluster. In this case, the number of hosts in the cluster does not affect the ability to restart failed virtual machines.

Creating a Linked Virtual Machine

You can create linked virtual machines in one of two ways:

- Clone the virtual machine from a snapshot.
- Clone the virtual machine from the current virtual machine state. This state might differ from the snapshot point.

Creating a Linked Virtual Machine From a Snapshot

You first create a snapshot, and then create the linked virtual machine from the snapshot.

1. To create the snapshot, call the **`CreateSnapshot_Task`** method for the virtual machine. The virtual machine can be in any power state. The following pseudo code creates a snapshot named **`snap1`**. The code does not include a memory dump. VMware Tools is used to quiesce the file system in the virtual machine if the virtual machine is powered on.

   ```
   myVm.CreateSnapshot("snap1", "snapshot for creating linked virtual machines", False,
   True)
   ```

2. To create the linked virtual machine, specify the snapshot you created and use a **`VirtualMachineRelocateDiskMoveOptions.diskMoveType`** of **`createNewDeltaDiskBacking`**, as illustrated in Creating a Linked Virtual Machine from a Snapshot. Creating linked virtual machines from a snapshot works with virtual machines in any power state.

Creating a Linked Virtual Machine from a Snapshot

```
relSpec = new VirtualMachineRelocateSpec()

relSpec.diskMoveType = VirtualMachineRelocateDiskMoveOptions.createNewChildDiskBacking


cloneSpec = new VirtualMachineCloneSpec()

cloneSpec.powerOn = False

cloneSpec.template = False

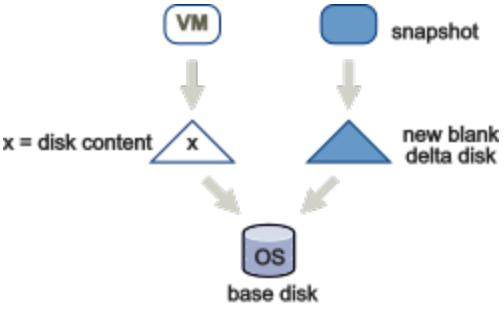
cloneSpec.location = relSpec

cloneSpec.snapshot = myVm.snapshot.currentSnapshot
```

```
myVm.Clone(myVm.parent, myVm.name + "-clone", cloneSpec)
```

The result is a virtual machine with the same base disk as the original, but a new delta disk backing.

Figure 34: Creating a Linked Virtual Machine from a Snapshot



Creating a Linked Virtual Machine From the Current Running Point

To create a virtual machine from the current running point, clone the virtual machine, as in , but use a **`diskMoveType`** of **`moveChildMostDiskBacking`**. The virtual machine can be in any power state.

For more information about cloning a virtual machine, see Creating a Linked Virtual Machine from a Snapshot.

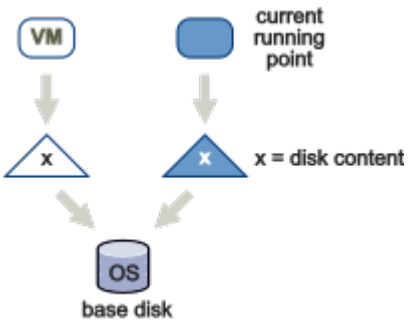
Creating a Linked Virtual Machine from the Current Running Point

```
relSpec = new VirtualMachineRelocateSpec()
relSpec.diskMoveType = VirtualMachineRelocateDiskMoveOptions.moveChildMostDiskBacking

cloneSpec = new VirtualMachineCloneSpec()
cloneSpec.powerOn = False
cloneSpec.template = False
cloneSpec.location = relSpec

myVm.Clone(myVm.parent, myVm.name + "-clone", cloneSpec)
```

Figure 35: Creating a Linked Virtual Machine from the Current Running Point



Instant Clone Virtual Machines

The instant clone operation creates a running virtual machine that continues running from the current state of the source virtual machine. There is no delay to power on the cloned virtual machine.

An instant clone virtual machine is a kind of linked clone with these characteristics:

- The clone starts out powered on and running.
- The clone replicates the running state of the source virtual machine.
- The clone shares memory pages with the source virtual machine as long as they remain unmodified.

The instant clone operation facilitates rapid deployment of virtual machines for low-latency response to peak computing demand. It also uses memory and disk space as efficiently as possible.

Because the clone replicates the running state of the source virtual machine, it usually requires customization of some network properties in the guest operating system to eliminate conflicts with the source. An instant clone virtual machine may also require new backings for some dedicated virtual devices, such as ISO files that back virtual optical drives.

Instant Clone Terminology

The instant clone feature has its own set of terminology, which uses some terms in ways specific to the feature.

stun
: To stop the clock of a virtual machine temporarily, preventing guest operations during an instant clone operation.

freeze
: To stop the clock of a virtual machine indefinitely, allowing a series of instant clone operations that reproduce the exact process state of the source virtual machine.

suspend
: Save the state of a virtual machine so applications can resume, and put it to sleep.

resume
: Continue running the virtual machine that was suspended, including its applications.

unstun
: To release a stunned source virtual machine, allowing I/O operations to proceed.

source
: A virtual machine that is cloned during an instant clone operation.

instant clone

1. An operation that quickly creates a new virtual machine that is an exact copy of a source virtual machine.
2. A new virtual machine created by an instant clone operation.

Run State of the Instant Clone Source

An instant clone can be created either from a source virtual machine in a frozen state, or from the current running point of a source virtual machine. Certain characteristics of the process and of the result can affect your choice of frozen virtual machine or current running point for the source virtual machine.

Advantages of Cloning from the Current Running Point

An instant Clone created from the current running point of a source virtual machine does not require an agent running in the guest operating system to invoke the frozen state.

If you create an instant clone from the current running point, the source virtual machine continues running from that point after a very brief delay, whereas a frozen source virtual machine requires intervention by the host to thaw it.

Advantages of Cloning from a Frozen Virtual Machine

A source virtual machine in the frozen state allows you to deploy any number of instant clones in exactly the same state, because the guest operating system retains its state after the clone operation.

When you create instant clones from the current running point of a source virtual machine, each cloning operation adds another delta disk to the source virtual machine. Each delta disk adds a small performance penalty, but a long chain of delta disks causes a noticeable performance penalty for virtual disk accesses.

The Instant Clone Process from a Frozen Source

The process to create an instant clone from a frozen source virtual machine requires one-time initial preparation, but after that it proceeds faster than creating an instant clone from the current running point. The source virtual machine must be booted and running, in a state that you want to replicate to other virtual machines. VMware Tools must be installed and running in the guest operating system so that you can run the command to freeze the source virtual machine .

Freezing the Source Virtual Machine for an Instant Clone Operation

Before you start the instant clone operation, you issue a VMware Tools command in the guest operating system, which freezes the source virtual machine. This is only needed one time, prior to the first instant clone operation. After that, the source virtual machine remains frozen. When VMware Tools receives the command to freeze the virtual machine, it passes the command through a back door to the host hypervisor.

The command to freeze the virtual machine depends on the guest operating system.

- For a Linux guest operating system, either of the following commands will freeze the virtual machine:
  - `vmware-rpctool "instantclone.freeze"`
  - `vmtoolsd -cmd "instantclone.freeze"`
- For a Windows guest operating system, the following command will freeze the virtual machine:
  - `C:\Program Files\VMware\VMware Tools\vmtoolsd.exe --cmd "instantclone.freeze"`

When a virtual machine is frozen, the hypervisor prevents future context switches to the guest operating system, which in effect stops the virtual machine's clock, although the virtual machine status continues to indicate that it is running. You can determine whether the virtual machine is frozen by examining the `runtimeinfo.InstantCloneFrozen` flag.

Instant Clone API Methods

The vSphere Web Services API offers several methods that implement different parts of the instant clone operation. By dividing the functionality into separate methods, the API provides flexibility in the timing and customization of the operation.

You can think of the instant clone operation as having three divisions: preparation, cloning, and customization.

Preparation includes these options:
- Installing the guest customization engine. The engine assists with customization of the guest network from the Web Services API. For more information, see Installing the Guest Customization Engine.
- Disconnecting virtual NICs. You have the option to disconnect the NICs in the source virtual machine before you start the clone operation, to avoid network collisions between the source and its clones. For more information, see Avoiding Network Collisions after Instant Clone Operations.
- Freezing the source virtual machine. If you need to create a large number of clones, consider cloning from a frozen source. For more information, see Run State of the Instant Clone Source.

Creating a clone of the source virtual machine requires a single method call. The `VirtualMachine.InstantClone_Task` method takes a parameter of type `InstantCloneSpec`, which enables you to give the clone a new name and configuration values. By using the key-value pairs in the `config` array, you can specify that the virtual NICs of the clone will start in a disconnected state, which avoids network conflicts with the source virtual machine. The `location` property of `InstantCloneSpec` is ignored.

Customization of a clone virtual machine takes place after the `InstantClone_Task` method completes. To customize, you invoke methods of the `VirtualMachineGuestCustomizationManager` managed object, as well as optional steps for application-dependent customization. For more information, see Guest Network Customization for Instant Clone Virtual Machines.

How the Instant Clone Operation Works

The instant clone operation involves the following sequence of steps.

Stunning the Instant Clone Source Virtual Machine

When you issue the API command to initiate an instant clone operation, the host begins by doing a stun operation on the source virtual machine. The stunned condition lasts a very short time, and is not directly visible to the API user.

The stunned condition blocks all I/O operations of the source virtual machine while the instant clone is created. The stun is redundant in the case of a frozen virtual machine. During the time the source virtual machine is stunned, the host creates a delta disk for the source virtual machine, which remains empty as long as the source virtual machine is stunned or frozen.

Copying Virtual Disks During the Instant Clone Operation

Then the host creates an empty delta disk for the instant clone virtual machine. This delta disk links to the base disk of the source virtual machine. More precisely, if the source virtual machine already had one or more delta disks, the instant clone delta disk links to the second delta disk in the disk chain of the source. The second delta disk is the delta disk that was the topmost delta disk when the stun operation was invoked.

Copying Memory During the Instant Clone Operation

The host creates a new virtual memory paging file for the instant clone virtual machine, whose page tables link to the paging file of the source virtual machine. When the instant clone virtual machine is complete and is allowed to run instructions, all its memory pages will be shared with the source virtual machine until either virtual machine does a write to memory.

Configuring the Instant Clone Virtual Machine

Then the host creates a virtual machine configuration file for the instant clone, which links to its delta disk and memory paging file. At this point, the instant clone virtual machine is capable of running instructions, and the host unstuns it. Both

the memory and the virtual disks are shared with the source virtual machine, so the instant clone is identical to the source virtual machine, from the guest operating system point of view.

When you create virtual machines with identical guest operating systems, the network settings conflict. Usually you will want to customize these settings and reset the network stack for each instance of the guest operating system.

Limitations of Instant Clones

The instant clone feature provides good performance and scalability, but it has a few limitations:

- The instant clone feature is not available when you connect directly to ESXi. Instant clone depends on functionality available only in vCenter Server.
- Delta disks created during the instant clone operation are not visible in the snapshot trees of the virtual machines. They cannot be managed with the VirtualMachineSnapshot managed object. Instead, you must manage the delta disks using the `VirtualMachine.Reconfigure()` method.
- The instant clone operation is partly parallel and partly serialized. You can start several operations concurrently, but part of the process is exclusive and requires serialization of requests.
- The instant clone operation responds to errors by backing out changes, such as delta disks, to restore the previous state of the file system and the source virtual machine. However, there is a small window of failure at the end of the operation where the cloned virtual machine is independent and cannot be backed out to recover from an error. This is a low risk situation.
- Instant clones create a delta disk for both the source and generated virtual machine, so resources can run out after vSphere raises a warning. The source virtual machine may be frozen to avoid excess growth. For current limits and ways to deal with failures, see the KB article *`Instant Clone fails due to exceeding maximum number of delta disks`*.
- The instant clone operation does not allow you to specify a different host for the clone. The clone must be on the same host as the source virtual machine because they share memory and virtual disk files.

Guest Customization of Instant Clone Virtual Machines

The instant clone operation produces a virtual machine whose memory is exactly the same as the source virtual machine. This can cause conflicts in network addressing, if two different virtual machines respond concurrently with the same network identity. The conflicting network attributes are usually the host name, IP address, and MAC address.

In the case of an instant clone derived from a frozen source virtual machine, a conflict might not immediately arise, because the source virtual machine is not active on the network. The instant clone simply assumes the identity of the source virtual machine when the host unstuns it. However, a second clone derived from the same frozen virtual machine could create a network conflict by duplicating the same identity as the first instant clone.

For an instant clone derived from the current running point of the source virtual machine, the problem is more acute because the source virtual machine resumes operation simultaneously with the instant clone.

To avoid network conflicts, you customize the instant clone to adopt a new identity, before allowing both the source virtual machine and the instant clone to run concurrently. In a situation where temporary network disruption is tolerable, you can ignore the conflict while you customize the network settings in the guest operating system of the instant clone.

Otherwise, you have two options to avoid the race condition that results when both virtual machines resume running from the same point. You can either disable the virtual NIC before the instant clone operation, or else you can specify that the virtual NIC in the instant clone will be disabled at the time when it resumes running after the operation. You specify the latter by setting the `config` property of the `spec` parameter when you invoke the `InstantClone_Task` method.

Avoiding Network Identity Collisions after Instant Clone Operations

To avoid a network conflict between a source virtual machine and its instant clone, you must disable at least one of the two conflicting virtual NICs until you complete network customization in the guest operating system of the instant clone. Here are some common approaches:

If the source virtual machine is frozen, its virtual NIC is effectively disabled, and you can reconfigure the guest network settings of the instant clone at any time before the source virtual machine is thawed.

If you derive the instant clone from the current running point of the source virtual machine, you can set the `connected` property of the virtual NIC backing to `false`, which disables it in the source before you start the instant clone operation. Then you re-enable the virtual NIC after the instant clone operation completes. After you customize the network settings in the guest operating system of the instant clone, you must re-enable the virtual NIC in the instant clone as well. This approach requires that you tolerate a network interruption until the instant clone operation is complete.

To avoid a network interruption, you can issue the instant clone operation with an `InstantCloneSpec` that specifies that the virtual NIC in the instant clone will be disconnected. The virtual NIC in the source virtual machine is not affected. After you complete guest operating system customization in the instant clone, you set the `connected` property of its virtual NIC back to `true`, and the instant clone operates under its new network identity.

In any case where you change the network identity settings in the guest operating system of the instant clone, you must reset the network stack for the virtual NIC to cause it to adopt the new settings. You can do this by invoking the `VirtalMachineGuestCustomizationManager.StartGuestNetwork_Task` method after you customize the clone's network settings, or you do the reset from the guest operating system, as with a physical machine. For more information, see Restarting the Guest Network After Customization and Resetting the Network Stack in a Virtual Machine.

Removing Snapshots and Deleting Linked Virtual Machines

After you have created a group of linked virtual machines, you can remove a snapshot that was the basis for a linked virtual machine, or delete a virtual machine. Those actions affect disks in the linked virtual machine group. Perform the actions when connected to a vCenter Server system for disk consolidation or deletion.

- Snapshot removal – During snapshot removal, the snapshot metadata is also removed, and the virtual machine from which the snapshot was taken is no longer shown as having snapshots. If you remove a snapshot while connected to the ESXi host directly, shared disks are not consolidated and unnecessary levels of delta disks might result. If you remove a snapshot while connected to a vCenter Server system, shared disks are not consolidated, but unshared disks are consolidated.
- Virtual machine deletion – When you delete a virtual machine by directly connecting to the ESXi host, shared disks are not deleted. When you delete a virtual machine by connecting to a vCenter Server system, shared disks are not deleted, but unshared disks are deleted.

  CAUTION

  Delete all linked virtual machines before deleting the source virtual machine from which they were created, so that you don’t have orphaned or corrupt disk files on your file system.

Relocating a Virtual Machine in a Linked Virtual Machine Group

You can move the virtual machines in a linked virtual machine group between datastores and save storage. The contents of the delta disk might not be as important as the contents of the base, and you can save storage by removing the delta disk.

For more information, see Figure: Creating a Linked Virtual Machine from a Snapshot.

Relocating a Linked Virtual Machine

```
relSpec = new VirtualMachineRelocateSpec()

relSpec.diskMoveType = VirtualMachineRelocateDiskMoveOptions.moveChildMostDiskBacking

relSpec.datastore = localDatastore

myVm.Relocate(relSpec)
```

You can relocate multiple linked virtual machines to a new datastore, but keep all shared storage during the relocation. To achieve the relocation, relocate the desired virtual machines one by one, giving the option to allow reattaching to an existing disk, as shown in Relocating Multiple Linked Virtual Machines.

Relocating Multiple Linked Virtual Machines

```
relSpec = new VirtualMachineRelocateSpec()

relSpec.diskMoveType =
VirtualMachineRelocateDiskMoveOptions.moveAllDiskBackingsAndAllowSharing

relSpec.datastore = targetDatastore

myVm.Relocate(relSpec)
```

Promoting a Virtual Machine's Disk

Promoting a virtual machine’s disk improves performance.

IMPORTANT
You can use the **`PromoteDisks`** API only when connected to a vCenter Server system.

You can use **`PromoteDisks`** to copy disk backings or to consolidate disk backings.

- Copy – If the **`unlink`** parameter is **`true`**, any disk backing that is shared by multiple virtual machines is copied so that this virtual machine has its own unshared version. Files are copied into the home directory of the virtual machine. This setting results in improved read performance, but higher space requirements. The following call copies and shares disks, and then collapses all unnecessary disks.
  `myVm.PromoteDisks(True, [])`
- Consolidate – If the **`unlink`** parameter is **`false`**, any disk backing that is not shared between multiple virtual machines and not associated with a snapshot is consolidated with its child backing. The net effect is improved read performance at the cost of inhibiting future sharing. The following call eliminates any unnecessary disks:
  `myVm.PromoteDisks(False, [])`

Promoting a virtual machine’s disk might also be useful if you end up with disk backings that are not needed for snapshots or for sharing with other virtual machines.

Both uses of **`PromoteDisks`** take an optional second argument, which allows you to apply the method to only a subset of disks. For example, you can unshare and consolidate only the virtual disk with key 2001 as follows:

```
for any of my VMs in dev

   if (dev.key == 2001)

      disk2001 = dev
```

```
myVm.PromoteDisks(True, [disk2001])
```

Performing Advanced Manipulation of Delta Disks

For advanced manipulation of delta disks, you can use **`VirtualDeviceConfigSpec`** methods such as **`VirtualDeviceConfigSpec.create`** and `VirtualDeviceConfigSpec.add`.

Together `add` and `create` allow you to create a blank delta disk on top of an existing disk. For the `VirtualDeviceConfigSpec` you specify **`add`** to prepare the VM for a new delta disk, then **`create`** to initiate the file operation for the delta disk whose **`parent`** property is an existing disk. These methods create a new delta disk whose parent is the pre-existing disk.

One use case is adding a delta disk on top of an existing virtual disk in a virtual machine without creating a snapshot. Adding a Delta Disk Backing illustrates how to add the delta disk for the first virtual disk in the virtual machine.

Adding a Delta Disk Backing

```
disk = None
for any of my VMs in dev
   if (VirtualDisk.isinstance == dev):
      disk = dev

# Remove the disk
removeDev = new VirtualDeviceConfigSpec()
removeDev.operation = "remove"
removeDev.device = disk

# Create a new delta disk which has the
# original disk as its parent disk
addDev = new VirtualDeviceConfigSpec()
addDev.operation = "add"
addDev.fileOperation = "create"
addDev.device = copy.copy(disk)
addDev.device.backing = copy.copy(disk.backing)
addDev.device.backing.fileName = "[" + disk.backing.datastore.name + "]"
addDev.device.backing.parent = disk.backing

spec = new VirtualMachineConfigSpec()
spec.deviceChange = [removeDev, addDev]
```

For working code with a similar use case, see the JAXWS sample program `VMDeltaDisk.java` in the SDK under `vsphere-ws/java`.

Virtual Machine Guest Operations

The vSphere Web Services SDK enables you to do operations inside a virtual machine, by communicating with the guest operating system.

Authenticating with the Guest Operating System

Authentication is required for most guest operations. Before you invoke a method that affects the guest, you must establish your credentials with the guest operating system and acquire a `GuestAuthentication` object that you pass to the method.

The `GuestAuthentication` type has several subclasses that hold guest credentials acquired by different methods. You can choose from several ways to authenticate:

- NamePasswordAuthentication
- SAMLTokenAuthentication
- SSPIAuthentication
- TicketedSessionAuthentication

Running Guest OS Operations

Guest OS operations manipulate processes, files, folders, and environment variables in a virtual machine's guest operating system.

The vSphere API offers the following managed object types for guest operations:

- GuestAuthManager – authenticate to acquire credentials in the guest OS.
- GuestFileManager – manipulate files, directories, and remote copying in the guest OS.
- GuestProcessManager – manipulate processes in the guest OS.
- GuestAliasManager – support single sign-on for guest operations; create and delete user aliases.
- GuestWindowsRegistryManager – manipulate keys and values in the Windows registry.
- VirtualMachineGuestCustomizationManager – customize guest settings, especially for instant clone virtual machines.

All the above managed objects are subclasses that inherit properties from GuestOperationsManager. In the vSphere API, the VirtualMachine and GuestInfo managed objects contain information about what guest operations might be running and relevant virtual machine state:

```
vim.VirtualMachine.guest()
```

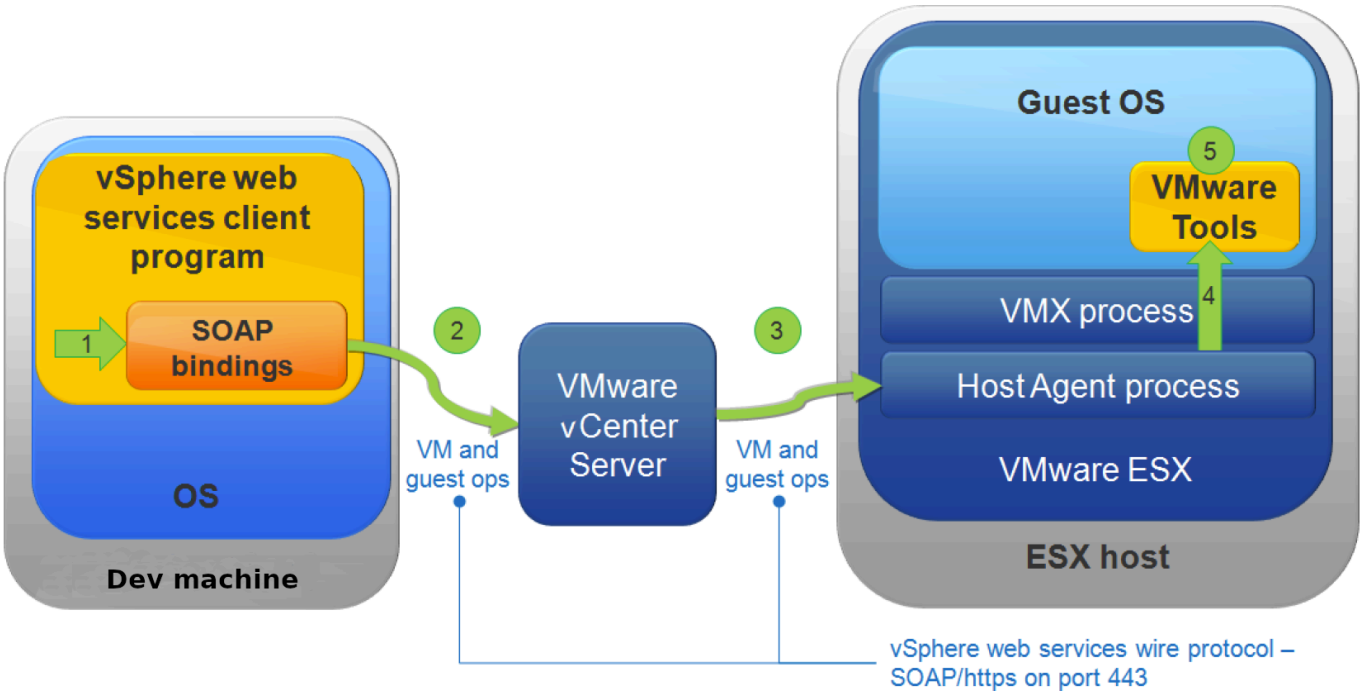
```
vim.vm.GuestInfo.guestOperationsReady
```

```
vim.vm.GuestInfo.interactiveGuestOperationsReady
```

VMware Tools must be present to run guest operations. To perform interactive guest operations, the user must be logged into the console, for example through the vSphere Client. Steps involved are:

1. Java program translates to SOAP bindings.
2. Directives of vSphere API pass through vCenter Server.
3. Directives are relayed to ESXi host agent process.
4. Virtual machine executable passes guest operations to VMware Tools.
5. Guest OS performs guest operations.

Figure 36: Guest operations control flow



The following table summarizes the methods available for guest operations.

Table 5: GuestOperationsManager methods

Managed Object	Methods	Description
GuestAliasManager	AddGuestAlias	define alias for guest account
	ListGuestAliases	list guest aliases for specified user
	ListGuestMappedAliases	list alias map for in-guest user
	RemoveGuestAliasByCert	remove certificate associated aliases
GuestAuthManager	AcquireCredentialsInGuest	authenticate, return session object
	ReleaseCredentialsInGuest	release session object
	ValidateCredentialsInGuest	check authentication data or timeout
GuestFileManager	ChangeFileAttributesInGuest	change attributes of file in guest
	CreateTemporaryDirectoryInGuest	make a temporary directory
	CreateTemporaryFileInGuest	create a temporary file
	DeleteDirectoryInGuest	remote directory in guest OS
	DeleteFileInGuest	remove file in guest OS
	InitiateFileTransferFromGuest	start file transfer from guest OS
	InitiateFileTransferToGuest	start file transfer to guest OS
	ListFilesInGuest	list files or directories in guest
	MakeDirectoryInGuest	make a directory in guest
	MoveDirectoryInGuest	move or rename a directory in guest
	MoveFileInGuest	rename a file in guest
GuestWindowsRegistryManager	CreateRegistryKeyInGuest	create a registry key
	DeleteRegistryKeyInGuest	delete a registry key

Table continued on next page

Continued from previous page

Managed Object	Methods	Description
	DeleteRegistryValueInGuest	delete a registry value
	ListRegistryKeysInGuest	list registry subkeys for a given key
	ListRegistryValuesInGuest	list registry values for a given key
	SetRegistryValueInGuest	set or create a registry value
GuestProcessManager	ListProcessesInGuest	list processes running in guest OS
	ReadEnvironmentVariableInGuest	read environment variable in guest
	StartProgramInGuest	start running program in guest
	TerminateProcessInGuest	stop a running process in guest

`InitiateFileTransferFromGuest` and `InitiateFileTransferToGuest` are useful for transferring small files between the host and guest. For large file transfers, virtual machines should be connected to the network because networking transfers are much faster.

Java Source Code Samples

Four Java code samples based on JAX-WS are available in the vSphere SDK for Web services, in this directory:

`SDK/vsphere-ws/java/JAXWS/samples/com/vmware/guest`

`CreateTemporaryFile.java` creates a temporary file inside a virtual machine, by calling the following method:

```
vimPort.createTemporaryFileInGuest(fileManagerRef, vmMOR, auth, prefix, suffix,
directoryPath);
```

`DownloadGuestFile.java` downloads a file from the guest to a specified path on the host where the client is running. The destination, a local file on the client host, is specified on the command line as `--localfilepath`.

```
vimPort.initiateFileTransferFromGuest(fileManagerRef, vmMOR, auth, guestFilePath);
```

`RunProgram.java` runs a specified program inside a guest operating system, with output re-directed to a temporary file, and downloads the resulting output to a file on the local client. The program must already exist on the guest, and is specified on the command line as `--guestprogrampath`. The output file to store on the client host is specified on the command line as `--localoutputfilepath`.

```
vimPort.startProgramInGuest(processManagerRef, vmMOR, auth, spec);
```

`UploadGuestFile.java` uploads a file from the client machine to a specified location inside the guest. The source, a local file on the client host, is specified on the command line as `--localfilepath`.

```
vimPort.initiateFileTransferToGuest(fileManagerRef, vmMOR, auth, guestFilePath,

                          guestFileAttributes, fileSize,
optionsmap.containsKey("overwrite"));
```

Guest Operating System Customization

You can use the vSphere Web Services SDK to customize several attributes of the guest operating system. The `CustomizationSpec` data object allows you to specify network and identity settings in the guest operating system.

Use the `CustomizationSpec` data object to specify the settings that you choose to modify. You can specify the following areas:

- IP addresses and gateway addresses for virtual NIC devices

- DNS servers
- Virtual hostname
- Domain name
- Time zone

You can customize a guest operating system while the virtual machine is stopped, by invoking the `VirtualMachine.CustomizeVM_Task` method. You can customize the guest operating system while the virtual machine is running, by invoking the `VirtualMachineGuestCustomizationManager.CustomizeGuest_Task` method. The instant clone feature requires that you customize the virtual machine while it is running. To initialize a virtual machine for a cloud environment, you must customize the virtual machine while it is stopped.

The two ways to customize a virtual machine are compared in the following table.

Table 6: Comparing Guest Operating System Customization

	Customization While Stopped	Instant Clone Customization
vSphere Compatibility	• vSphere 5 and later • Both vSphere and VMC	• vSphere 7 and later • vSphere only (not VMC)
API Method	`VirtualMachine.CustomizeVM_Task()`	• `VirtualMachineGuestCustomizationManager.CustomizeGuest_Task()` • `VirtualMachineGuestCustomizationManager.StartGuestNetwork_Task()` • `VirtualMachineGuestCustomizationManager.AbortCustomization_Task()`
Creating CustomizationSpec	Create customization specifications in vSphere Client or in API.	Create customization specifications in API.
Virtual Machine State	Must be powered off.	Must be powered on.
vmtools	Must be installed. For Linux cloud-init customization, use open-vm-tools.	Must be installed and running.
cloud-init	Must be installed if using cloud-init customization.	Not used.
Customization Engine	Not used.	Required.
Supported Guests	Both Linux and Windows. For supported versions, see https://partnerweb.vmware.com/programs/guestOS/guest-os-customization-matrix.pdf.	Linux only. Supported distributions are listed in Customizing Guest Network Settings for Running Virtual Machines.
Network Adapter State	For Sysprep or Linuxprep customization, `CustomizeVM_Task()` disconnects virtual adapters and prepares customization. The next time the virtual machine powers on, the	`InstantClone_Task()` disconnects virtual adapters before creating a clone. Use `StartGuestNetwork()` to restart virtual adapters after customization.

Table continued on next page

Continued from previous page

	Customization While Stopped	Instant Clone Customization
	customization will take effect and `vmtools` will reconnect virtual adapters. For CloudinitPrep customization, `CustomizeVM_Task()` does not disconnect virtual adapters.	
Rebooting the Guest	`vmtools` reboots the guest operating system after reconnecting virtual network adapters. No reboot needed after Linux cloud-init customization, either LinuxPrep or CloudinitPrep.	No reboot needed. `StartGuestNetwork()` restarts network services after customization.

Guest Customization for Stopped Virtual Machines

You can customize certain settings in the guest operating system with the help of the vSphere Web Services SDK. Customization is simpler for a virtual machine that is not running.

A guest operating system can be easily customized while it is not running. If stopping the guest will cause a significant service impact, such as in an instant clone virtual machine, see Guest Network Customization for Instant Clone Virtual Machines instead.

Use this general procedure to customize a guest operating system that you can shut down without a significant service impact. However, to initialize a virtual machine in a cloud environment, by using the cloud-init utility, see Guest Customization Using cloud-init.

1. Shut down the guest operating system by using the `ShutdownGuest` method or the `PowerOffVM_Task` method.
2. Prepare a `CustomizationSpec` data object that specifies the desired global IP settings, virtual NIC settings, and so on.
3. Invoke the `CustomizeVM_Task` method of the `VirtualMachine` managed object.
4. Start the guest operating system by using the `PowerOnVM_Task` method of the `VirtualMachine` managed object.

For more information about customizing the guest operating system, in the context of virtual machine configuration, see Customizing the Guest Operating System.

Guest Customization Using cloud-init

You can customize certain settings in a Linux guest operating system with the help of the vSphere Web Services SDK. The cloud-init utility is a standardized way to initialize a virtual machine when you deploy it in a cloud environment.

Guest operating systems can be customized in one of several ways, depending on certain cloud-init configuration flags and on the subtype of the `identity` property of the `CustomizationSpec` data object. The following table shows the combinations needed to achieve raw data cloud-init customization, LinuxPrep with traditional customization, or LinuxPrep with cloud-init customization. Other combinations are not allowed.

Table 7: Enabling Guest Customization Types

Flags	CustomizationSpec.identity	Result
`allow_raw_data: true` or absent (default=`true`)	`CloudInitPrep`	VMware Tools will work with cloud-init to customize the guest with the unprocessed `metadata` and `userdata` from `CloudinitPrep`.
`disable_vmware_customization : true` or absent (default=`true`)	`LinuxPrep`	VMware Tools will do traditional customization on the guest.
`disable_vmware_customization : false`	`LinuxPrep`	VMware Tools will work with cloud-init to customize the guest with configuration derived from `LinuxPrep`.

Use the following procedures to apply settings to a virtual machine by passing raw configuration data to the cloud-init utility. You will use the `CustomizeVM_Task` method with a `CustomizationSpec` that contains configuration data compatible with the cloud-init standard.

Prepare a Virtual Machine Template for Raw Data cloud-init Customization

The following steps prepare a virtual machine template for customization with raw cloud-init data.

1. In the guest operating system, install open-vm-tools version 11.3.0 or later. Configure the Linux guest to run open-vm-tools services when the guest operating system starts.
   Example for an Ubuntu 20.04 guest operating system:
   ```
   $ sudo apt update

   $ sudo apt install open-vm-tools

   $ sudo vmtoolsd -v

   $ sudo systemctl is-enabled open-vm-tools.service
   ```
2. In the guest operating system, install cloud-init version 21.1 or later. Configure the Linux guest to run cloud-init services when the guest operating system starts.
   Example for an Ubuntu 20.04 guest operating system:
   ```
   $ sudo apt update

   $ sudo apt install cloud-init

   $ sudo cloud-init -v

   $ sudo systemctl is-enabled cloud-init.service

   $ sudo systemctl is-enabled cloud-init-local.service
   ```
3. Configure cloud-init to accept the OVF data source for VMware raw cloud-init configuration.
   Edit the cloud-init configuration file in `/etc/cloud/cloud.cfg` or in `/etc/cloud/cloud.cfg.d/*`, depending on the Linux distribution. Add `OVF` to the list of datasources accepted by cloud-init.

   Example line for cloud-init version 21.3:
   ```
   datasource_list: [ NoCloud, ConfigDrive, Azure, OVF, OpenStack, Ec2 ]
   ```
4. Configure cloud-init to enable VMware customization with raw cloud-init data.

Edit the cloud-init configuration file in `/etc/cloud/cloud.cfg` or in `/etc/cloud/cloud.cfg.d/*`, depending on the Linux distribution. Set the flag `disable_vmware_customization` to `true` and set the flag `allow_raw_data` to `true`. If customization time-out is an issue, you can set the value of `vmware_cust_file_max_wait` (default: 15 seconds).

Example lines for cloud-init version 21.3:

```
disable_vmware_customization: true

datasource:

  OVF:

    allow_raw_data: true

vmware_cust_file_max_wait: 25
```

5. Run the cloud-init clean step to remove any artifacts from previous configuration failures.
   ```
   $ sudo cloud-init clean
   ```
6. Power off the virtual machine template.
   ```
   $ sudo poweroff
   ```

Prepare a cloud-init Metadata File

Use the following information to prepare your cloud-init metadata file. The metadata file is required for cloud-init.

- The metadata file is a plain-text file formatted as YAML or JSON.
- The maximum file size is 512KB.
- All standard `v1` keys are accepted, such as:
  - `distro`
  - `local_hostname`
  - `platform`
  - `public_ssh_keys`
- The `instance-id` key is required. All other keys are optional.
- For network configuration, both `v1` and `v2` keys are accepted.

This example shows a metadata file in YAML format that includes a minimal network configuration:

```
instance-id: cloud-vm-example-1

local hostname: cloud-vm

network:

  version: 2

  ethernets:

    nics:

      match:

        name: eth*

      dhcp4: yes
```

This example shows a metadata file in JSON format that specifies a fixed IP network configuration:

```
{

  "instance-id": "cloud-vm-example-2",
```

```
  "local-hostname": "cloud-vm",
  "cloud-name": "vSphere",
  "distro": "ubuntu",
  "distro-version": "20.04",
  "distro-release": "focal",
  "network": {
    "version": 2,
    "ethernets": {
      "nics": {
        "match": {
          "name": "eth*"
        },
        "nameservers": {
          "addresses": ["203.0.113.1"],
          "search": ["www.example.com", "example.com"]
        },
        "gateway4": "192.0.2.253",
        "dhcp4": false,
        "addresses": [ "192.0.2.0/24" ]
      }
    }
  }
}
```

Prepare a cloud-init Userdata File

Use the following information to prepare your cloud-init userdata file. The userdata file is optional for cloud-init.

- The userdata file is a plain-text file in raw cloud-init format, without compression or base64 encoding.
- The maximum file size is 512KB.
- Userdata configuration can be used to accomplish a number of configuration tasks in the guest operating system. For instance, some of the more common tasks include:
  - Add user accounts
  - Install software
  - Run scripts
  - Import SSH keys

This example shows a userdata file that adds a new user to the guest operating system:

```
#cloud-config


users:
  - default
  - name: testuser
    primary_group: testgroup
    sudo: ALL=[ALL] NOPASSWD:ALL
    groups: sudo, wheel
    ssh_import_id: None
    lock_passwd: true
    ssh_authorized_keys:
    - ssh-rsa myRsaPubKey
```

Initiate Raw Data cloud-init Customization with the vSphere Web Services API

Use the following steps to initiate customization with raw cloud-init data by using the vSphere Web Services API.

1. Build a customization specification of type `CustomizationSpec`. Set the following properties in the spec:

   a. The `globalIPSettings` property must be present but need not contain its optional child properties. If the child properties are present, they will be ignored during the method call.
   b. The `identity` property must be present and must have type `CloudinitPrep`.
      – The child property `identity.metadata` is required. It contains the entire text of the metadata file, less any final new line character.
      – The child property `identity.userdata` is optional. If present, it contains the entire text of the userdata file, less any final new line character.

   Example:

   ```
   CloudinitPrep cloudinit = new CloudinitPrep();

   cloudinit.setMetadata(metadataText);

   cloudinit.setUserdata(userdataText);

   CustomizationGlobalIPSettings ipSettings = new CustomizationGlobalIPSettings();

   CustomizationSpec custSpec = new CustomizationSpec();

   custSpec.setIdentity(cloudinit);

   custSpec.setGlobalIPSettings(ipSettings);
   ```

   NOTE
   The `nicSettingsMap` property, if present, will be ignored for raw data customization. The cloud-init tool accepts network customization only from the `identity.metadata` property.

2. Invoke the method `VirtualMachine.CustomizeVM_Task`, passing the `customizationSpec` as parameter. The method returns a Task object that you can use to monitor completion of the customization steps.

Example:
```
ManagedObjectReference custTask = service.CustomizeVM_Task(vmRef, custSpec);
```

3. Power on the virtual machine. As it boots, cloud-init applies the metadata configuration settings to the guest operating system. After the guest finishes booting and before it opens a shell window, cloud-init applies the userdata configuration.

Guest Network Customization for Instant Clone Virtual Machines

You can customize the network settings in the guest operating system with the help of the vSphere Web Services SDK. For instant clone operations, you must customize the clone without stopping the guest operating system.

Guest network customization for a running virtual machine involves several steps:
- Installation of the guest customization engine
- Disconnecting virtual NICs
- Customizing guest network settings
- Reconnecting virtual NICs
- Restarting the guest network
- Recovering from guest customization errors
- Running scripts for application-dependent customization

You can use these steps independently and you can run your own scripts in between steps as needed to customize, reset status, or restart processes in the guest.

> **NOTE**
> You must authenticate using credentials accepted by the guest operating system before you can run guest operations.

Installing the Guest Customization Engine

The guest customization engine needs to be installed in a virtual machine before using the `VirtualMachineGuestCustomizationManager.CustomizeGuest_Task` method.You can download the installer from the VMware Customer Connect web site.

Start a web browser in a running virtual machine. For instant clone operations, prepare the source virtual machine by installing the customization engine before creating clones.

The guest customization engine is available for most popular Linux distributions, using either an RPM package or a DEB package. To download and install it within the guest operating system, use the following steps.

1. Login to Broadcom support site and find VMware.
2. Navigate to the Download VMware vSphere page.
3. Click the **Drivers & Tools** tab.
4. Find the entry for the instant clone customization engine.
5. Download the installation package appropriate for your Linux distribution:
   - The `.rpm` file for RHEL or SLES
   - The `.deb` file for Ubuntu
6. Run the command to install the package in the guest:
   - For the `.rpm` package, use this command:
     ```
     sudo /bin/rpm -i --force yourdpkgFilePath >> ./rpm.log 2>&1
     ```
   - For the `.deb` package, use this command:
     ```
     sudo /usr/bin/dpkg -i --force-all yourdpkgFilePath >> ./dpkg.log 2>&1
     ```

After the guest customization engine is installed, the guest is prepared to invoke `VirtualMachine.InstantClone_Task` or `VirtualMachineGuestCustomizationManager.CustomizeGuest_Task`. You can delete the installation package to save virtual disk space.

Disconnecting Virtual NICs

Before you customize the guest network, you should disconnect the virtual NICs to quiesce network traffic during the operation. Disconnecting virtual NICs is a key step in the instant clone process.

You can disconnect a virtual NIC in one of these ways:

- In the `spec` parameter to the `InstantClone_Task` method you can specify that the virtual NIC should be disconnected during the instant clone operation. This way leaves the NIC in the source virtual machine active immediately after the clone operation, while the NIC in the clone is idle and ready for customization.
- In the `spec` parameter to the `ReconfigVM_Task` method you can change the connection state of a virtual NIC. This way is available either with or without an instant clone operation.

Disconnecting a Virtual NIC with InstantClone_Task

To disconnect a virtual NIC during an instant clone operation, prepare a `VirtualMachineInstantCloneSpec` data object that specifies the device should be in the disconnected state after the clone operation completes. You pass the clone specification in the `spec` parameter of `InstantClone_Task`. Assign a value of `disconnect` to the property `spec.location.deviceChange[].device.connectable.migrateConnect`.

Disconnecting a Virtual NIC with ReconfigVM_Task

To disconnect a virtual NIC by reconfiguring the virtual machine, prepare a `VirtualMachineConfigSpec` data object that specifies a disconnected virtual NIC. You pass the config specification in the `spec` parameter of the `ReconfigVM_Task` method. Assign a value of `False` to the property `spec.deviceChange[].device.connectable.connected`.

Customizing Guest Network Settings for Running Virtual Machines

You can use a method of the `VirtualMachineGuestCustomizationManager` managed object to apply new settings to the virtual NICs in the guest. The `VirtualMachineGuestCustomizationManager` uses the guest customization engine designed for instant clone operations to customize the guest while it continues running.

The customization step applies new settings for IP address, DNS server, and gateway server of one or more virtual NICs.

> **NOTE**
> DNS settings can be specific to individual NICs in Windows operating systems. DNS settings are global in a Linux operating system.

The guest customization engine must be installed in the virtual machine before you call this method. See Installing the Guest Customization Engine for instructions.

Customization applies the new guest network settings that you specify in the `spec` parameter to the `VirtualMachineGuestCustomizationManager.CustomizeGuest_Task` method. The `spec` parameter is a data object of type `CustomizationSpec`, which contains a `nicSettingMap` property that holds an array that maps MAC address to network settings for one or more virtual NICs in the guest.

There are two ways to use the `nicSettingsMap` array in the `spec` parameter. You can do one of the following:

- Specify each virtual NIC explicitly by its MAC address, in `spec.nicSettingMap[i].macAddress`, and specify the virtual NIC's new settings in the properties of `spec.nicSettingMap[i].adapter`.

- Omit `spec.nicSettingMap[i].macAddress` for all virtual NICs and specify the new settings for each NIC in PCI bus order.

In addition to virtual NIC settings, you can customize these properties of a virtual machine:
- You can set the `CustomizationSpec.identity` property to a data object derived from the type `CustomizationIdentitySettings`. You must choose a subclass of the type that corresponds to the guest operating system installed in the virtual machine. In the `identity` object you can specify a new host name, domain name, and system time settings.
- You can set the `CustomizationSpec.globalIPSettings` property to a data object of type `CustomizationGlobalIPSettings` in which you specify DNS settings for all virtual NICs. Use this for Linux guest operating systems.

The `CustomizeGuest_Task` method supports the following guest operating systems:
- Red Hat Enterprise Linux 6.8 and higher
- Red Hat Enterprise Linux 7.4 and higher
- CentOS 7.4 and higher
- SUSE 11SP4
- SUSE 12SP3 and higher
- Ubuntu 16.04 and higher (LTS distributions)

The toolsd service must be running and available for this customization step.

NOTE
The `CustomizeGuest_Task` method is asynchronous, but a critical part of the operation, which uses the customization engine, is synchronous.

You can pipeline the instant clone and customization operations to minimize the time needed to clone a virtual machine.

NOTE
During an instant clone operation, the toolsd service is briefly offline while it resets connections in the clone. This results in no noticeable delay, but you might need to allow for a retry if you run pre-customization code.

The customization data is stored in the Namespace database in a cust.cfg format containing name-value pairs. This makes the customization process resilient to vMotion operations. It also means that you can access the data if needed as input for user-supplied customization scripts.

Reconnecting Virtual NICs in a Running Virtual Machine

After you customize a guest network in a running virtual machine, you must reconnect the virtual NICs before the guest operating system can use them.

To reconnect a virtual NIC by reconfiguring the virtual machine, prepare a `VirtualMachineConfigSpec` data object that specifies a connected virtual NIC. You pass the config specification in the `spec` parameter of the `ReconfigVM_Task` method. Assign a value of `True` to the property `spec.deviceChange[].device.connectable.connected`.

Restarting the Guest Network After Customization

After customizing guest network settings, you can restart the guest network manually from within the guest operating system, or by using a method of the `VirtualMachineGuestCustomizationManager` managed object. This method is important for instant clone operations, but it can be used for other purposes as well.

To restart the guest network manually, see Resetting the Network Stack in a Running Virtual Machine.

To restart the guest network using the vSphere Web Services API, use the `VirtualMachineGuestCustomizationManager.StartGuestNetwork_Task` method. This method both reconnects the virtual NICs to the network and restarts the guest network service.

The `StartGuestNetwork_Task` method is supported on most popular Linux distributions.

Recovering from Guest Network Customization Errors

Guest network customization is a long-running operation with several potential failure points. If you encounter a failure, consider the following steps.

The guest network customization engine holds a lock on the `VirtualMachineGuestCustomizationManager` interface while it applies changes to the guest. A failure during customization or while restarting the network might leave a hung process or an orphaned lock.

If any step of the network cutsomization returns an error, use the `VirtualMachineGuestCustomizationManager.AbortCustomization_Task` method to stop all running instances of the customization engine and to release the lock.

To troubleshoot a failure, you can examine the log file used by the `VirtualMachineGuestCustomizationManager.CustomizeGuest_Task` method. The log file is on the host file system at `/var/log/vmware-gosc/instant_clone_customization.log` for Linux hosts.

When you abort a guest customization operation, the guest network can be left in an inconsistent state. After the abort operation completes, you should check your customization settings and retry the `CustomizeGuest_Task` method in case of transient error conditions.

> **NOTE**
> When you use the `VirtualMachineGuestCustomizationManager`, the customization procedure tolerates vMotion without failure.

Application-Dependent Customization

After you customize and restart the guest network, the virtual machine might need additional customization steps, depending on the applications that run in the guest operatin system. For example, you might need to generate new encryption keys for an instant clone, or signal an application to close and re-open network sockets when you have configured new network IP addresses.

To do additional customization steps in the guest, use the `GuestFileManager` and the `GuestProcessManager` interfaces in the `GuestOperationsManager` managed object.

Resetting the Network Stack in a Running Virtual Machine

In any case where you change the network identity settings in the guest operating system of a running virtual machine, you must reset the network stack for the virtual NIC to cause it to adopt the new settings. You can do this by invoking the `VirtualMachineGuestCustomizationManager.RestartNetwork_Task` method after you customize the network settings, or you can do the reset from the guest operating system, as with a physical machine. The way to reset a NIC differs, depending on the operating system.

For information about the `RestartNetwork_Task` method, see Restarting the Guest Network After Customization.

Resetting the Network Stack in a Linux Virtual Machine

When you change network settings in an instant clone, you must reset the network stack of the virtual NIC to adopt the new settings. For a Linux guest operating system, you must unbind and rebind the network driver so it adopts a new MAC address.

The following shell script rebinds all network drivers:

```
#!/bin/bash

for NETDEV in /sys/class/net/*

do

  DEV_LABEL=`basename \`readlink -f $NETDEV/device\``

  DEV_DRIVR=`basename \`readlink -f $NETDEV/device/driver\``

  echo $DEV_LABEL > /sys/bus/pci/drivers/$DEV_DRIVR/unbind

  echo $DEV_LABEL > /sys/bus/pci/drivers/$DEV_DRIVR/bind

done
```

If the guest operating system was configured to use DHCP, you must also force a DHCP refresh after the MAC addresses refresh has completed.

Tip: Avoid resetting the network stack in the guest operating system while you are working over a network connection, such as SSH. You can run the shell commands to reset the stack as part of a customization script installed in the guest operating system of the source virtual machine before running the instant clone operation.

Resetting the Network Stack in a Windows Virtual Machine

When you change network settings in a Windows virtual machine, you must disable and re-enable the network object. There are several ways to do that in Windows.

Reset the Network Stack in a Windows Virtual Machine Using the Control Panel

When you change network settings in a Windows virtual machine, you must reset the network stack of the virtual NIC to adopt the new settings. For a Windows guest operating system, you must disable and re-enable the network connection so it adopts a new MAC address. You can do this using the Control Panel.

1. In the Control Panel, navigate to **Control Panel › Networks and Internet › Network Connections**.
2. Right-click the network adapter icon and choose **disable**.
3. Right-click the network adapter icon and choose **enable**.
4. Repeat these steps for each network adapter.

Reset the Network Stack in a Windows Virtual Machine Using PowerShell

When you change network settings in a Windows virtual machine, you must reset the network stack of the virtual NIC to adopt the new settings. For a Windows guest operating system, you must disable and re-enable the network connection so it adopts a new MAC address. To perform this task with PowerShell, use the `Restart-NetAdapter` cmdlet.

The cmdlet is described in more detail at https://technet.microsoft.com/en-us/itpro/powershell/windows/netadapter/restart-netadapter.

1. List the virtual network adapters in the guest operating system.
   ```
   Get-NetAdapter
   ```

The names of the virtual network adapters appear in the first column of output.

2. Restart the network adapter representing the external interface.
   `Restart-NetAdapter -Name "Ethernet0"`
3. If you have more than one virtual interface, repeat the previous step for each one.

Reset the Network Stack in a Windows Virtual Machine Using netsh

When you change network settings in a Windows virtual machine, you must reset the network stack of the virtual NIC to adopt the new settings. For a Windows guest, you must disable and re-enable the network connection so it adopts a new MAC address. To perform this task, run `netsh` with `interface` command.

Run the following commands in a command prompt (Run > **`cmd`**).

1. List the network interfaces.
   `netsh interface show interface`
2. Choose the interface that represents the external network connection and disable it.
   `netsh interface set interface name="Ethernet0" admin=DISABLED`
3. Re-enable the interface with a similar command.
   `netsh interface set interface name="Ethernet0" admin=ENABLED`
4. Repeat the previous two commands if you have more than one NIC that needs to be reset.

Virtual Machine Security

Encryption in vSphere protects virtual machines, virtual disks, and related files. First you set up a trusted connection between vCenter Server and a key management server (KMS), then vCenter Server can retrieve keys from the KMS as needed. Physical device security and advanced options are additional concerns.

Various aspects of virtual machine encryption are handled differently. You manage setup of the KMS trusted connection and perform most encryption workflows from the vSphere Client. You manage automation of some advanced features using the vSphere Web Services SDK, as discussed in this chapter. You use the **`crypto-util`** command-line tool directly on ESXi hosts for some special cases, for example, to decrypt the core dumps in a vm-support bundle.

How Virtual Machine Encryption Protects a Datacenter

With vSphere virtual machine encryption, you can create encrypted virtual machines and encrypt existing ones. Because all virtual machine files with sensitive information are encrypted, the virtual machine is protected. Only administrators with encryption privileges can perform encryption and decryption tasks.

What Keys are Used

Two types of keys are used for encryption.

- The ESXi host generates and uses internal keys to encrypt virtual machines and disks. These keys are used as the disk encryption key (DEK) and are XTS-AES-256 keys.
- The key management server (KMS) sends keys to the vCenter Server upon request. These keys are used as the key encryption key (KEK) and are AES-256 keys. vCenter Server stores only the ID of each KEK, but not the key itself.
- ESXi hosts use the KEK to encrypt their internal keys, and store only the encrypted internal keys on disk, but not the KEK itself. When an ESXi host reboots, vCenter Server requests the necessary KEKs by sending the corresponding IDs to the KMS, and upon receipt, make the KEKs available to the ESXi host, which can then decrypt its internal keys as needed.

What Is Encrypted

Virtual machine encryption supports encrypting virtual machine files, virtual disk files, and core dump files.

Virtual Machine Files

Most virtual machine files, in particular guest data that are not stored in the VMDK file, are encrypted. This set of files includes but is not limited to the NVRAM (memory), VSWP (swap), and VMSN (snapshot) files. The key that vCenter Server retrieves from the KMS unlocks an encrypted bundle in the VMX file that contains internal keys and other secrets.

If you use the vSphere Client to create an encrypted virtual machine, all virtual disks are encrypted by default. For other encryption tasks, such as encrypting an existing virtual machine, you can encrypt and decrypt virtual disks separate from virtual machine files.

> **NOTE**
> You cannot associate an encrypted virtual disk with an unencrypted virtual machine.

Virtual Disk Files

Data in an encrypted virtual disk (VMDK) file are never written in cleartext to storage or physical disk, and is never transmitted over the network in cleartext. The VMDK descriptor file is mostly cleartext, but contains a key ID for the KEK and the internal key (DEK) in the encrypted bundle.

You can use the vSphere API to perform either a shallow recrypt operation with a new KEK, or a deep recrypt operation with new internal keys.

Core Dump Files

Core dumps on an ESXi host that has encryption mode enabled are always encrypted. You can decrypt and password protect ESXi core dumps using the `crypto-util` command-line tool on the ESXi host.

> **NOTE**
> Core dumps on the vCenter Server (Appliance) are not encrypted. Be sure to protect access to all vCenter Server systems.

What Is Not Encrypted

Some files that are associated with a virtual machine are not encrypted or partially encrypted.

Log Files

Log files are not encrypted because they should not contain sensitive data.

Virtual Machine Configuration Files

Most of the virtual machine configuration information, stored in the VMX and VMSD files, is not encrypted. Information about the KMS and the key ID is visible in those files.

Virtual Disk Descriptor File

To support disk management without a key, most of the virtual disk descriptor file is not encrypted.

Who Can Perform Cryptographic Operations

Only users who are assigned the Cryptographic Operations privileges can perform cryptographic operations. The privilege set is fine grained; see the *vSphere Security* guide. The default Administrator system role includes all Cryptographic Operations privileges. A new system role, No Cryptography Administrator, supports all Administrator privileges except for the Cryptographic Operations privileges.

You can create additional custom roles, for example, to allow a group of users to encrypt virtual machines but to prevent them from decrypting virtual machines.

For a full list of privileges, see the section “Cryptographic Operations Privileges” in the *vSphere Security* manual.

How Can I Perform Cryptographic Operations

The vSphere Client supports many cryptographic operations. For other tasks, you must use the API.

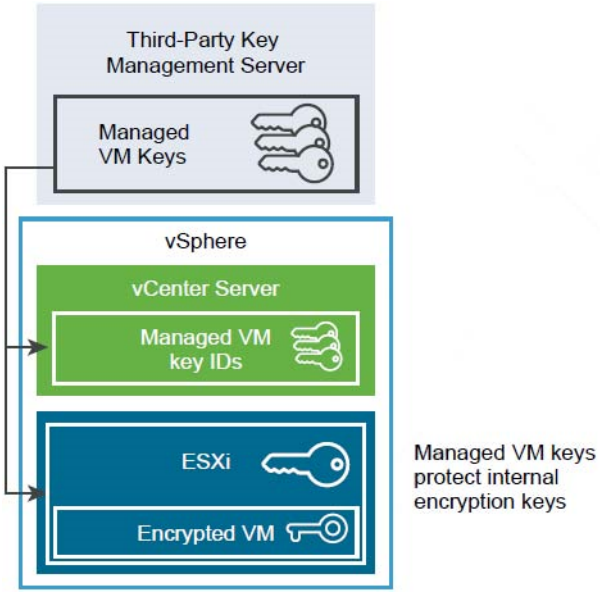
Table 8: Interfaces for performing cryptographic operations

Interface	Operations	Information
vSphere Client	Create encrypted virtual machines Encrypt and decrypt virtual machines	The *vSphere Security* guide.
vSphere Web Services SDK	Create encrypted virtual machines Encrypt and decrypt virtual machines Perform a deep recrypt of virtual machines (use a different KEK and DEK). Perform a shallow recrypt of virtual machines (use a different KEK).	This chapter.
`crypto-util`	Decrypt encrypted core dumps, check whether files are encrypted, and perform other management tasks directly on the ESXi host,	Command-line help and the *vSphere Security* guide.

vSphere Virtual Machine Encryption Components

An external key management server (KMS), the vCenter Server system, and ESXi hosts all contribute to the vSphere virtual machine encryption solution.

Figure 37: vSphere Virtual Machine Encryption Architecture



Key Management Server

The vCenter Server requests keys from an external KMS. The KMS generates and stores the keys, and passes them to vCenter Server for distribution.

You can use the vSphere Client or the vSphere API to add a cluster of KMS instances to the vCenter Server system. If you use multiple KMS instances in a cluster, all instances must be from the same vendor and must replicate keys.

If your environment uses different KMS vendors in different environments, you can add a KMS cluster for each KMS and specify a default KMS cluster. The first cluster that you add becomes the default cluster. You can explicitly change the default later.

As a KMS client, vCenter Server uses the Key Management Interoperability Protocol (KMIP) and makes it easy to use the KMS of your choice.

vCenter Server

Only vCenter Server has credentials for logging in to the KMS. The ESXi hosts do not have those credentials. The vCenter Server obtains keys from the KMS and pushes them to the ESXi hosts. The vCenter Server does not store the KMS keys, it merely keeps a list of key IDs.

The vCenter Server checks the privileges of users who perform cryptographic operations. You can use the vSphere Client to assign cryptographic privileges or to assign the No cryptography administrator custom role to groups of users. See Prerequisites and Required Privileges for Encryption Tasks.

The vCenter Server adds cryptography events to the list of events that you can view and export from the vSphere Client Event Console. Each event includes the user, time, key ID, and cryptographic operation.

ESXi Hosts

ESXi hosts are responsible for several aspects of the encryption workflow.

- Accepting keys and storing them in memory (never on disk). If a host has encryption mode enabled, and the current user's role includes cryptographic operations privileges, vCenter Server pushes keys to the ESXi host upon request. See Prerequisites and Required Privileges for Encryption Tasks.

- Ensuring that guest data for encrypted virtual machines are encrypted when stored on disk.
- Ensuring that guest data for encrypted virtual machines are never sent unencrypted over the network.

The keys that the ESXi host generates are called internal keys in this document. These keys are typically act as data encryption keys (DEKs).

Encryption Process Flow

After vCenter Server is connected to the KMS, users with the required privileges can create encrypted virtual machines and disks. Those users can also perform other encryption tasks such as encrypting existing virtual machines and decrypting encrypted virtual machines. During the encryption process, different components interact as follows.

1. When the user performs an encryption task, for example, creating an encrypted virtual machine, vCenter Server requests a new key from the default KMS. This key will be used as the KEK.
2. The vCenter Server stores the key ID and passes the key to an ESXi host. If the ESXi host is part of a cluster, vCenter Server sends the KEK to each host in the cluster. The key itself is not stored on the vCenter Server system. Only the key ID is stored.
3. The ESXi host generates internal keys (DEKs) for the virtual machine and its disks, using the KEK that it received from vCenter Server to encrypt the internal keys. The the internal keys are kept in memory only. Only the encrypted data are stored on disk.
4. The ESXi host encrypts the virtual machine and its disks with the encrypted internal key.

ESXi hosts that have the KEK and can access the encrypted key file can perform operations on an encrypted virtual machine or disk. Because they come from the KMS, ESXi hosts can use the same KEK across reboots.

If you later want to decrypt a virtual machine, you change its storage policy either for the virtual machine or for its disks. If you want to decrypt individual components, first decrypt selected disks, then decrypt the virtual machine by changing the storage policy for VM Home. Both keys are required for decryption of each component, virtual disks or VM Home.

When you encrypt an existing virtual machine, you need at least twice the space that the virtual machine is currently using, in most cases.

Prerequisites and Required Privileges for Encryption Tasks

Users who perform encryption related tasks must have the appropriate privileges. Additional privileges are required if virtual machine encryption tasks require changing the host encryption mode. An extensive number of Cryptographic Operations privileges allow fine-grained control.

Encryption tasks are possibly only in environments that include a vCenter Server. Additionally, the ESXi host must have encryption mode enabled for most encryption tasks. The user who performs an encryption task must have the appropriate privileges. Additional privileges are required if virtual machine encryption tasks require changing the host encryption mode. An extensive number of Cryptographic Operations privileges allow fine-grained control.

Cryptography Privileges and Roles

By default, the user with the vCenter Server Administrator role has all Cryptographic Operations privileges. You can assign the No cryptography administrator role to all vCenter Server administrators who do not need cryptographic privileges.

The user with the vCenter Server Administrator role has all privileges by default. You can assign the No cryptography administrator role to vCenter Server users who do not need Cryptographic Operations privileges. The No cryptography administrator lacks the following privileges for cryptographic operations:

- Add Cryptographic Operations privileges
- Global.Diagnostics
- Host.Inventory.Add host to cluster
- Host.Inventory.Add standalone host

- Host.Local operations.Manage user groups

To further limit what users can do, you can clone the No cryptography administrator role and create a custom role with only some of the Cryptographic Operations privileges. For example, you can create a role that allows users to encrypt but not to decrypt virtual machines, or that does grant privileges for management operations. See the *vSphere Security* manual for details.

Host Encryption Mode

You can encrypt virtual machines only if host encryption mode is enabled for the ESXi host. Host encryption mode is often enabled automatically, but it can be enabled explicitly.

You can check and explicitly set the current host encryption mode from the vSphere Client or by using the vSphere API; see API Methods to Prepare an ESXi Host.

After host encryption mode is enabled, it cannot be disabled easily. See the *vSphere Security* guide for details.

Automatic changes occur when encryption operations attempt to enable host encryption mode. For example, suppose that you add an encrypted virtual machine to an ESXi host, and host encryption mode is not enabled. If you have the required privileges on the host, encryption mode automatically changes to enabled.

Assume a cluster that includes three ESXi hosts, host A, B, and C. You add an encrypted virtual machine to host A. What happens depends on several factors. If all three hosts have encryption enabled, you can create an encrypted virtual machine if you have `Encrypt new` privileges. If none of the hosts has encryption enabled, and you have `Register host` privileges on host A, then the virtual machine creation process enables host encryption on that host; otherwise an error results. The scenario is more complicated if host B or C is not enabled for encryption; see the *vSphere Security* guide for details.

Encrypted vSphere vMotion

vSphere vMotion always tries to use encryption when migrating encrypted virtual machines. You cannot disable encrypted vMotion for encrypted virtual machines in a cluster.

For virtual machines that are not encrypted, you can set encrypted vMotion to `Opportunistic` (use encrypted vMotion if supported) or `Required` (do not migrate if unsupported). See Encrypted vSphere vMotion.

API Methods for vSphere Virtual Machine Encryption

Methods for managing vSphere virtual machine encryption are in the vSphere API, part of the Web Services SDK. In addition, you need a few Storage Policy APIs to create and assign encryption storage policies.

vSphere API Methods for KMS Management

Cryptographic key management interfaces are defined in the following hierarchy.

- CryptoManager – managed object, parent of CryptoManagerKmip. CryptoManager defines methods that are covered in vSphere API Methods for Cryptographic Operations.
  - CryptoManagerKmip – managed object for handing the cryptographic keys.
    - CertificateInfo – basic information of a certificate.
    - ClusterStatus – status of a KMS cluster.
    - ServerCertInfo – information about the KMS certificate.
    - ServerStatus – status of a KMS.

    CryptoManagerKmip defines the following methods:
  - **`String`**`generateClientCsr`**`(cluster)`** – generate a certificate signing request with its private key.
  - **`String`**`generateSelfSignedClientCert`**`(cluster)`** – generate self-signed client certificate with its private key.

  - **KmipClusterInfo[]** kmipServers **()** – get a list of registered KMS, grouped by clusters.
  - **KmipClusterInfo[]** listKmipServers **(limit)** – get a list of registered KMS.
  - **void** markDefault **(clusterId)** – set the default KMS cluster.
  - **void** registerKmipServer **(server)** – register a KMS.
  - **void** removeKmipServer **(clusterId,serverName)** – remove a KMS, even if in use.
  - **String** retrieveClientCert **(cluster)** – get the client certificate of the KMS cluster.
  - **String** retrieveClientCsr **(cluster)** – get the generated client certificate signing request.
  - **ServerCertInfo** retrieveKmipServerCert **(keyProvider,server)** – get the server certificate.
  - **ClusterStatus[]** retrieveKmipServersStatus **(clusters[])** – get status of the KMS instances.
  - **String** retrieveSelfSignedClientCert **(cluster)** – get generated self signed client certificate.
  - **void** updateKmipServer **(server)** – update a KMS.
  - **void** updateKmsSignedCsrClientCert **(cluster,certificate)** – Set KMS signed certificate as the client certificate for a KMS cluster.
  - **void** updateSelfSignedClientCert **(cluster,certificate)** – set a self-signed certificate as the client certificate for the KMS cluster.
  - **void** uploadClientCert **(cluster,certificate, privateKey)** – set a client certificate with private key for the KMS cluster.
  - **void** uploadKmipServerCert **(cluster,certificate)** – upload a server certificate.
  - **CryptoKeyResult** generateKey **(keyProvider)** – generate a new encryption key. (The **generateKey** method is out of alphabetic order because it is called after the others.)

Certificate Management

Administrators can use the vSphere Client to manage certificates. Some organizations require a third party certificate externally but use VMware Certificate Authority (VMCA) internally. `CertificateManager` is the managed object to reach a certificate authority and refresh or revoke certificates. It includes these methods:

- `refreshCACertificatesAndCRLs` – Fetch certificates and CSR (revocation lists) from trusted certificate authorities and push them to hosts.
- `refreshCertificates` – Get CSRs and certificates signed by VMCA and push them down to hosts.
- `revokeCertificates` – Revokes certificates of specified hosts.

API Methods to Prepare an ESXi Host

Encryption interfaces for ESXi hosts are additions to a previously existing managed object, `HostSystem`.

- `HostSystem` – managed object providing access to the ESXi host, including these additions:
  - `HostCryptoState` – enumeration indicating whether the ESXi host is incapable of, prepared for encryption mode, or cryptography safe with its host key already set. Safe means that the ESXi host can encrypt virtual machines and will not leak keys.
  - `ConfigureCryptoKey` – vCenter Server calls this method to set or change the key used for core dump encryption, and to place the host in safe mode. It is like calling `PrepareCrypto` and `EnableCrypto` in quick succession. Not supported if called on an ESXi host. As of vSphere 7.0 you can disable encryption on a host, after next reboot, by setting `pendingIncapable` in `HostCryptoState`.
  - `PrepareCrypto` – this method puts the ESXi host in prepared mode so it can receive sensitive data, after being enabled.
  - `EnableCrypto` – this method sets or changes the key for core dump encryption and puts the ESXi host in safe mode. It must be called in sequence after `PrepareCrypto`.

  The following data objects are used by the `ConfigureCryptoKey` and `EnableCrypto` methods above, and by the CryptoManager methods below in vSphere API Methods for Cryptographic Operations.

  - `CryptoKeyPlain` – data object representing a plain text cryptographic key.

  - `CryptoKeyId` – data object representing a cryptographic key.
  - `CryptoKeyResult` – data object representing the result of a cryptographic key operation.

vSphere API Methods for Cryptographic Operations

Cryptographic operations are defined in the following hierarchy.

- `CryptoManager` – managed object for handling cryptographic keys.
  `CryptoManager` defines the following methods:
  - **`void`** `addKey`**`(CryptoKeyPlain key)`** – add plain key to the vCenter Server.
  - **`CryptoKeyResult[]`** `addKeys`**`(CryptoKeyPlain[] keys)`** – add multiple plain keys to vCenter.
  - **`boolean`** `enabled`**`()`** – indicate if the encryption feature is enabled.
  - **`CryptoKeyId[]`** `listKeys`**`(int limit)`** – list keys.
  - **`void`** `removeKey`**`(CryptoKeyId key, boolean force)`** – remove a key (only its ID is needed).
  - **`CryptoKeyResult[]`** `removeKeys`**`(CryptoKeyId[] keys, boolean force)`** – multiple keys.
- `VirtualMachineConfigSpec` – previously existing data object passed as parameter to `CreateVM_Task` and `ReconfigVM_Task`. One of its newly added properties is crypto, a CryptoSpec with one of the following options, which is inherited by all virtual disks and virtual machine configuration files (VM home).
  - `CryptoSpecEncrypt` – indicates that the virtual machine should be encrypted.
  - `CryptoSpecDecrypt` – indicates that the virtual machine should be decrypted.
  - `CryptoSpecDeepRecrypt` – indicates that all KEKs and DEKs should be replaced.
  - `CryptoSpecShallowRecrypt` – indicates that only KEKs should be replaced.
  - `CryptoSpecNoOp` – indicates that encryption settings should not be changed.
  - `CryptoSpecRegister` – indicates that the operation should send keys but should not modify the encryption settings of the virtual machine or virtual disk. When an encrypted disk is hot attached, the program must pass `CryptoSpecRegister` with the key ID that encrypted the disk. The key can be obtained from the Datastore Browser.

  These data objects are informational properties of `VMConfigFileInfo` and `VMDiskFileInfo`, respectively. They can be used to check whether the VM home and its virtual disks are encrypted.
  - `VmConfigFileEncryptionInfo` – the encryption information of a virtual machine configuration.
  - `VmDiskFileEncryptionInfo` – the encryption information of a virtual disk.

  The enumeration `EncryptedVMotionModes` controls whether encrypted vMotion is `disabled`, `required`, or `opportunistic` (fall back to unencrypted vMotion if necessary, the default option).

More information about the encryption interfaces is available in vSphere Management SDK Documentation under vSphere Web Services SDK, in the *`vSphere API Reference`*.

SPBM API Methods for Encryption

The managed object PbmProfileProfileManager (**`pbm.profile.ProfileManager`**) provides methods to set encryption related storage policies, including create and update.

- **`PbmCreate(createSpec)`** – create a capability-based storage profile defining storage requirements.
- **`PbmUpdate(profileId, createSpec)`** – update a storage profile with new capability requirements.

Workflows for vSphere Virtual Machine Encryption

The sections below present vSphere API methods for key acquisition and encryption.

Set Up the Key Management Server Cluster

Before you can start encrypting virtual machines, you must set up the Key Management Server (KMS) cluster. This includes adding the KMS, establishing trust with the KMS, and setting the default KMS cluster. The KMS must support the KMIP (Key Management Interoperability Protocol) 1.1 standard, and it must be capable of acting as a symmetric key foundry and server.

Add Key Management Server(s) to Your Datacenter

Your organization's security administrator may be responsible for this task. This person will need to provide the cluster name (if it already exists), the KMS server name, its IP address, its connection port number, and possibly a proxy address, proxy port number, KMS user name, and corresponding password.

The `CryptoManager.java` code to add a KMS and set the default KMS cluster appears in CryptoManager Java program to add KMS and set default cluster. The user running this program must have Cryptographic operations.Manage key servers privileges.

Establish a Trusted Connection by Exchanging Certificates

There are four ways to establish trust between a KMS and vCenter Server. Different KMS vendors require different ways.

Java examples for these four ways follow.

1. Upload your Root CA Certificate to the KMS. Obtain it manually and upload with the self-signed call.
2. Upload a self-signed vCenter Certificate to the KMS.
3. Have vCenter Server generate a certificate signing request (CSR), which the KMS signs and returns.
4. Download a security certificate and private key generated by the KMS to vCenter Server.

CryptoManager Java program to add KMS and set default cluster

```
package com.vmware.general;

import com.vmware.common.annotations.Action;

import com.vmware.common.annotations.Option;

import com.vmware.common.annotations.Sample;

import com.vmware.connection.ConnectedVimServiceBase;

import com.vmware.vim25.CryptoKeyId;

import com.vmware.vim25.CryptoKeyResult;

import com.vmware.vim25.CryptoManagerKmipServerCertInfo;

import com.vmware.vim25.KeyProviderId;

import com.vmware.vim25.KmipServerInfo;

import com.vmware.vim25.KmipServerSpec;

import com.vmware.vim25.ManagedObjectReference;

import com.vmware.vim25.RuntimeFaultFaultMsg;

/**
```

```
 * CryptoManager. Demonstrates uses of the CryptoManager API. Parameters:
 * url      [required] : web service url, for example https://10.9.8.7/sdk
 * username [required] : username for the authentication
 * password [required] : corresponding password
 * Command line to run CryptoManager code:
 * run.bat com.vmware.general.CryptoManager ^
 *         --url webserviceurl --username name --password pass ^
 *         --kmsname kms --kmsip ipaddr --kmsclusterid providerId
 */
@Sample(name = "CryptoManager", description = "Demonstrates uses of the CryptoManager
API")
public class CryptoManager extends ConnectedVimServiceBase {
  private ManagedObjectReference cryptoManager = null;
  private KmipServerSpec kmipSpec = null;
  private String kmsName;
  private String kmsIp;
  private int kmsPort = 5696; // default
  private String kmsClusterId;

   public void initCryptoManager() throws RuntimeFaultFaultMsg {
      if (serviceContent != null) {
         cryptoManager = serviceContent.getCryptoManager();
         if (cryptoManager == null) {
            throw new RuntimeFaultFaultMsg("CryptoManager could not be obtained", null);
         }
      } else {
         throw new RuntimeFaultFaultMsg("ServiceContent could not be obtained", null);
      }
   }
   public void registerKmipServer() throws RuntimeFaultFaultMsg {
      KmipServerInfo serverInfo = new KmipServerInfo(); // Create KMS info
      serverInfo.setName(kmsName); // Set the name of your KMS here
      serverInfo.setAddress(kmsIp); // Set the IP addr of your KMS
```

```
      serverInfo.setPort(kmsPort); // Set KMS port, if different from default
      KeyProviderId providerId = new KeyProviderId(); // Set the name of KMS cluster here
      providerId.setId("KMScluster");
      kmipSpec = new KmipServerSpec();
      kmipSpec.setInfo(serverInfo); // KMS spec with server and cluster ID
      kmipSpec.setClusterId(providerId);
      vimPort.registerKmipServer(cryptoManager, kmipSpec); // Register server
   }
   public void trustKmip() throws RuntimeFaultFaultMsg { // Get KMS certificate
      CryptoManagerKmipServerCertInfo certInfo = vimPort.retrieveKmipServerCert(
            cryptoManager, kmipSpec.getClusterId(), kmipSpec.getInfo());
      // Upload retrieved certificate to vCenter Server and trust it
      vimPort.uploadKmipServerCert(
            cryptoManager, kmipSpec.getClusterId(), certInfo.getCertificate());
   }
   public void establishTrust() throws RuntimeFaultFaultMsg {
      // Make KMS trust vCenter Server by uncommenting and calling one of these lines
      // - establishTrustUsingSelfSignedCert() - see Example 12-2 for source code
      // - establishTrustUsingSignedCsr() - see Example 12-3 for source code
      // - establishTrustUsingCertAndKey() - see Example 12-4 for source code
      trustKmip(); // Now make the vCenter Server trust KMS
   }
   public void setDefaultKmipCluster() throws RuntimeFaultFaultMsg {
      vimPort.markDefault(cryptoManager, kmipSpec.getClusterId()); // Mark cluster as
default
   }
   public void generateNewKey() throws RuntimeFaultFaultMsg {
      CryptoKeyResult keyResult = vimPort.generateKey(cryptoManager,
kmipSpec.getClusterId());
      CryptoKeyId keyId = keyResult.getKeyId(); // Generate new key for encryption
   }
   @Action
   public void action() throws RuntimeFaultFaultMsg {
```

```
      initCryptoManager();
      registerKmipServer();
      establishTrust();
      setDefaultKmipCluster();
      generateNewKey();
   }
   @Option(name = "kmsname", description = "Name of the KMS", required = true)
   public void setKMSName(String name) {
      this.kmsName = name;
   }
   @Option(name = "kmsip", description = "IP address of the KMS", required = true)
   public void setKMSIp(String ip) {
      this.kmsIp = ip;
   }
   @Option(name = "kmsport", description = "KMS port", required = false)
   public void setKMSPort(String port) {
      this.kmsPort = Integer.parseInt(port);
   }
   @Option(name = "kmsclusterid", description = "KMS cluster Id", required = true)
   public void setKMSClusterId(String clusterId) {
      this.kmsClusterId = clusterId;
   }
}
```

This example method uploads a self-signed vCenter certificate, or the Root CA certificate, to the KMS.

Trust with self-signed certificate or Root CA certificate

```
public void establishTrustUsingSelfSignedCert() throws RuntimeFaultFaultMsg {
   // Generate self-signed cert, or obtain the Root CA certificate
   String selfSignedCert = vimPort.generateSelfSignedClientCert(
         cryptoManager, kmipSpec.getClusterId());
   // Follow steps for KMS to trust self-signed or Root CA cert, update vCenter to use it
   vimPort.updateSelfSignedClientCert(
         cryptoManager, kmipSpec.getClusterId(), selfSignedCert);
```

```
}
```

This example method generates a CSR and downloads the KMS signed certificate onto vCenter Server.

Trust with CSR then downloading KMS signed certificate

```
public void establishTrustUsingSignedCsr() throws RuntimeFaultFaultMsg {

   // Generate a Certificate Signing Request

   String csr = vimPort.generateClientCsr(cryptoManager, kmipSpec.getClusterId());

   String signedCert = null;

   // Follow steps for your KMS to sign CSR and get the signedCert to update on vCenter

   vimPort.updateKmsSignedCsrClientCert(

         cryptoManager, kmipSpec.getClusterId(), signedCert);

}
```

This example method downloads a certificate and private key generated by the KMS to vCenter Server.

Trust by downloading KMS certificate and private key

```
public void establishTrustUsingCertAndKey() throws RuntimeFaultFaultMsg {

   String certFromKms = null;

   String privateKeyKms = null;

   // Follow steps for KMS to generate certificate and private key (certFromKms,
privateKeyKms)

   vimPort.uploadClientCert(

         cryptoManager, kmipSpec.getClusterId(), certFromKms, privateKeyKms);

}
```

Create an Encryption Storage Policy

Before any virtual machines can be encrypted, vCenter Server must contain an encryption storage policy. The policy only needs to be created once; it can be assigned to any virtual machines that you want to encrypt.

Java program to set storage policy for encryption

```
package com.vmware.spbm.samples;

import java.util.ArrayList;

import java.util.List;

import com.vmware.common.annotations.Action;

import com.vmware.common.annotations.Option;

import com.vmware.common.annotations.Sample;
```

```
import com.vmware.pbm.InvalidArgumentFaultMsg;
import com.vmware.pbm.PbmCapabilityConstraintInstance;
import com.vmware.pbm.PbmCapabilityInstance;
import com.vmware.pbm.PbmCapabilityMetadata;
import com.vmware.pbm.PbmCapabilityMetadataPerCategory;
import com.vmware.pbm.PbmCapabilityProfileCreateSpec;
import com.vmware.pbm.PbmCapabilityPropertyInstance;
import com.vmware.pbm.PbmCapabilityPropertyMetadata;
import com.vmware.pbm.PbmCapabilitySubProfile;
import com.vmware.pbm.PbmCapabilitySubProfileConstraints;
import com.vmware.pbm.PbmCapabilityVendorNamespaceInfo;
import com.vmware.pbm.PbmCapabilityVendorResourceTypeInfo;
import com.vmware.pbm.PbmDuplicateNameFaultMsg;
import com.vmware.pbm.PbmFaultProfileStorageFaultFaultMsg;
import com.vmware.pbm.PbmProfileId;
import com.vmware.pbm.PbmServiceInstanceContent;
import com.vmware.spbm.connection.ConnectedServiceBase;
import com.vmware.spbm.connection.helpers.PbmUtil;
import com.vmware.vim25.ManagedObjectReference;
import com.vmware.vim25.RuntimeFaultFaultMsg;
/**
 * CreateVMEncryptionProfile
 * Create new Storage Profile with one rule-set based on vmwarevmcrypt capabilities.
 *  Parameters:
 * vcurl       [required] : web service url, for example https://10.9.8.7/sdk
 * username    [required] : username for the authentication
 * password    [required] : corresponding password
 * profilename [required] : name of the storage profile
 * Command Line:
 * run.bat com.vmware.spbm.samples.CreateVMEncryptionProfile --vcurl [webserviceurl] ^
 *         --username [username] --password [password] --profilename [Storage Profile
Name]
 */
```

```
@Sample(name = "CreateVMEncryptionProfile",
 description = "Create a new storage profile with "
             + "one rule-set based on vmwarevmcrypt capabilities.")
public class CreateVMEncryptionProfile extends ConnectedServiceBase {
   private PbmServiceInstanceContent spbmsc;
   private String profileName;


   // Build capability instance based on capability name associated with vmwarevmcrypt
provider
   PbmCapabilityInstance buildCapability(String capabilityName,
                                         List<PbmCapabilityMetadataPerCategory> metadata)
   throws InvalidArgumentFaultMsg {
      // Create Property Instance with capability vmwarevmcrypt
      PbmCapabilityMetadata capabilityMeta =
            PbmUtil.getCapabilityMeta(capabilityName,metadata);
      if (capabilityMeta == null)
         throw new InvalidArgumentFaultMsg("Specified Capability does not exist", null);
      // Create and associate Property Instances with a Rule
      PbmCapabilityConstraintInstance rule = new PbmCapabilityConstraintInstance();
      for (PbmCapabilityPropertyMetadata propMeta : capabilityMeta.getPropertyMetadata())
{
         PbmCapabilityPropertyInstance prop = new PbmCapabilityPropertyInstance();
         prop.setId(propMeta.getId());
         prop.setValue(propMeta.getDefaultValue());
         rule.getPropertyInstance().add(prop);
      }
      // Associate Rule with a Capability Instance
      PbmCapabilityInstance capability = new PbmCapabilityInstance();
      capability.setId(capabilityMeta.getId());
      capability.getConstraint().add(rule);
      return capability;
   }
   @Action
```

```
  public void createProfile() throws RuntimeFaultFaultMsg,
        com.vmware.pbm.RuntimeFaultFaultMsg, InvalidArgumentFaultMsg,
        PbmDuplicateNameFaultMsg, PbmFaultProfileStorageFaultFaultMsg {
     // Get PBM Profile Manager & Associated Capability Metadata
     spbmsc = connection.getPbmServiceContent();
     ManagedObjectReference profileMgr = spbmsc.getProfileManager();
     // Step 1: Check if there is a vmwarevmcrypt Provider
     Boolean encryptionCapable = false;
     List<PbmCapabilityVendorResourceTypeInfo> vendorInfo =
        connection.getPbmPort().pbmFetchVendorInfo(profileMgr, null);
     for (PbmCapabilityVendorResourceTypeInfo vendor : vendorInfo)
        for (PbmCapabilityVendorNamespaceInfo vnsi : vendor .getVendorNamespaceInfo())
           if (vnsi.getNamespaceInfo().getNamespace().equals("vmwarevmcrypt")) {
              encryptionCapable = true;
              break;
           }
     if (!encryptionCapable)
        throw new RuntimeFaultFaultMsg(
              "Cannot create storage profile. 'vmwarevmcrypt' Provider not found.",
null);
     // Step 2: Get PBM Supported Capability Metadata
     List<PbmCapabilityMetadataPerCategory> metadata =
       connection .getPbmPort().pbmFetchCapabilityMetadata(profileMgr,

PbmUtil.getStorageResourceType(),
                                                          "com.vmware.iofilters");
     // Step 3: Add Provider Specific Capabilities
     List<PbmCapabilityInstance> capabilities = new ArrayList<PbmCapabilityInstance>();
     capabilities.add(buildCapability("vmwarevmcrypt@ENCRYPTION", metadata));
     // Step 4: Add Capabilities to a RuleSet
     PbmCapabilitySubProfile ruleSet = new PbmCapabilitySubProfile();
     ruleSet.getCapability().addAll(capabilities);
     // Step 5: Add Rule-Set to Capability Constraints
```

```
      PbmCapabilitySubProfileConstraints constraints = new
PbmCapabilitySubProfileConstraints();

      ruleSet.setName("Rule-Set " + (constraints.getSubProfiles().size() + 1));

      constraints.getSubProfiles().add(ruleSet);

      // Step 6: Build Capability-Based Profile

      PbmCapabilityProfileCreateSpec spec = new PbmCapabilityProfileCreateSpec();

      spec.setName(profileName);

      spec.setDescription("Storage Profile in SDK. Rule based on Encryption capability");

      spec.setResourceType(PbmUtil.getStorageResourceType());

      spec.setConstraints(constraints);

      // Step 7: Create Storage Profile

      PbmProfileId profile = connection.getPbmPort().pbmCreate(profileMgr, spec);

      System.out.println("Profile " + profileName + " with ID: " + profile.getUniqueId());

   }

   @Option(name = "profilename", description = "Name of the storage profile", required =
true)

   public void setProfileName(String profileName) {

      this.profileName = profileName;

   }

}
```

Create an Encrypted Virtual Machine

The Web Services SDK provides Java code to create a virtual machine. The Java code can be modified to create an encrypted virtual machine. Steps below show how to implement virtual machine encryption in the `VMCreate.java` sample program. The **`crypto`** property in **`VirtualMachineConfigSpec`** should be set to **`CryptoSpecEncrypt`**, and an encryption key provided. The storage profile must have been previously set to specify encryption. If the **`CryptoSpec`** is unset, but a storage profile with encryption is set, vCenter Server automatically generates new keys and sets **`CryptoSpec`**, if a default KMS has been configured.

To enhance the **`com.vmware.spbm.samples.VMCreate`** program for encryption, follow these steps:

1. Import the following classes:
   ```
   import com.vmware.vim25.CryptoKeyId;

   import com.vmware.vim25.CryptoSpecEncrypt;

   import com.vmware.vim25.KeyProviderId;

   import com.vmware.vim25.VirtualDeviceConfigSpecBackingSpec;
   ```
2. Declare the following variables:

```
private CryptoKeyId cryptoKeyId;

private String keyId;

private String providerId;
```

3. In the **createVirtualDisk()** function, find the following line:
   ```
   diskSpec.setOperation(VirtualDeviceConfigSpecOperation.ADD);
   ```

4. In the **createVirtualDisk()** function, add the following code after the line you found in the previous step:
   ```
   if ((keyId != null) && (providerId != null)) {

      CryptoSpecEncrypt cSpec = new CryptoSpecEncrypt();

      VirtualDeviceConfigSpecBackingSpec backingSpec =

            new VirtualDeviceConfigSpecBackingSpec();

      cSpec.setCryptoKeyId(buildCryptoKeyId());

      backingSpec.setCrypto(cSpec);

      diskSpec.setBacking(backingSpec);

   }
   ```

5. In the **createVmConfigSpec()** function, find the following line:
   ```
   VirtualMachineConfigSpec configSpec = new VirtualMachineConfigSpec();
   ```

6. In the **createVmConfigSpec()** function, add the following code after the line you found in the previous step:
   ```
   if ((keyId != null) && (providerId != null)) {

      CryptoSpecEncrypt cryptoSpecForVMHome = new CryptoSpecEncrypt();

      cryptoSpecForVMHome.setCryptoKeyId(buildCryptoKeyId());

      configSpec.setCrypto(cryptoSpecForVMHome);

   }
   ```

7. Add the following options toward the end of file:
   ```
   @Option(name = "keyid", description = "Key Id", required = false)

   public void setKeyId(String kid) {

      this.keyId = kid;

   }

   @Option(name = "providerid", description = "Cluster/Provider Id", required = false)

   public void setProviderId(String pid) {

      this.providerId = pid;

   }
   ```

The SPBM sample code is in the Storage Policy SDK, not in the vSphere Web Services SDK.

Clone an Encrypted Virtual Machine

Set the **crypto** property in **RelocateSpec** at **cloneSpec.location**. The virtual machine must be powered off, with no existing snapshots. The encrypted virtual machine will be cloned, still encrypted.

Clone an encrypted virtual machine

```
// Step 5: Create Specs

VirtualMachineRelocateSpec relocSpec = new VirtualMachineRelocateSpec();

VirtualMachineCloneSpec cloneSpec = new VirtualMachineCloneSpec();

VirtualMachineConfigSpec configSpec = new VirtualMachineConfigSpec();

// Step 6: Associate Storage Profile

relocSpec.getProfile().add(getVMDefinedProfileSpec(profileName));

cloneSpec.setConfig(configSpec);

cloneSpec.setLocation(relocSpec);

cloneSpec.setPowerOn(false);

cloneSpec.setTemplate(false);

// Step 7: Clone VM

ManagedObjectReference

     cloneTask = connection.getVimPort().cloneVMTask(vmRef, vmFolderRef, cloneName,
cloneSpec);
```

Encrypt an Existing Virtual Machine or Disk

First set the **crypto** property in the **VirtualMachineConfigSpec** to **CryptoSpecEncrypt** and provide a key. The virtual machine must be powered off, with no existing snapshots.

Storage profiles must also be set to specify encryption. If the **CryptoSpec** is unset, but a storage profile with encryption is set, the vCenter Server automatically generates new keys and sets **CryptoSpec**, if the default KMS cluster has been configured.

Encrypt a virtual machine based on storage profile

```
void Encrypt() throws Exception {

   // Create VirtualMachineConfigSpec

   VirtualMachineConfigSpec vmConfigSpec = new VirtualMachineConfigSpec();

   // Create VirtualDeviceConfigSpec

   VirtualDeviceConfigSpec diskSpec = new VirtualDeviceConfigSpec();

   // Get VirtualMachineProfileSpec for new encryption profile and name it
encryptionProfile

   VirtualMachineProfileSpec encryptionProfile = new VirtualMachineDefinedProfileSpec();
```

```
   // Get VirtualDisk for disk to reconfigure as in com.vmware.vm.VMReconfig, name it disk

   diskSpec.setDevice(disk);

   diskSpec.setOperation(VirtualDeviceConfigSpecOperation.EDIT);

   // Add encryption profile to VirtualDeviceConfigSpec

   diskSpec.getProfile().add(encryptionProfile);

   // Create CryptoSpec for encryption

   // Get Key Id from CryptoManager as keyId

   // See CryptoManager for details on generating or retrieving CryptoKeyId

   CryptoSpecEncrypt cryptoSpec = new CryptoSpecEncrypt();

   cryptoSpec.setCryptoKeyId(keyId);

   // Add CryptoSpecEncrypt to diskSpec backing

   VirtualDeviceConfigSpecBackingSpec backingSpec = new
VirtualDeviceConfigSpecBackingSpec();

   backingSpec.setCrypto(cryptoSpec);

   diskSpec.setBacking(backingSpec);

   // When encrypting a VirtualDisk, the VM home needs to be encrypted also.

   // You can choose the same key to encrypt VM home and virtual disk, or use different
keys.

   // Set cryptoSpec and profile for encrypting virtual machine home.

   vmConfigSpec.setCrypto(cryptoSpec);

   vmConfigSpec.getVmProfile().add(encryptionProfile);

   // Set the device changes

   vmConfigSpec.getDeviceChange().add(diskSpec);

   // Issue reconfigure - See reConfig() in com.vmware.vm.VMReconfig to reconfigure VM.

}
```

Decrypt an Encrypted Virtual Machine or Disk

This is similar to encrypting an existing virtual machine, but with a different **`CryptoSpec`**. First set the **`crypto`** property in the **`VirtualMachineConfigSpec`** to **`CryptoSpecDecrypt`**. The virtual machine must be powered off, and the storage profiles must be set not to specify encryption.

If the **`CryptoSpec`** is unset, but a storage profile without encryption is set, the vCenter Server sets **`CryptoSpecDecrypt`**. The **`ConfigSpec.crypto`** parameter must be set explicitly. Only encryption is deduced from the storage profile.

Decrypt a virtual machine based on storage profile

```
void Decrypt() throws Exception {

   // Create VirtualMachineConfigSpec

   VirtualMachineConfigSpec vmConfigSpec = new VirtualMachineConfigSpec();

   // Create VirtualDeviceConfigSpec

   VirtualDeviceConfigSpec diskSpec = new VirtualDeviceConfigSpec();

   // Create Empty VirtualMachineProfileSpec for decryption

   VirtualMachineProfileSpec emptyProfile = new VirtualMachineEmptyProfileSpec();

   // Get VirtualDisk for disk to reconfigure in com.vmware.vm.VMReconfig, name it disk.

   diskSpec.setDevice(disk);

   diskSpec.setOperation(VirtualDeviceConfigSpecOperation.EDIT);

   // Remove any encryption profile set for the disk

   diskSpec.getProfile().add(emptyProfile);

   // Create CryptoSpec for decryption

   CryptoSpecDecrypt cryptoSpec = new CryptoSpecDecrypt();

   // Add CryptoSpecEncrypt to diskSpec backing

   VirtualDeviceConfigSpecBackingSpec backingSpec = new
VirtualDeviceConfigSpecBackingSpec();

   backingSpec.setCrypto(cryptoSpec);

   diskSpec.setBacking(backingSpec);

   // Decrypting virtual machine home is not necessary when decrypting a virtual disk.

   // If no encrypted disk is present on virtual machine after that disk is decrypted,

   // you can choose to decrypt virtual machine home.

   // Set cryptoSpec and profile for decrypting virtual machine home

   vmConfigSpec.setCrypto(cryptoSpec);

   vmConfigSpec.getVmProfile().add(emptyProfile);

   // Set the device changes

   vmConfigSpec.getDeviceChange().add(diskSpec);

   // Issue reconfigure - See reConfig() in com.vmware.vm.VMReconfig to reconfigure VM.

}
```

Encrypt with Different Keys for VM and Disk

This method shows how to use two different keys to encrypt the virtual machine (VM home) and its disk.

Different encryption keys for VM home and virtual disk

```
void EncryptUsingDifferentKeys() throws Exception {
   // Create VirtualMachineConfigSpec and VirtualDeviceConfigSpec
   VirtualMachineConfigSpec vmConfigSpec = new VirtualMachineConfigSpec();
   VirtualDeviceConfigSpec diskSpec = new VirtualDeviceConfigSpec();
   // Get VirtualMachineProfileSpec for encryption profile you created and name it
   VirtualMachineProfileSpec encryptionProfile = new VirtualMachineDefinedProfileSpec();
   // Get VirtualDisk for disk to be reconfigured as shown in VMReconfig and name it
   diskSpec.setDevice(disk);
   diskSpec.setOperation(VirtualDeviceConfigSpecOperation.EDIT);
   // Add encryption profile to VirtualDeviceConfigSpec
   diskSpec.getProfile().add(encryptionProfile);
   // Create CryptoSpec for disk encryption and get KeyId from from CryptoManager
   // See CryptoManager for details on generating or retrieving CryptoKeyId
   CryptoSpecEncrypt cryptoSpecForDisk = new CryptoSpecEncrypt();
   cryptoSpecForDisk.setCryptoKeyId(keyIdForDiskEncryption);
   // Add CryptoSpecEncrypt to diskSpec backing
   VirtualDeviceConfigSpecBackingSpec backingSpec = new
VirtualDeviceConfigSpecBackingSpec();
   backingSpec.setCrypto(cryptoSpecForDisk);
   diskSpec.setBacking(backingSpec);
   // When encrypting a VirtualDisk, the virtual machine home must also be encrypted.
   // You can choose the same key to encrypt VM home and disk, or use different keys.
   // Create CryptoSpec for VM Home encryption and get KeyId from CryptoManager.
   CryptoSpecEncrypt cryptoSpecForVMHome = new CryptoSpecEncrypt();
   cryptoSpecForVMHome.setCryptoKeyId(keyIdForVMHomeEncryption);
   // Set cryptoSpec and profile for encrypting virtual machine home
   vmConfigSpec.setCrypto(cryptoSpecForVMHome);
   vmConfigSpec.getVmProfile().add(encryptionProfile);
   // Set the device changes
   vmConfigSpec.getDeviceChange().add(diskSpec);
}
```

Recrypt (Rekey) Encrypted Virtual Machines

You can recrypt virtual machines using either the vSphere API or the vSphere Client (as of 7.0.x). In the *vSphere Security* manual, recrypt is usually called rekey, or sometimes re-encrypt.

There are two kinds of recryption operations. Deep recrypt replaces all keys, rewriting encrypted data in a powered-off virtual machine and its disks. Shallow recrypt replaces only top-level keys and is comparatively fast.

For details on generating or retrieving the CryptoKeyId, see CryptoManager code in CryptoManager Java program to add KMS and set default cluster.

Recrypt Only Key Encryption Keys

For shallow recrypt, which affects only the key encryption keys (KEKs), set the **`crypto`** property in the **`VirtualMachineConfigSpec`** to **`CryptoSpecShallowRecrypt`** and call the **`Reconfigure`** method.

Shallow Recrypt

```
void ShallowRecrypt() throws Exception {
    // Shallow recrypt follows the same flow as encrypt. The two differences are:
    // - Instead of using a new encryption profile, just get the previously
    //   applied profile from the virtual machine to be reconfigured and use it.
    // - The type of CryptoSpec object created is CryptoSpecShallowRecrypt()
    // Create CryptoSpec for shallow recrypt
    // Get Key Id from CryptoManager as newKeyId
    CryptoSpecShallowRecrypt cryptoSpec = new CryptoSpecShallowRecrypt();
    cryptoSpec.setNewKeyId(newKeyId);
    // Follow steps from Encrypt()
}
```

Shallow recrypt can be done with the virtual machine powered on. Deep recrypt requires power off.

Recrypt Both Key and Disk Encryption Keys

For deep recrypt, which affects both KEKs and disk encryption keys (DEKs), set the **`crypto`** property in the **`VirtualMachineConfigSpec`** to **`CryptoSpecDeepRecrypt`** and call the **`Reconfigure`** method.

Deep Recrypt

```
void DeepRecrypt() throws Exception {
    // Deep recrypt follows the same flow as encrypt. The two differences are:
    // - Instead of using a new encryption profile, just get the previously
    //   applied profile from the virtual machine to be reconfigured and use it.
    // - The type of CryptoSpec object created is CryptoSpecDeepRecrypt()
```

```
    // Create CryptoSpec for deep recrypt
    // Get Key Id from CryptoManager as newKeyId
    CryptoSpecDeepRecrypt cryptoSpec = new CryptoSpecDeepRecrypt();
    cryptoSpec.setNewKeyId(newKeyId);
    // Follow steps from Encrypt()
}
```

Query Crypto Key In-Use Status

Sometimes it is useful to determine key status, for example whether a key is available on vCenter Server, which virtual machines and ESXi hosts are using it, and if third party applications reference it.

As of vSphere 6.7 Update 2, the `queryCryptoKeyStatus` method is available to check use of a KMS key, such as if vCenter Server can access the key, if the key is being used by some virtual machines, or as a host key. The first parameter is a key ID or an array of key IDs to query. The second parameter is a bitmap defining items to check:

- 0x01 - ask if key data is available to vCenter Server
- 0x02 - query which virtual machines use this key
- 0x04 - check the ESXi hosts using this as a host key
- 0x08 - check third party programs using this key.

Key status results are returned in a `CryptoManagerKmip::CryptoKeyStatus` data object, called `keyStatusArray` in the code below. If your program queried multiple keys, an array of results is returned.

The `queryCryptoKeyStatus` method and its returned data object are new in vSphere 6.7 U2, so it does not work in earlier releases.

The following Java code checks all items above by passing 15, a bitwise OR of the settings above.

```
CryptoManagerKmipCryptoKeyStatus[] keyStatusArray =
   vimPort.queryCryptoKeyStatus(cryptoManager,
   [vmKeyId, diskKeyId, srcHostkey, dstHostkey], 15);
for (CryptoManagerKmipCryptoKeyStatus keyStatus : keyStatusArray[]) {
   System.out.println("keyId: " + keyStatus.getKeyId);
   System.out.println("keyAccessible: " + keyStatus.getKeyAvailable)
   System.out.println("reason: " + keyStatus.getReason)
   System.out.println("encryptedVMs: " + keyStatus.getEncryptedVMs)
   System.out.println("affectedHosts: " + keyStatus.getAffectedHosts)
   System.out.println("referencedByTags: " + keyStatus.getReferencedByTags)
}
```

The `keyAccessible` means the key is available on vCenter Server. The `reason` is either valid, or indicates why the key is not available. The `encryptedVMs` is an array of virtual machine MoRefs, and `affectedHosts` is an array of MoRefs to ESXi hosts. The `referencedByTags` field gives names of third party applications using the key.

Encrypted vSphere vMotion

When virtual machines and data move across the network, all sensitive data remain encrypted.

Enable Encrypted vMotion

The enumeration VirtualMachineConfigSpecEncryptedVMotionModes controls whether encrypted vMotion is disabled, opportunistic (fall back to default unencrypted vMotion if necessary), or required. If encrypted vMotion is required, or opportunistic and available, the vCenter Server can migrate either encrypted or unencrypted virtual machines within a disk cluster, or from one disk cluster to another using Storage vMotion.

Virtual Disk Manager

The following methods in managed object **`VirtualDiskManager`** are affected by encryption:

`copyVirtualDisk` – If the source disk is encrypted, its copied disk is encrypted with the same key, regardless of the **`crypto`** spec. When calling **`copyVirtualDisk_Task`** on vCenter Server, do not specify the **`destSpec`** parameter, which throws a Not Implemented fault; **`destSpec`** is supported only on ESXi hosts.

`createVirtualDisk` – Given a valid **`crypto`** spec, this task creates an encrypted virtual disk.

`moveVirtualDisk` – Encrypted virtual disk cannot be moved to an unencrypted virtual machine.

`createChildDisk` – The child disk of an encrypted disk remains encrypted with the same key.

`reparentDisks` – Encrypted child disk cannot be reparented to a new disk that is not in the same disk chain.

Best Practices for Virtual Machine Encryption

This section gives tips for optimum use of keys and virtual machine encryption.

Key Lifecycle and Removal

The **`removeKey`** and **`removeKeys`** methods delete key(s) from vCenter Server, but they do not delete keys from the KMS. Key lifecycle is managed entirely from the KMS, where stale keys persist. You can invoke the **`listKeys`** method to show keys in use on the vCenter Server, but there is currently no method to query whether a specific key is in use.

Be Careful with Force Remove

The **`force`** parameter of **`removeKey`** and **`removeKeys`** should be used judiciously. With the **`force`** option, the **`removeKey`** and **`removeKeys`** methods delete key(s) from both the vCenter Server and ESXi hosts, even if the key is currently in use. The result can leave virtual machines in a locked state until the key is replaced. The use case for the force option is for customers to prevent a key from being used anywhere, after it was compromised or expired.

Remove Keys on ESXi host

If you call **`removeKey`** on an ESXi host, even without the **`force`** option, the key gets deleted from the host's key cache, and the encrypted virtual machine becomes unusable. ESXi hosts do not track which keys are in use. Rebooting the ESXi host causes vCenter Server to push all keys to the host again, but the virtual machine may not be fully recoverable from its failed state.

Carefully Manage Differential Keys

When you encrypt both a virtual machine and its virtual disks from the vSphere Client, the same key is used for both. When you encrypt a virtual machine and its disks using the API, you can set different encryption keys for the virtual machine and each virtual disk. If one of the disk keys is missing, the power on operation may fail. If the missing key is for

a non-boot virtual disk, you can remove that disk from the virtual machine and retry the power on operation. Take care when managing the lifecycle of different keys for a virtual machine and its disks.

Rename During Registration

The **registerVM_Task** method can rename a virtual machine at registration time. In vSphere 6.5 this required a two-step workaround for encrypted virtual machines, but as of vSphere 6.7, this operation is supported with encryption. Also as of vSphere 6.7, the `reloadVirtualMachineFromPath_Task` is supported for encrypted virtual machines. The reload API is equivalent to unregistering and registering a virtual machine on a different configuration path.

Encryption on vSAN Datastores

For security on vSAN datastores, you should use vSAN encryption instead of virtual machine encryption. This is because vSAN encryption is designed to be compatible with deduplication and compression, whereas virtual machine encryption causes vSAN deduplication and compression to become less effective.

Unlocking Encrypted Virtual Machines

There are many reasons why an encrypted virtual machine could be, in effect, locked. For solutions, see section “Resolve Missing Key Issues” in the *vSphere Security* manual.

In vSphere 6.7 and later you can unlock locked virtual machines with the `cryptoUnlock_Task` method. A vCenter Server alarm notifies you when an encrypted virtual machine is in a locked state. To unlock it you must have **Cryptographic operations.RegisterVM** privilege. Before unlocking, it is a good idea to troubleshoot and attempt to fix the cause of the locked virtual machine. The `cryptoUnlock_Task` method takes one parameter, the `MoRef` of a virtual machine.

Configuring Advanced Options

The advanced options settings of an ESXi host can affect virtual machine security.

Advanced options include welcome messages, **sshd** prompts, SOAP session timeout, eager zeroing of VM memory, login attempts before lockout, seconds delay after lockout, password history, password duration, password quality, DCUI timeout, shell timeout, and host client session timeout. Eager zeroing of memory can be helpful for security.

Setting advanced options on individual ESXi hosts using the UI can be impractical at scale. The vSphere API offers a programmatic interface to manipulate advanced options. The UI is built on top of the API calls.

Advanced options are controlled by the `OptionManager` managed object, a property of `HostConfigManager`. You can show advanced options with `QueryOptions`, and set them with `UpdateOptions`. For details, see the *vSphere API Reference*.

Querying Advanced Options

The full list of advanced options is available in the `OptionManager.supportedOption[]` field, and the list of non-default settings is available in the `OptionManager.setting[]` field. The contents of these arrays are fixed for a particular ESXi build and do not change at runtime.

You can use the `QueryOptions` method to get the setting for any supported option.

Setting Advanced Options

To set an advanced option, call the `UpdateOptions` method with the desired key and value. The following pseudo-code sets the advanced option for memory zeroing to "1" for true:

```
MoRef optionMgr = hostSystem.configManager.advancedOption;

opts = new OptionValue[]
```

```
opts[0].key = "Mem.MemEagerZero"

opts[0].value = "1"

optionMgr.UpdateOptions(changedValue=opts)
```

Physical Platform Resources

ESXi hosts allow direct access to physical devices in limited scenarios.

Many vSphere devices are implemented as virtual devices. Access attempts from the guest are trapped and implemented in software. For example, the keyboard is implemented by trapping I/O port access in the guest and transmitting keycode data obtained using a remote console connection; it is not possible to connect the physical keyboard to any VM.

Several categories of devices can be configured for direct access, including USB, storage, optical disc drives, virtual GPU, and PCI passthrough (DirectPath I/O) devices. Use of physical "passthrough" resources is uncommon, because it prevents vMotion. Some physical resources can be configured for access and still allow vMotion.

Physical device access can be revoked by removing an associated virtual device from its VM configuration.

USB

In its hardware devices field, data object `VirtualMachineConfigInfo` may contain the type `VirtualUSB`. Virtual devices can have a backing field, which specifies access to a particular physical device. For USB connectivity, the VM must have a compatible virtual controller configured for the VM. Programs can set the type of backing object:

- `vim.vm.device.VirtualUSB.USBBackingInfo` – USB device is attached to the ESXi host as a passthrough device, which disallows vMotion.
- `vim.vm.device.VirtualUSB.RemoteHostBackingInfo` – USB device is attached to a particular remote ESXi host, which remains attached to the device during vMotion.
- `vim.vm.device.VirtualUSB.RemoteClientBackingInfo` – USB device gets forwarded over the network along with the VM's remote console.

See data object `VirtualUSB` in the *`vSphere API Reference`* for details. You can use the **`esxcli`** command to find host-attached USB devices for connection to a VM. The following command prints available devices, including VID (Vendor ID) and PID (Product ID), which programs use to configure backing objects types above:

```
esxcli hardware usb passthrough device list
```

You can also attach USB devices to a VM using the Host Client. See procedure "Add a USB Device to a Virtual Machine in the VMware Host Client" on https://docs.vmware.com.

Storage

Passthrough storage devices are known as Raw Device Mappings (RDM) and represent essentially a passthrough of SCSI commands to a particular LUN. RDM can be used with vMotion if LUN IDs are kept consistent across participating ESXi hosts.

Configuring an RDM occurs through the `vim.vm.ConfigInfo.hardware.devices[]` array, with instances of type `vim.vm.device.VirtualDisk` that have backing field of type `vim.vm.device.VirtualDisk.RawDiskVer2BackingInfo`.

Physical CD-ROM or DVD

A physical disc drive on a host can be attached directly to a VM. Such drives are automatically detected by the host; no configuration is required. You can configure VMs to access a physical disc drive. See procedure "Add a CD or DVD Drive to a Virtual Machine in the vSphere Client" on https://docs.vmware.com.

Virtual GPU (vGPU)

When used in conjunction with an nVidia graphics card and the nVidia GRID software package, a vGPU can be passed through to one or more VMs for GPU or general purpose GPU (GPGPU) acceleration.

PCI Passthrough

ESXi allows specific PCI devices to be configured for passthrough use by a VM instead of by the host. This feature is called DirectPath I/O and is used in limited cases for improved low-latency performance. When operating in this configuration, the PCI device must be behind a VT-d bridge that implements an IOMMU. VMkernel arranges for this IOMMU to be programmed with only pages that the guest is allowed to access. This ensures that the guest can program the PCI device to access only pages that the guest should access. If the VM powers off or becomes unavailable, the PCI device receives a PCI “reset” to clear any pending DMA transactions before detaching the IOMMU domain from the device.

The host configuration that indicates which devices are eligible for assignment to VMs is through the `vim.host.PciPassthruSystem` managed object, accessible using the host’s `ConfigManager.pciPassthruSystem`. The `UpdatePassthruConfig` method on this managed object can be used to enable or disable PCI passthrough for a particular PCI bus address (bus:slot:function). Making a device eligible for passthrough usually requires host reboot. Devices not configured for PCI passthrough on the host are instead claimed by ESXi and are thus unavailable to VMs.

The VM configuration uses `vim.vm.ConfigInfo.hardware.devices[]`. When an instance in that array is of type `vim.vm.device.VirtualPCIPassthrough`, the backing field has type `vim.vm.device.VirtualPCIPassthrough.DeviceBackingInfo` and this object describes the exact PCI device that is being passed through.

To enable power-on of a VM, the administrator must specify a PCI device that can be configured for assignment to the VM.

Physical Network

ESXi does not allow guests to have direct access to a physical network except when the administrator configures a virtual switch with a network interface attached to specific physical network interfaces. Any VMs expected to have external network access may be configured to have their virtual network interfaces connected to this virtual switch. See Using a Distributed Virtual Switch for information about configuring virtual switches and virtual network interface backings.

Alternatively, the administrator can configure a virtual switch to connect with a specific VLAN on the physical network and then connect VMs to that virtual switch. This provides logical isolation of the VMs from the physical network. See Adding a Virtual Port Group for information about configuration of the management network.

Virtual Applications: the vApp

A virtual application consists of one or more virtual machines, which are deployed, managed, and maintained as a single unit. This chapter explains how to use the vSphere Web Services SDK for building and managing a virtual application.

About Virtual Applications

A virtual application specifies and encapsulates the components of virtual machines and applications, and the operational policies and service levels associated with those components. A virtual application can be as simple as an individual virtual machine with a specific operating system (virtual appliance), or as complex as a complete corporate Web site. Each virtual machine in a virtual application contains a preinstalled, preconfigured operating system and might contain an application stack optimized to provide a specific set of services.

In the vSphere Web Services SDK, the **`VirtualApp`** managed object represents a virtual application. A **`VirtualApp`** object extends **`ResourcePool`** with the following capabilities:

- Store product information such as product name, vendor, properties, and licenses in **`vAppConfigInfo`**.

- Specify power-on and power-off sequence specification.
- Import and export of **VirtualApp** objects as OVF packages.
- Perform application-level customization using the OVF environment.

Management Overview

You can use the Web Services SDK to create and manage virtual applications by following these steps:

1. Call the **CreateVApp** method to create a virtual application without children. See Creating a VirtualApp.
2. Add child objects. See Managing VirtualApp Children.
3. Export the **VirtualApp** to OVF (**ExportVApp** method) See Exporting a Virtual Application.

   You can then import the OVF to create and customize the virtual application.

Direct and Linked Children

A virtual application consists of one or more child virtual machines or virtual applications. **VirtualApp** children have the following characteristics:

- Each child has exactly one parent **VirtualApp**.
- Each child can participate in power-on and power-off sequences.
- The lifetime of each child is determined by the parent **VirtualApp** object.

VirtualApp children are either direct or linked, based on where a child derives its resources.

- Direct Children. A direct child of a virtual application is a virtual machine or virtual application object that you add explicitly. See Managing VirtualApp Children for a list of methods. Direct children share resources with the parent **VirtualApp** object. Both virtual machines and virtual application can be direct children.
- Linked Children. A linked child of a virtual application is a virtual machine or virtual application that you add by calling the **UpdateLinkedChildren** method. Linked children increase the flexibility of the **VirtualApp** by allowing child entities to use different resources from the parent **VirtualApp** object. Linked children can be part of a different clusters, but a virtual application and its children must be in the same **Datacenter**. Both virtual machines and virtual applications can be linked children.

Linked children gives better flexibility. In particular, you can create virtual applications that span clusters. The vSphere Client does not support adding or removing links, though it does show links.

When you add a linked child to a virtual application, the following rules apply:

- An **InvalidArgument** fault is thrown if the **UpdateLinkedChildren** method is called on a link target that is a direct child of another virtual application.
- When you add a virtual machine or virtual application that is already a linked child of another virtual application, the existing link is removed and replaced with the new link.
- The life-time of a linked child is determined by the **destroyWithParent** property on the **VAppEntityConfigInfo** data object. If set to **true**, the child is destroyed when the parent **VirtualApp** is destroyed. Otherwise, the link is removed when the **VirtualApp** is destroyed.

If you add a virtual application that consists of multiple entities, for example multiple virtual machines, the entities are moved sequentially and committed one at a time, as specified in the list. If a failure is detected, the method terminates with an exception.

OVF Packages

Open Virtualization Format (OVF) is a distribution format for virtual applications. vSphere uses the OVF package as a unit of distribution and storage for virtual applications. Because these entities are uploaded, downloaded, and stored in OVF package format, vSphere supports access to and deployment of a wide variety of virtual applications.

A virtual application typically consists of one or more virtual disk files and a configuration file.

- The virtual disk files contain the operating systems and applications that run on the virtual machines in the virtual application.
- The configuration file contains metadata that describes how the virtual application is configured and deployed.

An OVF package might also include certificate and manifest files.

The OVF package contains metadata that describes the capabilities and infrastructure requirements of the virtual application, and contains references to the virtual disks and other files that store the virtual machine state. Most of this information is stored in an XML document called the OVF envelope. When an OVF package is instantiated into either a **VirtualApp** or a **VirtualMachine** object (which depends on metadata in the envelope), then the configuration stored in the OVF envelope is applied to the **VirtualVApp** and the **VirtualMachine** objects.

Some of the information in the OVF file is used unaltered, with entire **ovf:Section_Type** elements included in the **VirtualApp** object body. Other sections are transformed or extended by instantiation. You do not need detailed knowledge of all OVF package elements, but a basic understanding of key parts of the package and how they relate to virtual applications is useful.

See the OVF specification at the DMTF Web site for additional information.

Creating a VirtualApp

You always create a **VirtualApp** without children. The **CreateVApp** method includes the following parameters:

- **resSpec** – Properties you would specify for a **ResourcePool**.
- **configSpec** – **VAppConfigSpec** data object for specifying virtual-application specific information.
- **vmFolder** – Depends on the **VirtualApp** structure:
  - When creating top-level virtual applications, that is, virtual applications with no ancestor virtual applications, you must specify a folder.
  - If the **VirtualApp** has another virtual application in the ancestry chain, the **folder** parameter must be **NULL** when you create the **VirtualApp**.

Managing VirtualApp Children

You can add virtual machines and virtual applications to your virtual application as direct or linked children.

You use different methods for adding or removing direct or linked children, as follows:

- Direct children. Use one of the following methods:
  - **CreateChildVMTask** adds a new virtual machine.
  - **CreateVApp** adds a new virtual application.
  - **MoveIntoResourcePool** adds or removes an existing virtual machine or virtual application
- Linked children. **Use UpdateLinkedChildren** to add or remove virtual machines or virtual applications.

You can call the **UpdateVappConfig** method to specify how each virtual machine fits into the virtual application.

Property	Enumeration
destroyWithParent	True if the entity should be removed when the **VirtualApp** is removed.
key	Key for the virtual machine or virtual application, a managed object reference to the child.
startAction	One of the strings in the **VAppAutoStartAction** enumeration.
startDelay	Delay, in seconds, before continuing with the next entity.

Table continued on next page

Continued from previous page

Property	Enumeration
startOrder	Specifies the start order for this entity. Entities are started from lower numbers to higher-numbers and reverse on shutdown. Multiple entities with the same start order are started in parallel and the order is unspecified. This value must be 0 or higher.
stopAction	Defines the stop action for the entity. Can be set to **none**, **powerOff**, **guestShutdown**, or **suspend**. If set to **none**, then the entity does not participate in auto-stop.
stopDelay	Delay, in seconds, before continuing with the next entity.
tag	Tag for the entity.
waitingForGuest	Determines if the virtual machine should start after receiving a heartbeat, from the guest.

For more information about direct children and linked children, see Direct and Linked Children.

Exporting a Virtual Application

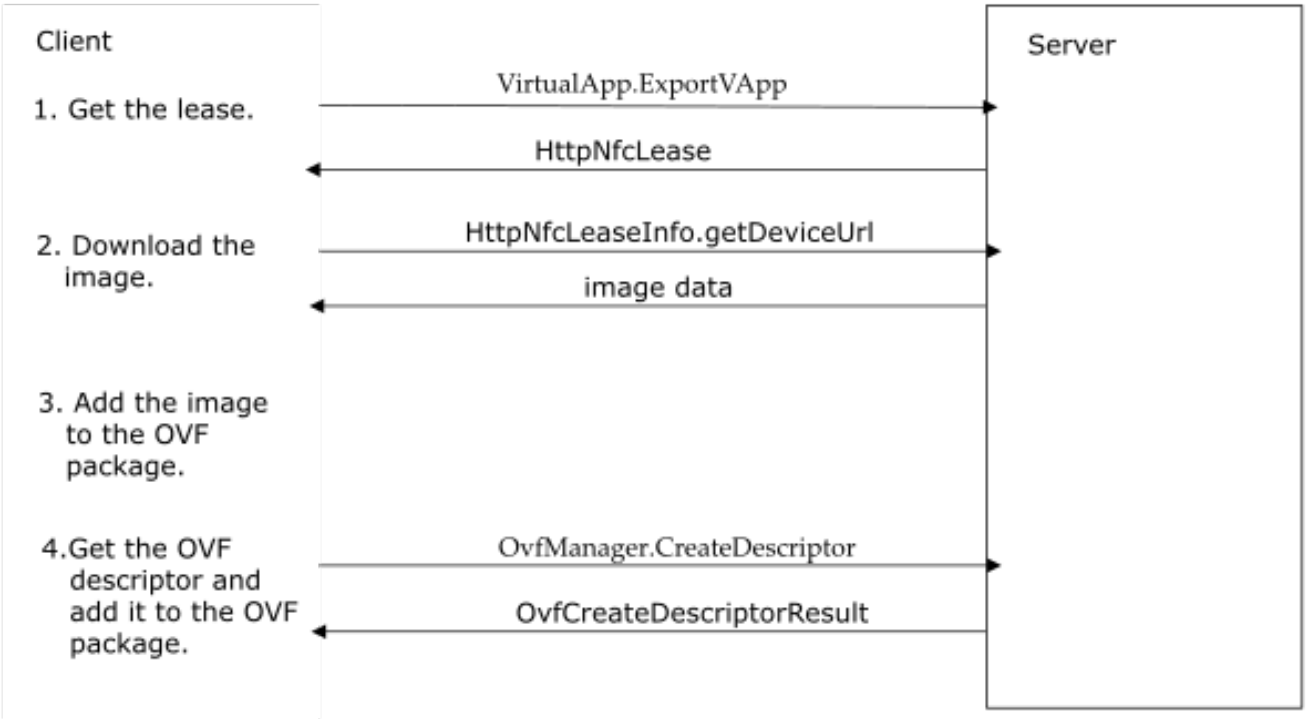
To export a virtual application, you must generate an OVF package. The Web Services API supports the generation of OVF packages. It does not support the generation of OVA files. An OVA file is a tar file that contains an OVF package. The OVF package consists of one of more images and an OVF file descriptor. You can create an OVA file by creating a tar file out of the OVF package for your exported virtual application.

The following steps describe how to use the vSphere `VirtualApp` and `OvfManager` API to generate an OVF package for a virtual application. The steps assume the simplest scenario: downloading one image from one device URL. You use the same steps to download many images from many device URLs. You can also export a VirtualMachine with the same steps, but use `VirtualMachine.ExportVm` rather than `VirtualApp.ExportVApp`.

1. Call the `VirtualApp.ExportVApp` method, which returns `HttpNfcLease`. The `deviceURL` is stored in the `info` property of `HttpNfcLease`.
2. Call the `HttpNfcLeaseInfo.getDeviceUrl` method to access the device URL and download the image data from the device URL.
3. Add the image to the OVF package.
4. Call the `OvfManager.CreateDescriptor` method, which returns `OvfCreateDescriptorResult`. Write the file descriptor to a file with the file extension `.ovf`. Add the `.ovf` file to the OVF package.

Figure: Generating an OVF Package shows the major steps.

Figure 38: Generating an OVF Package



VirtualApp and OvfManager Methods

The following table describes the methods used by the `VirtualApp` and the `OvfManager` API:

Method	Description
CreateDescriptor	Creates an OVF descriptor for the specified `ManagedEntity`, which may be a `VirtualMachine` or a `VirtualApp`. `CreateDescriptor` is a method in the `OvfManager` managed object.
ExportVApp	Obtains an export lease on the virtual application. The export lease contains a list of URLs for the disks of the virtual machines in this virtual application. `ExportVApp` is a method in the `VirtualApp` managed object.
getDeviceUrl	Retrieves the device IDs and URLs from the server. `getDeviceUrl` is an accessor method provided in the generated JAX-WS bindings. It does not appear in the class diagram in VirtualApp Data Structures.

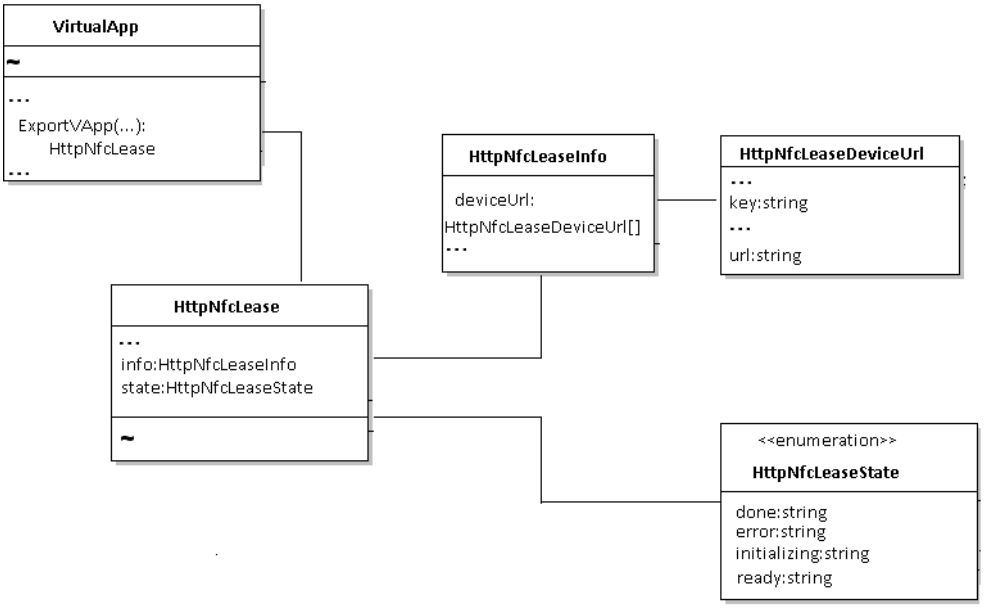
The next two sections deal with the `VirtualApp` and `OvfManager` data structures.

VirtualApp Data Structures

The `VirtualApp` managed object contains the `ExportVApp` method, which returns an `HttpNfcLease`. The `HttpNfcLease` contains the info and state properties, where info is of type `HttpNfcLeaseInfo` and state is of type `HttpNfcLeaseState`. The `HttpNfcLeaseInfo` data object has several properties, one of which is the `deviceUrl` of type `HttpNfcLeaseDeviceUrl[]`. The `HttpNfcLeaseState` has four different states—`done`, `error`, `initializing` and `ready`.

The following class diagram shows the UML representation of the data structures used in the VirtualApp API.

Figure 39: VirtualApp Class Diagram



The VirtualApp API data structures are the following:

- `VirtualApp`—A managed object that is a collection of virtual machines (and potentially other VirtualApp containers) that are operated and monitored as a unit.
- `HttpNfcLease`—A managed object returned when you call `VirtualApp.ExportVApp`. It represents a lease on the virtual application. While you hold the lease, you block the operations that alter the state of the virtual application.
- `HttpNfcLeaseInfo`—A data object that holds information about the lease, such as the virtual application covered by the lease, and the device URLs for up/downloading images.
- `HttpNfcLeaseState`—An enumeration that is a list of possible states of a lease.

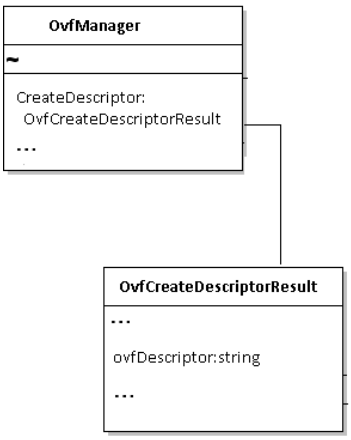
`HttpNfcLeaseDeviceUrl`—A data object that provides a mapping from logical device IDs to upload/download URLs.

OvfManager Data Structures

The OvfManager managed object has a `CreateDescriptor` method that returns an `OvfCreateDescriptorResult`. The `OvfCreateDescriptorResult` has the `ovfDescriptor` string.

Figure: OvfManager Class Diagram shows the UML representation of the data structures used in the OvfManager API.

Figure 40: OvfManager Class Diagram



The `OvfManager` data structures are the following:

- `OvfManager`—A managed object that provides a service interface to parse and generate OVF descriptors.
- `OvfCreateDescriptorResult`—A data object that contains the result of creating the OVF descriptor for the virtual application.

Example of Generating an OVF Package

In summary, the steps in generating an OVF package are the following:

1. Get the managed object reference to the `VirtualApp` object. Call the `ExportVApp` method, which returns an `HttpNfcLease` data object. Wait for the state of the lease to turn to `READY`. Get the list of device URLs from the lease and store them in an array.
2. For each of the URLs in the list of device URLs, download the images from that URL to the client.
3. Save the image to the OVF package (directory/folder). Create an `OvfFile` object using the `deviceID`, absolute path of the downloaded image, and the size of the image on the local disk.
4. Call the `OvfManager.CreateDescriptor` method by passing the managed object reference to the `VirtualApp` and the `OvfFile` object wrapped in an `OvfCreateDescriptorParams` object. This method returns `OvfCreateDescriptorResult`, which contains the file descriptor. Write the file descriptor to a file with the file extension `.ovf`. Add the `.ovf` file to the OVF package.

   The following is an example of how to generate an OVF package. The example assumes a more complex scenario: downloading more than one image from more than one device URL. The example is based on the `OVFManagerExportVAAP.java` sample, which is located in the `SDK/vsphere-ws/java/JAXWS/samples/com/vmware/vapp/` directory.

You can use the `ExportVM` method instead of the `ExportVapp` method when exporting a `VirtualMachine`.

```
package com.vmware.vapp;


import java.io.*;

import java.net.URL;

import java.util.*;
```

```
...
  /** 1. Get the MOR of the VirtualApp.
  ManagedObjectReference vAppMoRef = getVAPPByName(vApp);
...
  /**    Call the ExportVApp method, which returns an HttpNfcLease data object. */
  ManagedObjectReference httpNfcLease = vimPort.exportVApp(vAppMoRef);


...
  /**    Wait for the state of the lease to turn to READY. */
  Object[] result = waitForValues.wait(httpNfcLease,
                                       new String[]{"state"},
                                       new String[]{"state"},
                                       new Object[][]{new Object[]{
                                          HttpNfcLeaseState.READY,
                                          HttpNfcLeaseState.ERROR}});
  if (result[0].equals(HttpNfcLeaseState.READY)) {


...
  /**    Get the list of device URLs from the lease. */
     List<HttpNfcLeaseDeviceUrl> deviceUrlArr = httpNfcLeaseInfo.getDeviceUrl();
     if (deviceUrlArr != null) {
...
  /** 2. For each of the URLs in the list of device URLs,
   *     download the images from that URL to the client. */
        for (int i = 0; i < deviceUrlArr.size(); i++) {
           String deviceId = deviceUrlArr.get(i).getKey();
           String deviceUrlStr = deviceUrlArr.get(i).getUrl();
           String absoluteFile = deviceUrlStr.substring(deviceUrlStr.lastIndexOf("/") +
1);


  /** 3. Save the image to the OVF package (directory/folder). Create an OvfFile object
using
   * the deviceID, absolute path of the downloaded image, and the size of the image on the
```

```
   * local disk.
   */
            long writtenSize = writeVMDKFile(absoluteFile, deviceUrlStr.replace("*",
host));
            OvfFile ovfFile = new OvfFile();
            ovfFile.setPath(absoluteFile);
            ovfFile.setDeviceId(deviceId);
            ovfFile.setSize(writtenSize);
            ovfFiles.add(ovfFile);
         }


  /** 4. Call the OvfManager.CreateDescriptor method by passing the managed object
reference
   * to the VirtualApp and the OvfFile object wrapped in an OvfCreateDescriptorParams
object.
   * This method returns OvfCreateDescriptorResult, which contains the file descriptor.
   * Write the file descriptor to a file with the file extension .ovf. Add the .ovf file
to
   * the OVF package.
   */
         ovfCreateDescriptorParams.getOvfFiles().addAll(ovfFiles);
         OvfCreateDescriptorResult ovfCreateDescriptorResult =
               vimPort.createDescriptor( serviceContent.getOvfManager(),
                                         vAppMoRef,
                                         ovfCreateDescriptorParams);


         String outOVF = localpath + "/" + vApp + ".ovf";
         File outFile = new File(outOVF);
         FileWriter out = new FileWriter(outFile);


         out.write(ovfCreateDescriptorResult.getOvfDescriptor());
         out.close();
```

Importing an OVF Package

To import the virtual application OVF template, you follow a few basic steps. The steps are the same for an OVF package that contains a single virtual machines or an OVF package that contains a more complex virtual application.

1. Parse the OVF descriptor by calling **`OvfManager.parseDescriptor`**.
2. Validate the target ESX/ESXi host by calling **`OvfManager.validateHost`**.
3. Create the **`VirtualAppImportSpec`** by calling **`OvfManager.createImportSpec`**.

   This structure contains all the information needed to create the entities on the vCenter Server, including children. Clients do not have to read or modify **`VirtualAppImportSpec`** to perform basic OVF operations.

4. Create the vCenter Server entities by calling **`ResourcePool.importVApp`**.

   The method uses a parsed OVF descriptor to create **`VirtualApp`** and **`VirtualMachine`** objects in the vSphere environment.

The import process itself consists of two steps:

- The server creates the virtual machines and virtual applications.
  You must wait for the server to create all inventory objects. During object creation, the server monitors the **`state`** property on the **`HttpNfcLease`** object returned from the **`ImportVApp`** call. When the server completes object creation, the server changes the lease to **`ready`** state and you can begin uploading virtual disk contents. If an error occurs while the server is creating inventory objects, the lease changes to the error state, and the import process is terminated.
- The client application uploads virtual disk contents do an HTTP POST request with the content of the disk to the provided URLs. The disk is in the stream-optimized VMDK format. As an alternative, you can use the OVF tool, available for download at the VMware by Broadcom developer site.
  When all inventory objects have been created and the **`HttpNfcLease`** has changed to ready state, you can upload disk contents by using the URLs provided in the **`info`** property of the **`HttpNfcLease`** object. You must call the **`HttpNfcLeaseProgress`** method on the lease periodically to keep the lease alive and report progress to the server. Failure to do so causes the lease to time out, terminating the import process.

When you are done uploading disks, complete the lease by calling the **`HttpNfcLeaseComplete`** method. You can terminate the import process by calling the **`HttpNfcLeaseAbort`** method.

If the import process fails, is terminated, or times out, all created inventory objects are removed, including all virtual disks.

Virtual Application Life Cycle

You can power a virtual application on or off and perform other lifecycle operations.

Powering a Virtual Application On or Off

You can use the **`PowerOnVApp_Task`** method to power on a **`VirtualApp`** object. This method starts the virtual machines or child virtual applications in the order specified in the virtual application configuration.

While a virtual application is starting, all power operations performed on subentities are disabled.

If a virtual machine in a virtual application fails to start, an exception is returned and the power-on sequence terminates. In case of a failure, virtual machines that are already started remain powered on.

You can use the **`PowerOffVApp_Task`** method to power off a virtual application. This method stops the virtual machines or child virtual applications in the order specified in the **`VirtualApp`** object configuration if **`force`** is **`false`**. If **`force`** is

set to **true**, this method stops all virtual machines (in no specific order and possibly in parallel) regardless of the **VirtualApp** object auto-start configuration.

While a virtual application is stopping, all power operations performed on subentities are disabled.

Unregistering a Virtual Application

You can call the **UnregisterVApp_Task** method to remove a **VirtualApp** object from the inventory without removing any of the component virtual machine files on disk. All high-level information stored with the management server (ESXi or vCenter Server system) is removed, including information about **VirtualApp** object configuration, statistics, permissions, and alarms.

Suspending a Virtual Application

You can call the **SuspendVApp_Task** method to suspend all running virtual machines in a virtual application, including virtual machines running in child virtual application. The virtual machines are suspended in the order that is used for a power off operation, which is the reverse of a power on sequence.

While a virtual application is being suspended, all power operations performed on subentities are disabled. If you attempt to perform a power operation, a **TaskInProgress** error results.

Destroying a Virtual Application

When a **VirtualApp** object is destroyed, all of its virtual machines and any child virtual applications are destroyed.

The **VirtualAppVAppState** type defines the set of states a **VirtualApp** object can be in. The transitory state between started and stopped is modeled explicitly, since the starting or stopping of a virtual application might take minutes to complete.

The life-time of a linked child is determined by the **destroyWithParent** property on the **VAppEntityConfigInfo** data object. If set to **true**, the child is destroyed when the parent virtual application is destroyed. Otherwise, only the link is removed when the virtual application is destroyed.

Resource Management

Underlying all virtual components are the actual physical resources of the host system, such as CPU, RAM, storage, network infrastructure, and so on. vSphere supports sharing of resources on an individual host or across hosts using resource pools. vSphere also supports clusters for failover or load balancing.

Resource Management Objects

Central to resource management for all environments is either a **ComputeResource** or a **ClusterComputeResource** managed object.

- The `ComputeResource` managed object represents the set of resources for a set of virtual machines. A `ComputeResource` is always associated with a root `ResourcePool` object, representing the resources of a single host.
- The `ClusterComputeResource` data object aggregates the compute resources of multiple associated `HostSystem` objects into a single compute resource for use by virtual machines. A `ClusterComputeResource` is associated with a root `ResourcePool` representing the combined resources of the cluster. If you plan on using VMware cluster services such as HA (High Availability), DRS (Distributed Resource Scheduling), or on using EVC (Enhanced vMotion Compatibility), use `ClusterComputeResource`.

IMPORTANT
HA, DRS, and EVC may require feature licenses. If any clustering functionality does not work properly, check whether you have licenses for it.

- The `ResourcePool` managed object represents a set of physical resources of a single host, a subset of a host's resources, or resources spanning multiple hosts. Resource pools can be subdivided by creating child resource pools. Only virtual machines associated with a resource pool can be powered on.

Introduction to Resource Management

An ESXi host allocates to each virtual machine a portion of the underlying hardware resources based on several factors:

- Total available resources for the ESXi host, resource pool, or cluster to which the virtual machine belongs.
- Number of virtual machines powered on and resource usage by those virtual machines.
- Overhead required to manage the virtualization.
- Constraints defined by the user.

Resource management allows you to dynamically allocate resources to virtual machines so that you can more efficiently use available capacity. You can change resource allocation in the following ways.

- Specify resource allocation for individual virtual machines. See Configuring Resource Allocation Constraints for Virtual Machines.
- Create a hierarchy of resource pools and add the virtual machine to a resource pool with characteristics appropriate for its use. See Resource Pool Hierarchies.
- Add hosts and virtual machines to a cluster so you can take advantage of VMware DRS for recommendations or automatic resource redistribution. See Creating and Configuring Clusters.

Resource Allocation

When you create a virtual machine, you always specify the resource pool that the virtual machine can draw resources from and optionally a host on which the virtual machine should run. You can access the resource pool as follows:

- Standalone host – When you call **`Folder.AddStandaloneHost_Task`**, the call returns a **`Task`** object that contains the **`ComputeResource`**. The **`ComputeResource.resourcePool`** property is the root resource pool associated with the compute resource (and with the host).
- Cluster – When you call **`Folder.CreateClusterEx`**, the method returns a managed object reference to a **`ClusterComputeResource`** instance. Because **`ClusterComputeResource`** inherits all properties of **`ComputeResource`**, you can access the root resource pool through the **`ClusterComputeResource.resourcePool`** property.

Resource Pool Hierarchies

Resource pool hierarchies allow detailed control over which virtual machines are allowed how many resources, by dividing resources between children at each level of the hierarchy. Child resource pools can be configured with reservations and limits, similar to virtual machines. Child resource pools can also be configured with shares settings, which take effect in resource contention situations.

For more information about how reservations and limits apply to resource pool hierarchies, see Configuring Reservation and Limit for Resource Pools. For more information about how shares apply to resource pool hierarchies, see Configuring Priority Shares for Resource Pools.

Cluster Overview

vSphere supports grouping ESXi hosts that are managed by the same vCenter Server system into clusters. Clusters take advantage of features such as VMware DRS and VMware HA.

- VMware HA (VMware High Availability) fails over virtual machines from one host in a cluster to another host, in the event of host failure.
- VMware DRS (VMware Distributed Resource Scheduler) provides dynamic redistribution of resources. DRS also includes support for Distributed Power Management (DPM), which makes recommendations or decisions to power off hosts and power them on again as needed, to save energy.
  You can set up VMware DRS to automatically migrate virtual machines, or to display recommendations if resources are not used efficiently across the cluster.

See Creating and Configuring Clusters and Managing DRS Clusters.

Creating and Configuring Resource Pools

A root resource pool is associated with each **`ComputeResource`** and with each **`ClusterComputeResource`**.

You can create a hierarchy of resource pools by calling the **`ResourcePool.CreateResourcePool`** method and passing in a **`ResourceConfigSpec`** argument. The **`ResourceConfigSpec.cpuAllocation`** and **`ResourceConfigSpec.memoryAllocation`** properties point to **`ResourceAllocationInfo`** objects that allow you to specify the following information.

- **`reservation`** – Amount of CPU or memory that is guaranteed available to virtual machines within the resource pool. Reserved resources are not wasted if they are not used. If the utilization is less than the reservation, the resources can be borrowed by virtual machines running within other resource pools.
- **`expandableReservation`** – In a resource pool with an expandable reservation, the reservation on a resource pool can expand beyond the specified value, if the parent resource pool has unreserved resources. A non-expandable reservation is called a fixed reservation. See Understanding Expandable Reservations.
- **`limit`** – Upper limit for CPU or memory resources assigned to this resource pool. The resource pool does not allocate more resources to its children, even if resources are available through its parent. This property is typically used to ensure consistent performance by isolating other resource pools from the effects of the running virtual machines within this pool. Set this property to -1 to indicate no fixed upper limit on resource usage.
- **`shares`** – Relative metric for allocating memory or processing capacity among multiple resource pools in resource contention situations. The shares value indicates resource priority relative to the shares values of sibling resource pools or virtual machines. The **`SharesInfo`** data object has two properties, **`level`** and **`shares`**, that allow you to specify resource allocation.
  - **`level`** – Choose **`high`**, **`low`**, or **`normal`** to map to a predetermined set of numeric values for shares. See the *`API Reference Guide`* for the numbers for CPU, memory, and disk shares. Set this property to **`custom`** to specify an explicit number of shares instead.
  - **`shares`** – Allows you to specify a custom value for the number of shares you want to allocate to the resource pool. This property is ignored unless the **`level`** is set to **`custom`**.

To change the configuration, call the **`ResourcePool.UpdateConfig`** or **`ResourcePool.UpdateChildResourceConfiguration`** method and pass in a ResourceConfigSpec that contains values for all fields you want to update.

Configuring Reservation and Limit for Resource Pools

Resource pools are a tool to aggregate physical host resources and channel the aggregated resources to individual virtual machines. You use resource pools to group virtual machines, either to isolate portions af aggregated resources or to prioritize resource allocations between groups of virtual machines.

Resource pools, like individual virtual machines, can be configured with boundaries for resource allocation. Resource pools can also be configured with priority shares, although shares settings act somewhat differently for resource pools than for virtual machines. See Understanding Fixed Shares and Understanding Scalable Shares.

You configure a `reservation` value for a resource pool to help protect its virtual machines from the negative impact of virtual machines in other pools. If you configure a fixed reservation for a resource pool, the host will always make at least

that much of the resource available for the resource pool to distribute among its children. See Understanding Fixed Reservations.

> **NOTE**
> If a resource pool's virtual machines are not using the resource pool's entire reserved amount of a resource, the host may allocate the unused amount to virtual machines in other pools, on a temporary basis. The host will return the borrowed resource to the resource pool that reserved it, whenever needed to start virtual machines within that resource pool.

You configure a `limit` value for a resource pool to help prevent its virtual machines from negatively impacting performance of virtual machines in other resource pools. The host will never allocate more than that quantity of the resource to any or all of the virtual machines within the pool, whether immediate children or chidlren of a nested pool. After all of the resource is allocated, a resource shortage might cause virtual machines within the pool to fail to start or fail to progress, but it has no effect on virtual machines outside the pool.

If you configure expandable reservations for a resource pool and its siblings, and you configure no reserved amounts for any of the siblings, then they share the parent's `reservation` amount on a 'first come first serve' basis. If any siblings are configured for fixed reservations, those siblings are guaranteed their fixed reservations but no more, while the remainder of the parent's reservation is available to share among the pools configured for expandable reservations. See Understanding Exxpandable Reservations.

Understanding Fixed Reservations

A fixed reservation for a resource pool provides a way to guarantee that all virtual machines running simultaneously within the pool have access to a minimum quantity of the resource. The pool's reservation should be large enough to allocate among the virtual machines such that each one has its configured minimum required to start up.

To configure fixed reservations, set the `epandableReservation` property to `false` in `ResourceConfigSpec.ResourceAllocationInfo`. Set the pool's `reservation` amount to an aggregate value that is sufficient to support the maximum number of virtual machines that need to run concurrently within the resource pool. If the pool has nested resource pools that are also configured for fixed reservations, you should determine their `reservation` amounts in the same way, recursively.

When you configure a resource pool hierarchy for fixed reservations, consider these guidelines:

- A resource pool that contains only virtual machines should be configured with a reservation amount at least as great as the sum of the reservations of its children, or as many of them as need to run concurrently. Otherwise, one or more virtual machines will fail to get its minimum resource allocation and the host will not start it.
- A resource pool that contains only nested resource pools configured for fixed reservations must be configured with a reservation amount at least as great as the sum of the reservations of its children. This ensures that its child resource pools can fulfill their obligations to their own children, according to their configured `reservation` amounts. If you cannot configure the parent pool with an adequate amount of the resource, consider configuring the nested resource pools for expandable reservations instead.
- A resource pool that contains both virtual machines and nested resource pools is not a best practice, because it can complicate configuration management. If you configure a resource pool in this way, you should first make sure the virtual machines can receive their own reservation amount, and then configure reservations for the nested pools from the remainder of the parent's reservation after subtracting the virtual machine reservations.

Before you create new child resource pools, check available resources in the parent pool. The `ResourcePool.runtimeInfo` property is a `ResourcePoolRuntimeInfo` data object. The `ResourcePoolRuntimeInfo.cpu` and `ResourcePoolRuntimeInfo.memory` properties are `ResourcePoolResourceUsage` objects with resource usage information, including an `unreservedForPool` property. If the parent pool does not have enough available resources, reconfigure the reservation values of child pools before adding the new pool.

Understanding Expandable Reservations

Expandable reservations enable dynamic allocation of resources to meet the minimum requirements of virtual machines within a number of resource pools. When you set the `expandableReservation` property on sibling resource pools, you enable them to act as a single resource pool for the purpose of providing the minimum of resource that virtual machines need to power on. When a virtual machine starts in any one of the sibling resource pools, it draws its `reservation` amount from the parent's `reservation` amount, rather than from its own resource pool.

If the parent resource pool is also expandable, it can draw resources in turn from its parent, and the sequence can continue in this way until it ends with a parent resource pool that has a fixed reservation.

To configure expandable reservations, set the `expandableReservation` property to `true` in the `ResourceConfigSpec.cpuAllocation` or `ResourceConfigSpec.memoryAllocation` of sibling resource pools to `true`. For maximum flexibility, set the `reservation` amount of each sibling pool `0`. When sibling pools do not reserve any resources, any virtual machine that starts in the sibling pools will take its reservation amount from the parent pool's reservation.

If a sibling pool contains critical virtual machines whose reservations need to be prioritized, set the pool's reservation to the sum of the reservations of the prioritized virtual machines. That reservation amount is saved for the resource pool's virtual machines, and is unavailable for siblings to use.

To illustrate how expandable reservations work, consider the following examples.

Expandable Reservation Example 1

Assume an administrator manages a parent pool P1, and defines two child resource pools, C1 and C2, for two different groups of users. C1 and C2 are self-serve resource pools, allowing users to configure their own virtual machines. The administrator does not know in advance exactly what resources the users will need, so the administrator wants the resource pool configuration to be flexible.

Without expandable reservations, the administrator needs to explicitly allocate fixed amounts of resources to C1 and C2. Such specific allocations can be inflexible, especially in deep resource pool hierarchies, and can complicate setting reservations for C1 and C2. By making the reservations for C1 and C2 expandable, the administrator allows users to more flexibly share and inherit the common reservation for pool P1.

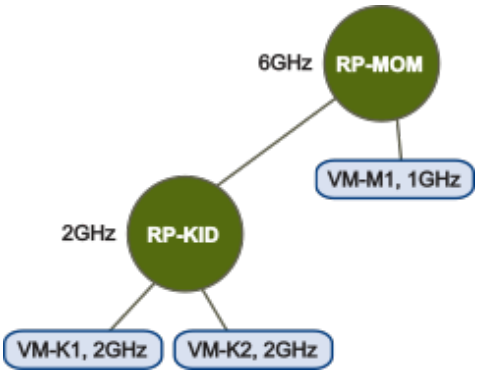
Expandable reservations cause a loss of resource pool isolation in the context of admission control. For example, if C1 and C2 have their reservation amounts set to `0`, then virtual machines in C1 might use all of P1's memory reservation, so that no memory is available to start virtual machines in C2.

Expandable Reservation Example 2

Assume a parent resource pool RP-MOM has a reservation of 6GHz and one running virtual machine VM-M1 that reserves 1GHz. RP-MOM also has a child resource pool RP-KID with a reservation of 2GHz and with **Expandable Reservation** selected. RP-KID contains two virtual machines, VM-K1 and VM-K2, with reservations of 2GHz each.

When a user powers on VM-K1, it can reserve the resources it needs directly from RP-KID (which has 2GHz). When the user tries to power on VM-K2, RP-KID has already allocated its 2GHz reservation to VM-K1, but it has **Expandable Reservation** configured, so it tries to borrow resources from RP-MOM's reservation. RP-MOM has 6GHz minus 1GHz (reserved by VM-M1) minus 2GHz (reserved by RP-KID), which leaves 3GHz of RP-MOM's reservation that is not reserved by other resource pools or virtual machines in RP-MOM. With 3GHz available, VM-K2 is able to power on.

Figure 41: Admission Control with Expandable Resource Pools, Scenario 1

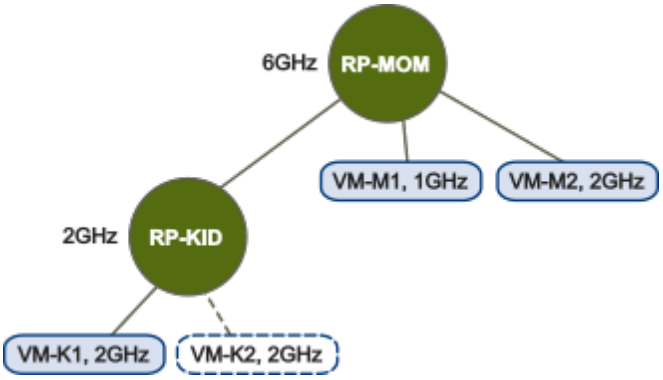


Expandable Reservation Example 3

Assume a parent resource pool RP-MOM has a reservation of 6GHz and two running virtual machines, VM-M1 that reserves 1GHz, and VM-M2 that reserves 2GHz. RP-MOM also has a child resource pool RP-KID with a reservation of 2GHz and with **Expandable Reservation** selected. RP-KID contains two virtual machines, VM-K1 and VM-K2, with reservations of 2GHz each.

When a user powers on VM-K1, it can reserve the resources it needs directly from RP-KID (which has 2GHz). When the user tries to power on VM-K2, RP-KID has already allocated its 2GHz reservation to VM-K1, but it has **Expandable Reservation** configured, so it tries to borrow resources from RP-MOM's reservation. RP-MOM has 6GHz minus 3GHz (reserved by VM-M1 and VM-M2) minus 2GHz (reserved by RP-KID), which leaves only 1GHz of RP-MOM's reservation that is not reserved by other resource pools or virtual machines in RP-MOM. Since VM-K2 requires 2GHz to pass admission control, VM-K2 is not able to power on.

Figure 42: Admission Control with Expandable Resource Pools, Scenario 2



Configuring Priority Shares for Resource Pools

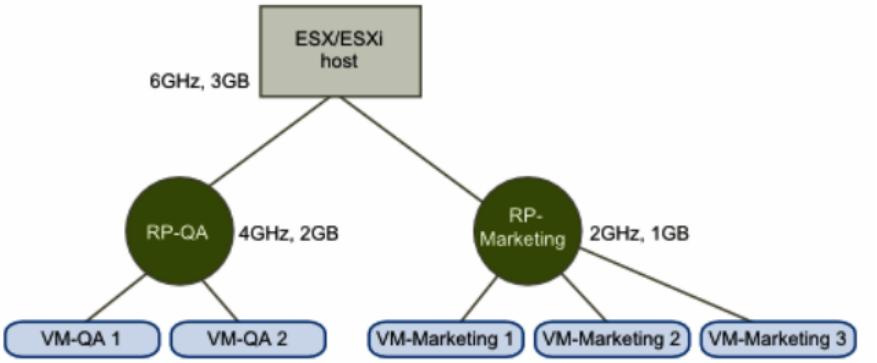
In cases of resource contention, data center hosts need a way to arbitrate between competing virtual machines. You can configure the shares settings of a virtual machine to set its priority relative to its siblings. You can configure the shares settings of a resource pool to affect the priorities of all virtual machines within the resource pool, as a group.

The following illustration shows a standalone host that has several virtual machines. The marketing department uses three of the virtual machines and the QA department uses two virtual machines. Because the QA department needs larger amounts of CPU and memory, the administrator creates one resource pool for each group. The administrator sets CPU Shares to High for the QA department pool and to Normal for the Marketing department pool so that the QA department

users can run automated tests. The second resource pool with fewer CPU and memory resources is sufficient for the lighter load of the marketing staff.

Whenever the QA department is not fully using its allocation, the marketing department can use the available resources. When there is resource contention because running virtual machines demand more resources than are available from the host, the resource shares configuration guides arbitration between the competing virtual machines.

Figure 43: Allocating Resources to Resource Pools



You have two options for priority shares configuration that cause the priorities to be applied in different ways. The fixed shares option limits the adverse performance impact of resource contention to a single resource pool and all its descendants. The scalable shares option distributes the performance impact across all resource pools, in proportion to their priority levels.

The chief benefit of configuring fixed shares for a parent resource pool is the predictability of performance for virtual machines within its child resource pools. You can know in advance that its virtual machines can demand a fixed fraction of a virtual resource available from a host or cluster. The drawback of configuring fixed shares is that adding virtual machines to a resource pool impacts the performance of all virtual machines in the same pool and its descendants because the resource pool is not entitled to allocate additional resources.

The chief benefit of configuring scalable shares for a parent resource pool is that resource allocation during contention is adjusted at run time to achieve a fair allocation to virtual machines beyond the boundary of a single resource pool. In effect, the child pool's resource entitlement expands to accommodate more virtual machines as they are added to the pool. The drawback of configuring scalable shares is that a child resource pool cannot isolate its virtual machines from the demands of virtual machines in other pools that draw from the same scalable parent resource pool.

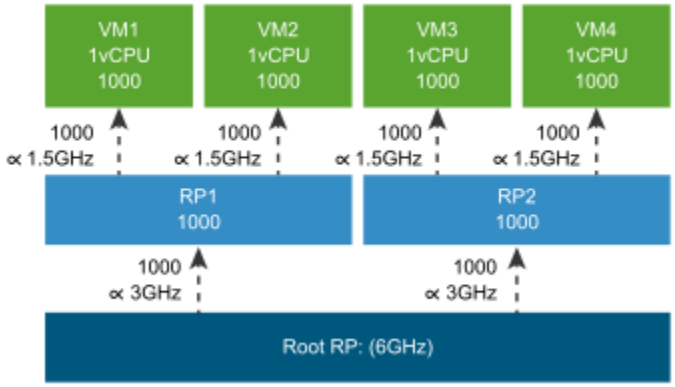
Understanding Fixed Shares

When a parent resource pool is not configured to be scalable, the shares configured for its child resource pools provide a way to prioritize allocation of fixed fractions of the parent resource pool to its children. This serves to isolate users of sibling resource pools so that virtual machines in one child resource pool cannot impact the performance of virtual machines in other child resource pools.

Suppose for example a cluster has 3 physical cores running at 2GHz, for a total of 6GHz. The root resource pool has a total of 6GHz of virtual CPU capacity to divide between its children. If there is no resource contention, all running virtual machines can be allocated their configured amounts of CPU. If there is contention for CPU resources, the 6GHz is divided between child resource pools according to the CPU shares configured for them.

This fictional data center has two child resource pools, each supporting the users in a different division of a business. Division 1 and Division 2 start out about the same size, so the IT department configures RP1 and RP2 with equal values for custom shares, and an equal number of virtual machines.

Figure 44: Example Resource Pool Configuration with Custom Shares



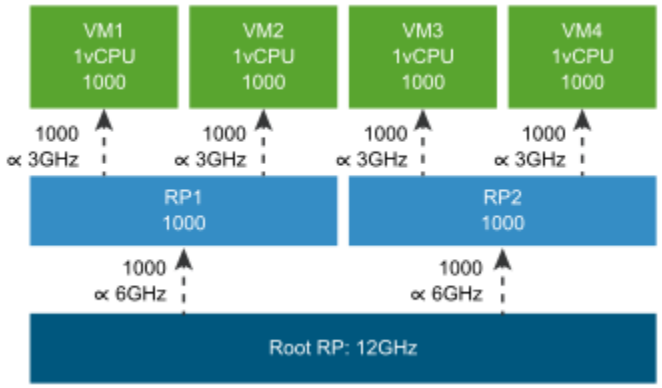
All child resource pools and virtual machines are configured with custom settings of 1000 shares each. Because the two resource pools are configured with the same number of shares, they can allocate the same amount of compute resource to their virtual machines in cases of resource contention. Likewise, all virtual machines are entitled to draw the same fraction of compute resource as their siblings.

> **NOTE**
> For simplicity of illustration, these examples deal only with CPU shares. A real configuration would include both CPU and memory resources, which act independently in regard to calculating the absolute resources available to a running virtual machine.

In effect, a non-scalable resource pool guarantees that each of its children gets a fixed fraction of its resources. However, the amount allocated to each child is fixed only in relation to the quantity of resources controlled by the resource pool itself. If a root resource pool gains additional resources due to a hardware upgrade, all its children gain resource entitlements in proportion to their configured shares.

For example, suppose the cluster has been upgraded from 3 to 6 physical cores running at 2GHz, for a total of 12GHz of pooled physical capacity. The root resource pool now has a total of 12GHz of virtual CPU capacity to divide between its children. If there is contention for CPU resources, the 12GHz is divided between child resource pools according to the CPU shares configured for them.

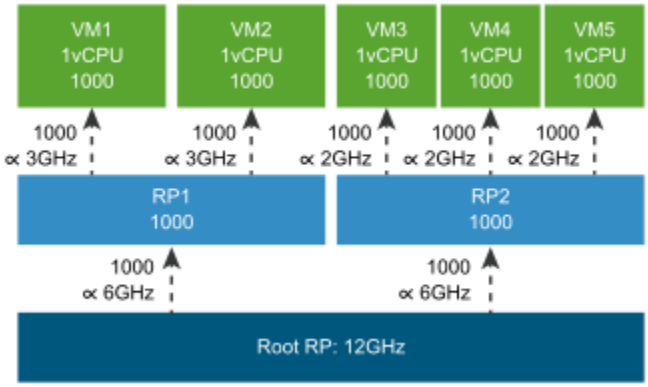
Figure 45: Increased Physical Capcity Provides Increased Virtual Capacity to All Virtual Machines



After the upgrade, each virtual machine is entitled to twice its previous resource allocation, in absolute terms. However, if another child, whether resource pool or virtual machine, is added to a parent pool, previously existing children find their resource allocation reduced, as the pool's resources are shared between more children.

Suppose Division 2 hires some new users, and IT adds an extra virtual machine to RP2 for their use.

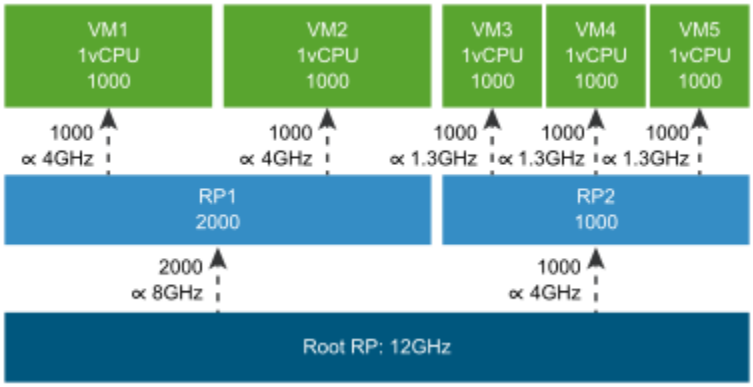
Figure 46: Dilution of Fixed Shares From Adding a Virtual Machine to a Resource Pool



Where previously all virtual machines in the business had the same priority for allocation of scarce compute resource, increasing the virtual machine load in RP2 means that more virtual machines contend for the same quantity of its resource. Consequently, each virtual machine in RP2 might see its performance reduced when all virtual machines are running at full capacity.

To continue from the previous example, some users in Division 1 might be concerned about performance because they plan to hire more employees in the future. Suppose they persuade the IT department that their virtual machines should be in a higher priority resource pool than others, so they will have access to more compute resource if needed. The IT department configures different custom shares values for RP1 and RP2, so that RP1 has twice the priority of RP2.

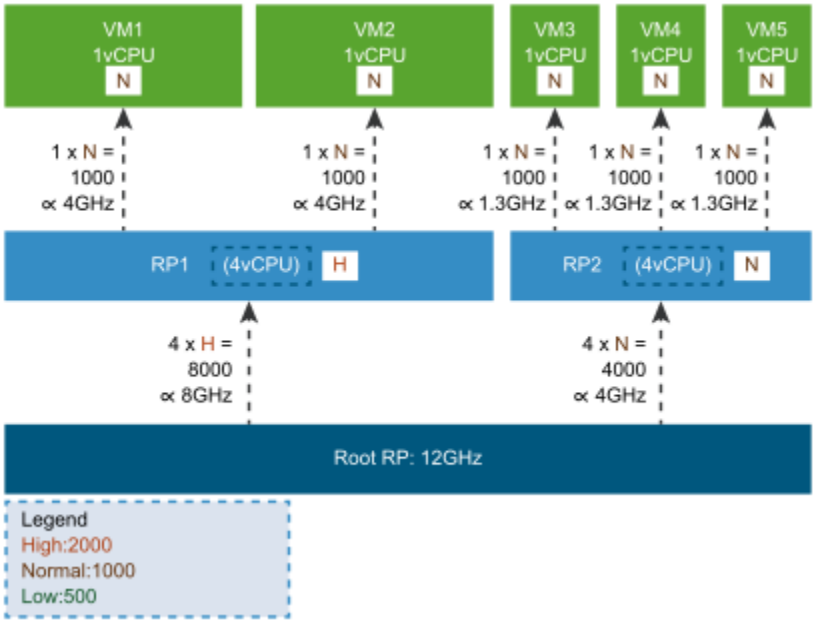
Figure 47: Effects of Reprioritizing a Resource Pool in a Fixed Shares Configurotion



The child resource pool RP1 has 2000 shares configured, and RP2 has 1000 shares configured, so RP1 gets 2/3 of the physical CPU resources to share among its children, and RP2 gets 1/3 of the physical CPU resources to share among its children. You can calculate the amount of compute resource available to RP1 for its children as `12 x 2000 / (2000+1000) = 8`GHz, and the amount of compute resource available to RP2 for its children as `12 x 1000 / (2000+1000) = 4` virtual GHz. The children within each resource pool have identical shares settings, so they share their own pool's resources equally.

Now suppose a new CIO joins the business and decides that from now on priority shares should be specified as levels rather than custom values. The IT department reconfigures all the shares settings to use the enums for shares levels.

Figure 48: Priority Shares Expressed as Levels Instead of Custom Values



When you specify shares levels, you have to make two calculations. First you convert the specified level to a numeric value that corresponds to custom shares settings. This conversion takes into account the number of virtual cores.

NOTE
For memory resources, the conversion takes into account the configured memory size of the virtual machine.

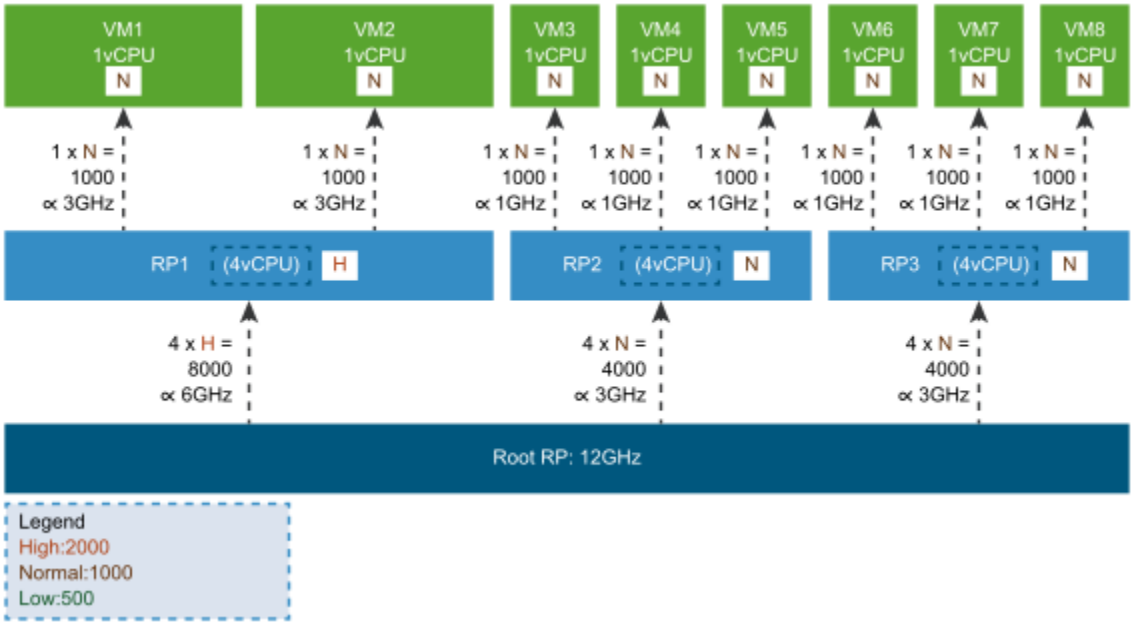
Then you use that numeric value to calculate the proportion of resources to allocate to the child pool, in the same way as with the custom shares settings.

To convert a shares level to a numeric value, multiply the number of vCPUs by a constant that corresponds to the level. For the purpose of this calculation, all resource pools have an implicit size of 4 vCPUs. Use the `VirtualMachine.config.hardware.numCPU` value as the number of vCPUs for virtual machines. The constant for the `high` priority level is 2000, and the constant for the `normal` level is 1000. You can calculate that the numeric shares belonging to RP1 are `4 x 2000 = 8000`. RP2 has a priority level of `normal`, so the numeric shares belonging to RP2 are `4 x 1000 = 4000`. The compute resource that RP1 can allocate is `12 x 8000 / (8000+4000) = 8` GHz, and the compute resource that RP2 can allocate to its children is `12 x 4000 / (8000+4000) = 4` GHz. The new CIO is satisfied that RP1 and RP2 achieve the same result with priority levels as with the previous custom shares values.

NOTE
For memory shares, the constants are different. See the vSphere Web Services API Reference.

But suppose the IT department is asked to support a 3rd division, newly acquired by the business, which is about the same size as the division assigned to RP2. The IT department creates a new child resource pool, RP3, and configures the same number of virtual machines and the same shares levels as RP2. Now the division of resources looks like this:

Figure 49: Resource Dilution from Adding a Resource Pool



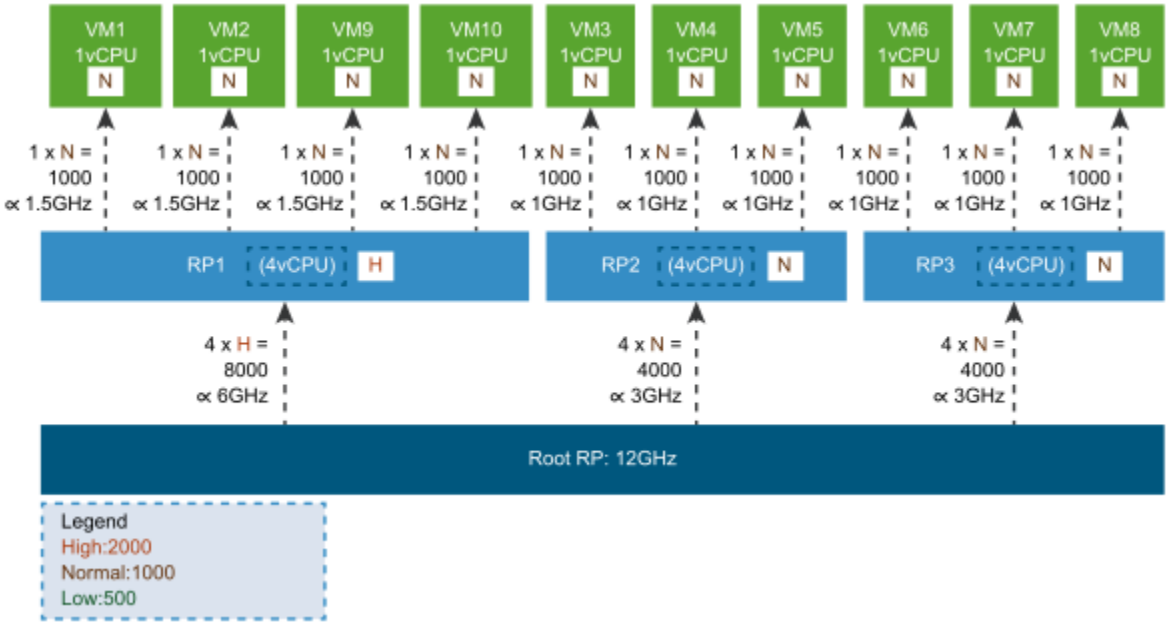
Assume the hardware resources available to the root resource pool have not changed at the time of the acquisition, so the 12GHz of virtual compute resource must be divided into fractions according to the configured shares of the child resource pools. After adding RP3, both RP1 and RP2 get less of the virtual compute resource to divide among their own children.

Virtual machines available to Division 1 now compete for `12 x 8000 / (8000+4000+4000) = 6`GHz in RP1. Virtual machines available to Division 2 compete for `12 x 4000 / (8000+4000+4000) = 3`GHz in RP2. Virtual machines available to Division 3 similarly compete for `12 x 4000 / (8000+4000+4000) = 3`GHz in RP3.

In this example, Division 1 and Division 2 employees might both perceive reduced performance until the IT department's capital budget increases to support the new configuration.

When Division 1 hires a number of new employees, IT adds two more virtual machines to RP1 to accommodate the increased size of Division 1. All four virtual machines in RP1 still benefit from RP1's `high` priority level, but now that RP1's compute resources are spread among a larger number of virtual machines, each of its virtual machines gets a smaller allocation when they are competing for CPU cycles.

Figure 50: Resource Dilution from Adding Virtual Machines to a High Priority Resource Pool

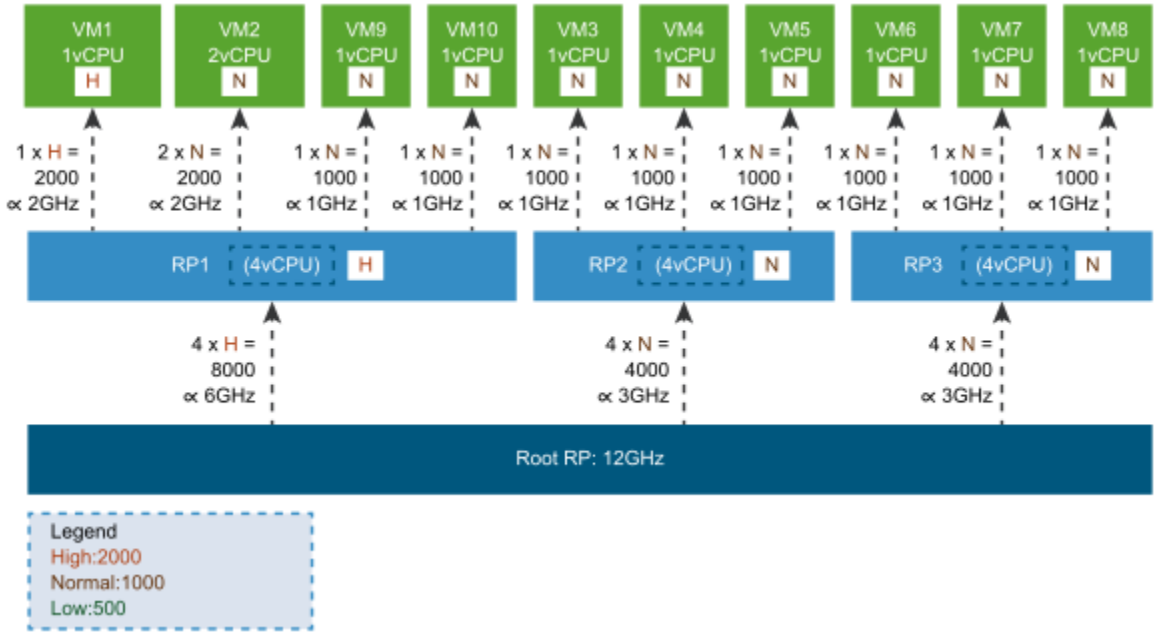


Some virtual machine users who were in Division 1 before the new employees joined are noticing greatly reduced performance. They are only getting 1.5GHz where they used to get 3GHz. When they learn that Division 2 employees are getting 1GHz from their virtual machines in RP2, which has `normal` priority, they might not be happy that they are getting only 1.5GHz from virtual machines in RP1, which has `high` priority.

The disappointed users speak to the IT department, where they learn that this is the way fixed shares for resource pools are intended to work. They isolate Division 1 virtual machine performance from the other divisions, but not from other users of the same resource pool. These Division 1 users learn that there are other ways to protect the performance of high priority virtual machines.

Some Division 1 users try other ways to get more compute resource from RP1's fixed allocation. They request changes to virtual machine configurations, which result in increased priority for some of their virtual machines at the expense of others.

Figure 51: Prioritizing Individual Virtual Machines Within a Resource Pool



After VM1 is reconfigured, you calculate its numeric shares by multiplying the configured `numCPU` (1) by its shares level (`high`), getting `1 x 2000 = 2000`. You calculate its resource allocation by using its numeric shares to prorate the allocation available to its parent resource pool: `6GHz x 2000 / (2x2000+2x1000) = 2GHz`.

You calculate VM2's numeric shares by multiplying the configured `numCPU` (2) by its shares level (`normal`), getting `2 x 1000 = 2000`. You calculate its resource allocation by using its numeric shares to prorate the allocation available to its parent resource pool: `6GHz x 2000 / (2x2000+2x1000) = 2GHz`.

In a similar way, calculate VM9's numeric shares by multiplying its configured `numCPU` (1) by its shares level (`normal`), getting `1 x 1000 = 1000`. You calculate its resource allocation by using its numeric shares to prorate the allocation available to its parent resource pool: `6GHz x 1000 / (2x2000+2x1000) = 1GHz`. VM10's resource configuration and associated calculations are identical to VM9.

When other Division 1 users realize that VM9 and VM10 only get 1GHz of virtual compute resource, they might not be satisfied with this result. These virtual machines are in a `high` priority resource pool, but they get the same resource allocation as Division 2's virtual machines, which are in a `normal` priority resource pool. That doesn't seem fair.

Division 1 users request a meeting with the CIO, who explains that the scalable shares feature is the best solution to their problem.

Understanding Scalable Shares

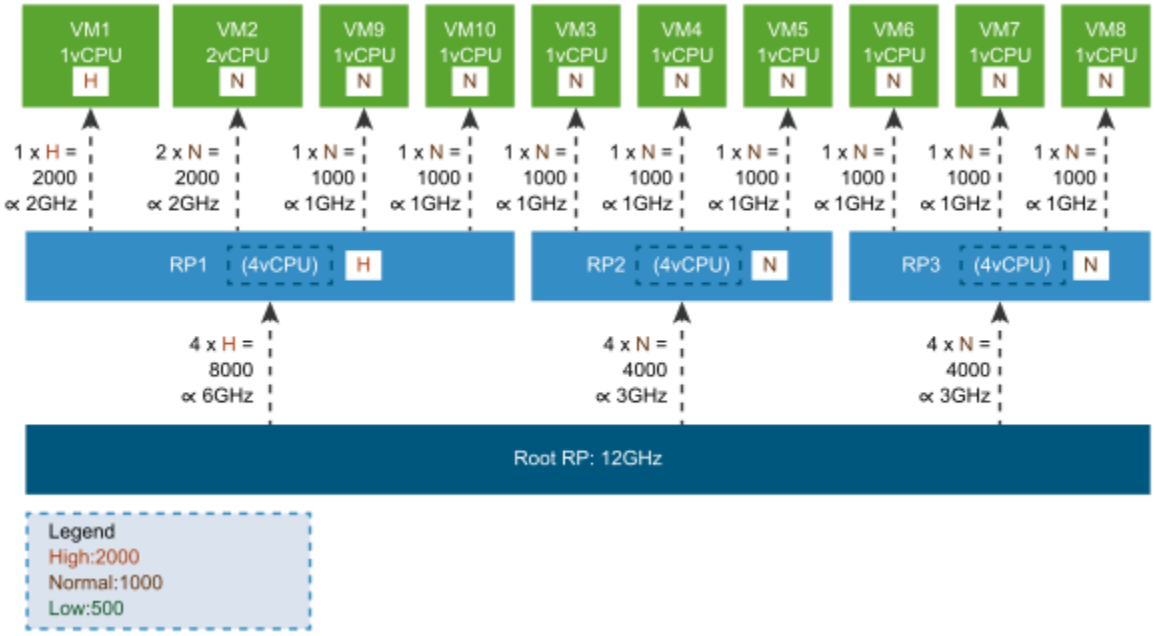
Configuring scalable shares for a parent resource pool means that all its child resource pools are not isolated from each other. This avoids the resource dilution problem within child resource pools, by expanding their resource entitlements when they have more virtual machines running.The effect is to negate resource pool isolation while retaining the proportions of resource pool priority levels. A `high` priority resource pool is entitled to give twice as much of the available resource to its virtual machines, in comparison with a `normal` priority sibling resource pool, but all virtual machines suffer proportionately when many virtual machines run in the normal priority pool.

To illustrate the use of the scalable shares option, consider how it might apply to a fictional situation in which a data center is configured, with non-scalable shares, to support three divisions of a business. Division 1 is supported by resource pool RP1, which is configured as `high` priority, in an attempt to compensate for the dilution of RP1's resources that resulted

from adding new virtual machines to RP1. Some virtual machines in RP1 are individually configured to get higher priority than their siblings.

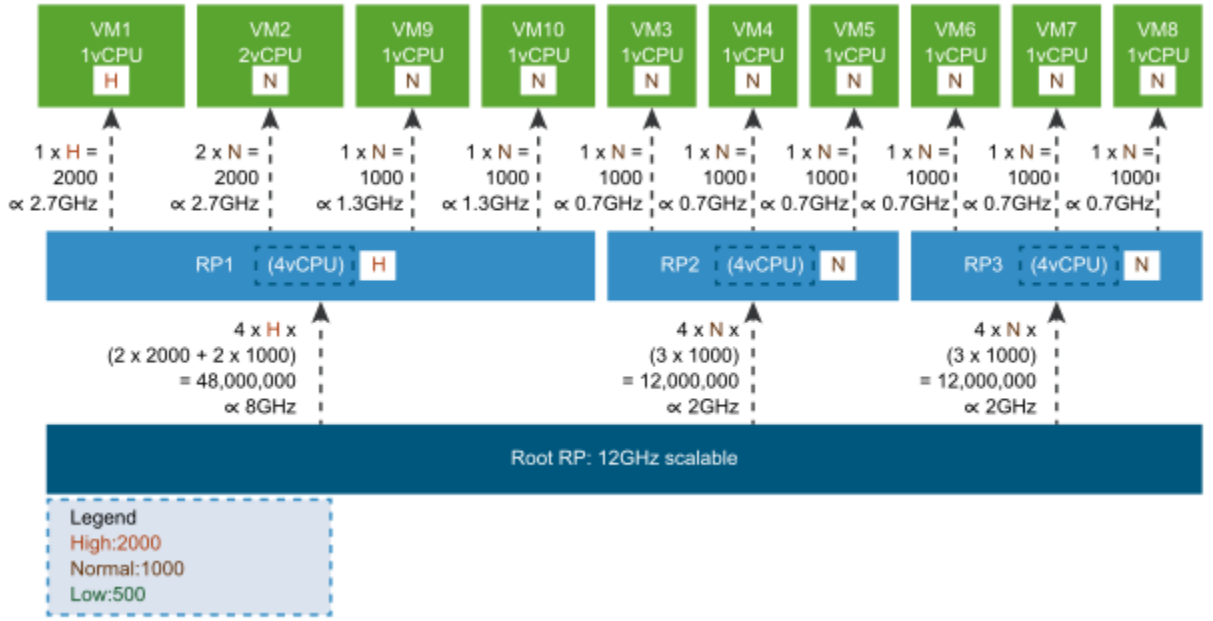
NOTE
For simplicity of illustration, these examples deal only with CPU shares. A real configuration would include both CPU and memory resources, which act independently in regard to calculating the absolute resources available to a running virtual machine.

Figure 52: Example Data Center Configured with Fixed Shares



As a result of resource dilution within RP1 in a fixed shares configuration, some of RP1's virtual machines are entitled to exactly the same resource allocation as RP2's virtual machines, even though RP2 is configured as `normal` priority but RP1 is configured as `high` priority. Some users in Division 1 are disappointed because they expected their virtual machines to receive twice the allotment of RP2's virtual machines. The company's CIO agrees to reconfigure the data center to take advantage of the scalable shares option.

Figure 53: Example Scalable Shares Configuration



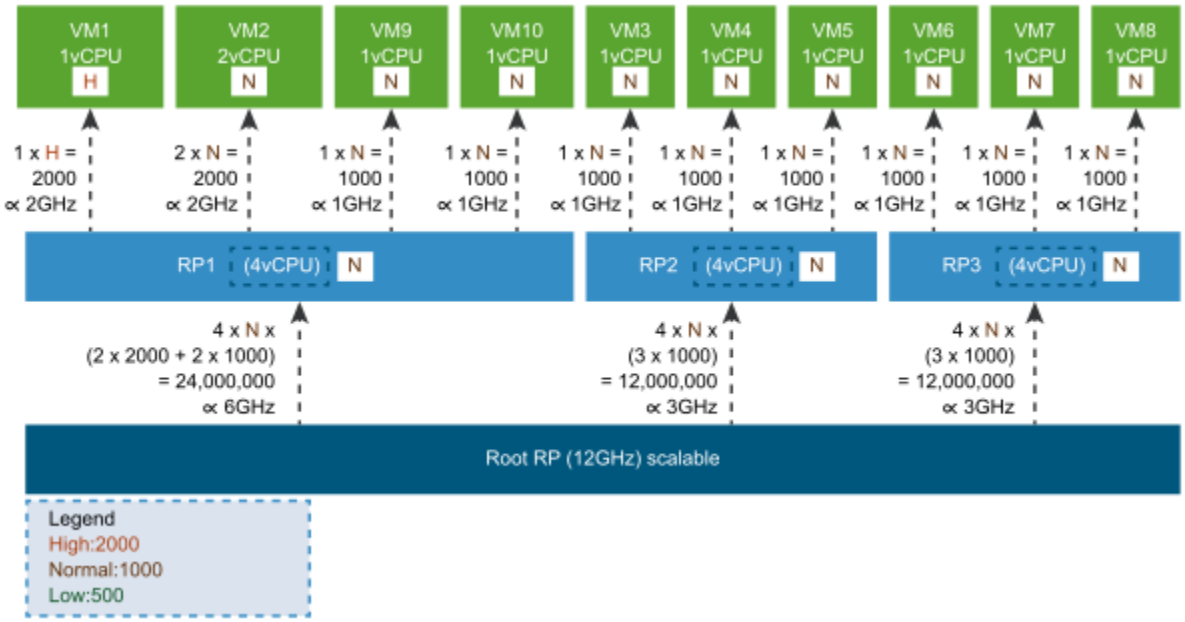
With scalable shares enabled on a parent resource pool, resource entitlements change. Because RP1 has more virtual machines than RP2 or RP3, it is entitled to a larger fraction of the root resource pool's resources during a resource contention situation. This is represented mathematically by multiplying the resource pool's numeric shares by the total of the shares of its children.

Calculating resource pool allocation in a scalable shares configuration is conceptually a six-step recursive process, working from leaf to root of the resource pool tree:

1. Convert the priority levels of its child virtual machines into numeric shares equivalents, in the same way as in a fixed shares configuration: Multiply `VirtualMachine.config.hardware.numCPU` by a constant that depends on the priority level. The constants are `2000`, `1000`, and `500` for `high`, `normal`, and `low` respectively.
2. If the resource pool has child resource pools, convert their priority levels into numeric shares equivalents, recursively.
3. Do the same conversion for the resource pool itself. For the purpose of this conversion, all resource pools are assigned an implicit size of 4 vCPUs.
4. Sum the children's numeric shares and multiply the sum by the parent's numeric shares.
5. Repeat steps 1-4 for all the parent's siblings.
6. Allocate the available resources among all siblings, in proportion to the products from steps 4 and 5.

As a result of the scalable shares configuration, Division 1 users find their virtual machine performance improved, but Divisions 2 and 3 suffer by comparison. To rectify this, the IT department resets the priority levels of RP1, RP2, and RP3 to `normal`. This adjustment leaves virtual machine performance comparable across divisions: `high` priority virtual machines get twice the resource allocation of `normal` priority virtual machines, even across resource pool boundaries.

Figure 54: Scalable Shares Configuration with All Resource Pools Configured at `normal` Level



A best practice when configuring scalable shares is to configure resource pools to the `normal` priority level and apply `high` priority levels or `low` priority levels to specific virtual machines.

Deleting Child Resource Pools

The **`ResourcePool.DestroyChildren`** method recursively deletes all the child resource pools of a resource pool. The operation takes a single parameter, a reference to the parent **`ResourcePool`** managed object. Any virtual machines associated with the child resource pool are reassigned to the parent resource pool.

Moving Resource Pools or Virtual Machines Into a Resource Pool

You can move a resource pool and its children within a resource pool hierarchy.

The **`ResourcePool.MoveIntoResourcePool`** method lets you move virtual machines, virtual applications, or resource pool hierarchies into a new resource pool. You call the method with an array of **`ResourcePool`** or **`VirtualMachine`** managed object references that you want to move. The whole resource pool hierarchy, including child resource pools and virtual machines, is moved when you move a resource pool.

Minimum available resources of the immediate children must always be less than or equal to the resources of the immediate parent. The root resource pool cannot be moved.

Introduction to vSphere Clusters

Clusters are useful with VMware DRS (distributed resource scheduler) and VMware HA (high availability). Clusters can be created quickly with VMware HCI. This guide briefly introduces the vSphere interfaces for cluster configuration. Clusters are useful to reduce power use, downtime, and maintenance.

- VMware DRS. *`vSphere Resource Management`* guide
- VMware HA. *`vSphere Availability`* guide
- VMware HCI, Hyper-converged infrastructure. In vSphere Client, click **New Cluster**.

VMware DRS

A VMware DRS cluster is a collection of ESXi hosts and associated virtual machines with shared resources and a shared management interface. To obtain the benefits of cluster-level resource scheduling you create a DRS cluster.

When you add a host to a DRS cluster, the host's resources become part of the cluster's resources. In addition to aggregation of resources, a DRS cluster supports cluster-wide resource pools and enforces cluster-level resource allocation policies. The following cluster-level resource management capabilities are available.

- Load Balancing. The vCenter Server monitors distribution and use of CPU and memory resources for all hosts and virtual machines in the cluster. DRS compares these metrics to an ideal resource utilization given the attributes of the cluster's resource pools and virtual machines, the current demand, and the DRS Score, which measures the execution efficiency of the virtual machine. DRS then performs (or recommends) virtual machine migrations. When you first power on a virtual machine in the cluster, DRS attempts to maintain proper load balancing either by placing the virtual machine on an appropriate host or by recommending one.
- Power Management. When the VMware DPM (Distributed Power Management) feature is enabled, DRS compares cluster- and host-level capacity to the demands of the cluster's virtual machines, including recent historical demand. DPM places (or recommends placing) hosts in standby power mode if sufficient excess capacity is found. DPM powers on (or recommends powering on) hosts if capacity is needed. Depending on the resulting host power state recommendations, virtual machines might need to be migrated to and from the hosts.
- Virtual Machine Placement. You can control the placement of virtual machines on hosts within a cluster, by assigning DRS affinity or antiaffinity rules.

See Managing DRS Clusters.

VMware HA

VMware HA supports high availability for virtual machines by pooling the hosts they reside on into a cluster. If one host fails, its virtual machines move to a different host in the cluster.

VMware HA monitors ESXi hosts and in the event of host failure, migrates virtual machines to hosts with capacity. When you add new virtual machines to a VMware HA cluster, VMware HA checks whether enough capacity to power on that virtual machine is available on that host or a different host.

See Managing HA Clusters.

VMware HCI

VMware HCI manages the configuration of hyper-converged compute, storage, and network resources for a cluster, as a collection of identically configured ESXi hosts. This unified approach simplifies cluster configuration, offering a way to configure all resource types as a group, using a single interface.

VMware HCI clusters are ideal for newly established datacenters. Only conformant hosts are allowed to be a part of the HCI cluster. They must have the same CPU design and contain the minimum number of physical NICs specified in **`DvsProfile`**. Network settings are wiped to a clean state before configuration. Non-management port groups in standard switches are migrated to a virtual distributed switch.

The **`ConfigureHCI_Task`** method configures a VMware HCI (hyper converged infrastructure) cluster. The **`ExtendHCI_Task`** method expands and reconfigures a VMware HCI cluster. Before and after HCI cluster configuration, you can run the **`ValidateHCIConfiguration`** method to check that the **`ClusterComputeResourceHCIConfigSpec`** is correct.

The **`BatchAddHostsToCluster_Task`** method adds more hosts as needed to a VMware HCI cluster, but cannot change cluster configuration.

These configure, extend, and host add methods are composite tasks. That is, they perform multiple operations in a single task. For example, **`ConfigureHCI_Task`** can:

- Set up cluster services such as DRS, HA, and vMotion

- For an HCI cluster, verify that all hosts have same CPU vendor
- Add new standalone hosts and set state (lockdown)
- Move all hosts into the HCI cluster
- Configure vSAN datastore
- Configure virtual distributed switch and port groups

Automatic remediation of failed hosts is not attempted. The administrator must intervene to correct issues before attempting to place hosts back in the cluster. Remediated hosts must be ready to participate in the vSAN datastore.

Creating and Configuring Clusters

The vSphere Web Services SDK includes objects and methods for all cluster management tasks. Some documentation is available in the *vSphere API Reference*. For additional background and details about the failover and load balancing behavior, see *vSphere Resource Management* and *vSphere Availability*.

Creating a Cluster

If your environment includes a vCenter Server and multiple ESXi hosts, you can create a cluster by calling the **Folder.CreateClusterEx** method. You pass in a name for the new cluster and a **ClusterConfigSpecEx** data object. In the data object, you can specify the following properties:

- VMware DRS
  - **drsConfig** property in the **ClusterDrsConfigInfo** data object contains configuration information for the VMware DRS service. Properties in this object specify the cluster-wide (default) behavior for virtual machine and the threshold for generating cluster recommendations. You can enable and disable VMware DRS with the **ClusterDrsConfigInfo.enabled** property.
  - **drsVmConfigSpec** property in the **ClusterDrsVMConfigSpec** data object points to a **ClusterDrsVmConfigInfo** data object that specifies the DRS configuration for a single virtual machine. **ClusterDrsVmConfigInfo** overrides the default DRS configuration for an individual virtual machine and allows you to specify the DRS behavior and whether DRS can perform migration or recommend initial placement for a virtual machine.
    To update a DRS configuration, you call **ComputeResource.ReconfigureComputeResource_Task** and pass in a **ClusterConfigSpecEx** object. In the **ClusterConfigSpecEx.drsVmConfigSpec** property, you can specify an array of **ClusterDrsVMConfigSpec** objects that define the configuration for individual virtual machines.
  - **rulesSpec** property in the **ClusterRuleSpec** data object points a **ClusterRuleInfo** data object that specifies the affinity and antiaffinity rules DRS should use. See the *vSphere API Reference* entry for **ClusterRuleInfo**.
- VMware HA – distributed availability service (DAS)
  - **dasConfig** property in the **ClusterDasConfigInfo** data object specifies HA service on the cluster. Properties in this object determine whether strict admission control is enabled, what the default virtual machine settings in this cluster are, whether VMware HA restarts virtual machines after host failure, and so on. See the *vSphere API Reference*.
  - **dasVMConfigSpec** property in **ClusterDasVMConfigSpec** object, where the `info` property is a **ClusterDasVmConfigInfo** data object that specifies the HA configuration for a single virtual machine. You can apply different settings to different virtual machines, or use the default specified in the **dasConfig** property for all virtual machines in the cluster.
- VMware HCI
  - The **ClusterComputeResourceHCIConfigInfo** data object contains properties to capture the configured hosts, virtual distributed switch settings, host configurations, and HCI workflow state. The HCI cluster includes vSphere facilities for DRS, HA, vMotion, and vSAN.
  - The **ClusterComputeResourceHCIConfigSpec** data object specifies the virtual distributed switch settings, host configurations, enhanced vMotion capability, and vSAN configuration.

  - Once configurations are specified, you create an HCI cluster with the **ConfigureHCI_Task** method, and reconfigure an HCI cluster with the **ExtendHCI_Task** method.

Adding a Host to a Cluster

The methods available for adding hosts to a cluster are useful under different circumstances. Each method returns a managed object reference to a task.

- **ClusterComputeResource.AddHost_Task** adds a host to the cluster. The host may be specified by a numeric IP address or a DNS resolvable name. If the cluster supports nested resource pools and you pass an optional **resourcePool** argument, the host's resource pool hierarchy is imported into the new nested resource pool. If a cluster does not support nested resource pools, the host resource pool hierarchy is discarded and all virtual machines on the host are added to the cluster's root resource pool.
- **ClusterComputeResource.MoveInto_Task** moves a host in the datacenter into a cluster, or from one cluster into another. The **MoveHostInto_Task** method is similar, with extra **resourcePool** parameter.
- **Folder.BatchAddStandaloneHosts_Task** adds a list of **newHosts** to the inventory as standalone hosts. This operation works though the list of hosts and returns, in the task result, a list of hosts that were successfully added.

Reconfiguring a Cluster

You can alter a vSphere cluster by calling the **ComputeResource.ReconfigureComputeResource_Task** method. You can alter an HCI cluster by calling the **ClusterComputeResource.ExtendHCI_Task** method.

To reconfigure a vSphere cluster, for instance to enable or disable VMware DRS or VMware HA, you can change properties in the **ComputeResourceConfigSpec** data object, and pass it in to **ReconfigureComputeResource**_Task. For relevant properties, see Creating a Cluster. To enlarge a vSphere cluster, you can call **AddHost_Task**.

To enlarge an HCI cluster, possibly after adding hosts with **BatchAddStandaloneHosts_Task**, you can call **ClusterComputeResource.ExtendHCI_Task**, passing in the new hosts as parameter **hostInputs**.

To move a datacenter host into a cluster, or from one cluster to another if the host is in maintenance mode, you can call the **ClusterComputeResource.MoveInto_Task** method. The **MoveHostInto_Task** method is similar but allows moving the host into a nested resource pool.

To move hosts out of a cluster, and make them standalone hosts in the datacenter, you can call the **Folder.MoveIntoFolder_Task** method.

To remove a host from a cluster, and from the datacenter, you can call the **Folder.Destroy_Task** method, if you are granted the **Host.Inventory.RemoveHostFromCluster** privilege.

Managing DRS Clusters

The vSphere Client UI allows you to explore DRS cluster behavior, which is also described in the *vSphere Resource Management* guide. When DRS is running, it generates recommendations and associated information that result in a well balanced cluster, including:

- Initial placement of virtual machines
- Virtual machine migration for load balancing. Each migration recommendation has a rating, which you can find in the **ClusterRecommendation.rating** property. Client applications can choose to consider only high-priority migrations or migrations with multiple priority levels.
- Whether or not DRS clusters are valid, and have enough resources to start additional virtual machines.

DRS recommendations are stored in the **ClusterComputeResource.recommendation** property, which is an array of **ClusterRecommendation** data objects. Each **ClusterRecommendation** includes information about the action to perform and information you can use to display information to end users or for logging.

- Client applications can call **`ClusterComputeResource.ApplyRecommendation`** to apply one or more recommendations.
- For more fine-grained control, client applications can perform individual actions only. The **`ClusterRecommendation.action`** property is an array of **`ClusterAction`** objects. Each **`ClusterAction`** includes a target for the action and the type, which is a string that is one of the values of the **`ActionType`** enum (**`HostPowerV1`**, **`MigrationV1`**, **`VmPowerV1`**). Client applications can use the **`ActionType`** information to act on DRS recommendations by powering on hosts, migrating virtual machines, or powering on virtual machines by calling **`Datacenter.PowerOnMultiVM_Task`**.

Managing HA Clusters

You can add a host to an HA cluster by calling the **`AddHost_Task`** method, or move a host into a cluster by calling the **`MoveHostInto_Task`** method or similar.

You might have to call **`HostSystem.ReconfigureHostForDAS_Task`** to reconfigure the host for HA if the automatic HA configuration fails. See Adding a Host to a Cluster.

Primary and Secondary Hosts

You can add a secondary host to a cluster by calling the **`ClusterComputeResource.AddHost_Task`** method, which requires that you specify the host name, port, and password for the host to be added as a **`HostConnectSpec`**.

When you add a host to a VMware HA cluster, an agent is uploaded to the host and configured to communicate with other agents in the cluster. The first five hosts added to the cluster are designated as primary hosts, and all subsequent hosts are designated as secondary hosts. The primary hosts maintain and replicate all cluster state and are used to initiate failover actions. If a primary host is removed from the cluster, VMware HA promotes another host to primary status.

Any host that joins the cluster must communicate with an existing primary host to complete its configuration (except when you are adding the first host to the cluster). At least one primary host must be functional for VMware HA to operate correctly. If all primary hosts are unavailable (not responding), no hosts can be successfully configured for VMware HA.

One of the primary hosts is also designated as the active primary host and its responsibilities include:

- Deciding where to restart virtual machines
- Keeping track of failed restart attempts
- Determining when it is appropriate to keep trying to restart a virtual machine.

Failure Detection and Host Network Isolation

Agents on the different hosts contact and monitor each other through the exchange of heartbeats, by default every second. If a 15-second period elapses without the receipt of heartbeats from a host, and the host cannot be pinged, the host is declared as failed. The virtual machines running on the failed host are restarted on the alternate hosts with the most available unreserved capacity (CPU and memory).

Host network isolation occurs when a host is still running, but it can no longer communicate with other hosts in the cluster. With default settings, if a host stops receiving heartbeats from all other hosts in the cluster for more than 12 seconds, it attempts to ping its isolation addresses. If this also fails, the host declares itself isolated from the network.

When the isolated host's network connection is not restored for 15 seconds or longer, the other hosts in the cluster treat that host as failed and try to fail over its virtual machines. However, when an isolated host retains access to the shared storage it also retains the disk lock on virtual machine files. To avoid potential data corruption, VMFS disk locking prevents simultaneous write operations to the virtual machine disk files. Therefore attempts to fail over the isolated host's virtual machines do not succeed. By default, the isolated host leaves its virtual machines powered on, but you can change HA host isolation response as described in the *`vSphere Availability`* guide.

Using VMware HA and DRS Together

Using VMware HA with VMware DRS combines automatic failover with load balancing. This combination can result in faster rebalancing of virtual machines after VMware HA has moved virtual machines to different hosts.

When VMware HA performs failover and restarts virtual machines on different hosts, its first priority is the immediate availability of all virtual machines. After the virtual machines have been restarted, those hosts on which they were powered on might be heavily loaded, while other hosts might be comparatively lightly loaded. VMware HA uses the CPU and memory reservation to determine failover, but the actual usage might be higher.

In a cluster using DRS and VMware HA with admission control turned on, virtual machines might not be evacuated from hosts entering maintenance mode because of resources reserved to maintain the failover level. You must manually migrate the virtual machines off of the hosts using VMotion.

When VMware HA admission control is disabled, failover resource constraints are not passed on to DRS and VMware Distributed Power Management (DPM). The constraints are not enforced.

- DRS evacuates virtual machines from hosts and place the hosts in maintenance mode or standby mode regardless of the effect this might have on failover requirements.
- VMware DPM powers off hosts (place them in standby mode) even if doing so violates failover requirements.

The VMware HCI cluster includes VMware DRS and VMware HA features, so all the above remarks apply.

Tasks and Scheduled Tasks

VMware vSphere uses an asynchronous client-server communication model. Methods that end with _Task are non-blocking, returning a reference to a **`Task`** managed object. You can use **`Task`** and **`ViewManager`** managed objects to monitor tasks, cancel certain tasks, and create custom tasks.

If you are using a vCenter Server system, the **`ScheduledTaskManager`** allows you to schedule your own tasks for a one-time run or for repeated runs.

Creating Tasks

Each time a vSphere server runs a method, it creates a **`Task`** and a corresponding **`TaskInfo`** data object. Some methods run synchronously and return data as the **`Task`** completes. But methods that end with _Task run asynchronously, and return a reference to a Task that will be created and completed as a processor becomes available. They are created to perform the functions in a non-blocking manner. Therefore, you must use the reference to the **`Task`** to monitor the status and results of the Task. vSphere operations that include the suffix **`_Task`** in their names are asynchronous and return **`Task`** references.

The **`Task`** object provides information about the status of the invoked operation through its **`TaskInfo`** data object. An instance of **`TaskInfo`** populates the **`info`** property of the **`Task`** managed object at runtime. By monitoring properties of the **`TaskInfo`** object, a client application can take appropriate action when the **`Task`** completes, or can handle errors if the **`Task`** does not complete successfully.

When a vSphere server creates a **`Task`**, it also creates a **`TaskEvent`** object. The **`TaskEvent`** object contains a copy of the **`TaskInfo`** object (`TaskEvent.info`). The **`TaskEvent`** copy of the **`TaskInfo`** object is a snapshot of the Task state at the time of its creation. It does not change after it is created. To find the current status of the task, use the **`Task.info.eventChainId`** property.

Session Persistence

A **`Task`** and its associated objects are session specific, so they will not persist after the session is closed. When your client opens a session, you can only obtain information about the **`Task`** objects that your client is authorized to view.

Cancelling a Task

To cancel a **Task** that is still running, call the **Task.CancelTask** method, passing in the managed object reference to the **Task** you want to cancel, as shown in this example:

```
my_conn.cancelTask(taskMoRef);
```

You can only cancel a **Task** that has its **cancelable** property set to **true** and its **state** property set to **running**. The operation that initiates the **Task** sets the value of **cancelable** when it creates the **Task**. For example, a **CreateVM_Task** cannot be cancelled. Before attempting to cancel a running **Task**, you can check the values of the **cancelable** property and the **state** property of the **TaskInfo** data object associated with the **Task**.

Using TaskInfo to Determine Task Status

A **Task** object provides information about the status of the invoked operation through its **TaskInfo** data object. An instance of **TaskInfo** populates the **info** property of the **Task** managed object at runtime. By monitoring properties of the **TaskInfo** object, a client application can take appropriate action when the **Task** completes, or can handle errors if the **Task** does not complete successfully.

The `Task.info` property contains a **TaskInfo** data object that contains information about the **Task** the server returns to your client application.

When a **Task** is instantiated by the server, the **TaskInfo.result** property is initialized to **Unset**. Upon successful completion of an operation, the **result** property is populated with the return type specific to the operation. The **result** might be a data object, a reference to a managed object, or any other data structure as defined by the operation.

For example:

1. The **ClusterComputeResource.AddHost_Task** method returns a **Task** object whose **info** property contains a **TaskInfo** data object.
2. At the start of the operation, the **result** property is **Unset**.
3. Upon successful completion of the operation, the **result** property of **TaskInfo** contains the managed object reference of the newly added **HostSystem**.

TaskInfo Values

The following table lists some of the values obtained from a **TaskInfo** data object at the beginning and the end of the **Task** instantiated by the **CreateVM_Task** method.

Property	Datatype	Start of Task Sample Values	End of Task Sample Values
cancelable	boolean	false	false
...			
completeTime	dateTime	Unset	"2009-02-19T22:53:35.015338Z"
progress	int	36	100
queueTime	dateTime	"2009-02-19T22:50:39.111604Z"	"2009-02-19T22:50:39.111604Z"
reason	TaskReason	reason	reason

Table continued on next page

Continued from previous page

Property	Datatype	Start of Task Sample Values	End of Task Sample Values
result	**anyType**	**Unset**	**64**
....			
state	**TaskInfoState**	**"running"**	**"success"**

Monitoring TaskInfo Properties

To monitor the state of a **Task**, use the **PropertyCollector.WaitForUpdatesEx** method.

You can monitor the values of **TaskInfo** properties, which change as the **Task** runs to completion. For example, you can check the values of **startTime**, **queueTime**, **completeTime**, **progress**, **result**, and **state** as the operation progresses. Monitor these properties in your code in a separate thread until the **Task** completes, while the main line of your code continues with other activities.

For more information about monitoring properties, see Client Data Synchronization with WaitForUpdatesEx.

Your code must handle the datatype returned when the **Task** completes (managed object reference, data object, and so on). In addition to **success**, **queued**, and **running**, an operation can enter an **error** state, which your code must handle.

A **Task** object has a lifecycle that is independent of the **TaskManager** that creates it and independent of the entity with which it is associated. It exists to convey status about an operation. You can discard the reference to it when your application no longer needs the information.

The following example shows a code fragment that obtains values for the **info** property from each **Task** object in the array.

Displaying TaskInfoState Values for Tasks in recentTask Array

```
...
private void displayTasks(ObjectContent[] oContents) {
      for(int oci=0; oci<oContents.length; ++oci) {
         System.out.println("Task");
         DynamicProperty[] dps = oContents[oci].getPropSet();
         if(dps!=null) {
            String op="", name="", type="", state="", error="";
            for(int dpi=0; dpi<dps.length; ++dpi) {
               DynamicProperty dp = dps[dpi];
               if("info.entity".equals(dp.getName())) {
                  type = ((ManagedObjectReference)dp.getVal()).getType();
               } else if ("info.entityName".equals(dp.getName())) {
                  name = ((String)dp.getVal());
               } else if ("info.name".equals(dp.getName())) {
```

```
            op = ((String)dp.getVal());
         } else if ("info.state".equals(dp.getName())) {
            TaskInfoState tis = (TaskInfoState)dp.getVal();
            if(TaskInfoState.error.equals(tis)) {
               state = "-Error";
            } else if(TaskInfoState.queued.equals(tis)) {
               state = "-Queued";
            } else if(TaskInfoState.running.equals(tis)) {
               state = "-Running";
            } else if(TaskInfoState.success.equals(tis)) {
               state = "-Success";
            }
         } else if ("info.cancelled".equals(dp.getName())) {
            Boolean b = (Boolean)dp.getVal();
            if(b != null && b.booleanValue()) {
               state += "-Cancelled";
            }
        }...
```

Sample Run of the TaskList Java Application shows output from a run of the program. See the source code listing for `TaskList.java` or for **`TaskList.cs`** in the vSphere Web Services SDK package for details.

Sample Run of the TaskList Java Application

```
java com.vmware.samples.general.TaskList --url https://srv/sdk --username root --password
*******


Started

Task

Operation AcquireCimServicesTicket

Name srv

Type HostSystem

State -Success

Error

======================
```

```
Ended TaskList
```

Accessing and Manipulating Multiple Tasks

Use the `ViewManager`'s `ListView` method to identify the set of Tasks you want to monitor.

You can specify a smaller and more efficient data set using one of the **`ViewManager`** views with the Property Collector. Each view represents objects you have selected on the server. Views are more efficient because you only need a single instance of a **`PropertyCollector`** object, instead of multiple instances with multiple filter specifications.

Task Monitoring Example Using the Listview Object

You can use the ViewManager's **`CreateListView`** method to specify a set of tasks that you want to monitor with the PropertyCollector. This example shows how to initiate virtual machine cloning operations and monitor completion by tracking the `Task` objects.

For general task monitoring, a best practice is to use a **`ViewManager`** to monitor specific tasks. See the *`API Reference`* for more information about using views.

> **NOTE**
> For simplicity, the calls to `RetrievePropertiesEx()` and `WaitForUpdatesEx()` in this example ignore the possibility of additional chunks of data returned by the property collector. For more information about chunking in property collector results, see Server Data Transmission.

Create an Authenticated Session with the Server

Use command-line arguments to connect to a vSphere server and log in.

For this task you need:
- The Web Services API WSDL bindings file.
- The hostname of the vSphere server.
- A username and password to authenticate with the vSphere server.
- The name of a virtual machine available to be cloned.

Given the arguments on the command line, this task makes an SSL connection to the vSphere server, retrieves a reference to the SessionManager, and authenticates a session with the credentials passed on the command line.

1. Import the vSphere Web Services API libraries and the necessary Java (and JAX-WS connection, bindings, and SOAP) libraries:

```
import com.vmware.vim25.*;


import java.util.*;

import javax.net.ssl.HostnameVerifier;

import javax.net.ssl.HttpsURLConnection;

import javax.net.ssl.SSLSession;

import javax.xml.ws.BindingProvider;

import javax.xml.ws.soap.SOAPFaultException;
```

2. Create a class to hold data and methods.

```
public class cloneVMTask {
  static ManagedObjectReference pCollector;
  static ManagedObjectReference viewMgr;
  static ServiceContent serviceContent;
  static VimPortType methods;
```

3. Define a TrustManager implementation.

   Authentication is handled by using a TrustManager and supplying a host name verifier method.

   NOTE
   For the purposes of this example, this TrustManager implementation will accept all certificates. This is only appropriate for a development environment. Production code should implement certificate validation.

```
  private static class TrustAllTrustManager
     implements javax.net.ssl.TrustManager,
                javax.net.ssl.X509TrustManager {
    public java.security.cert.X509Certificate[] getAcceptedIssuers() {
      return null;
    }
    public boolean isServerTrusted(java.security.cert.X509Certificate[] certs) {
      return true;
    }
    public boolean isClientTrusted(java.security.cert.X509Certificate[] certs) {
      return true;
    }
    public void checkServerTrusted(java.security.cert.X509Certificate[] certs,
                                   String authType)
    throws java.security.cert.CertificateException {
      return;
    }
    public void checkClientTrusted(java.security.cert.X509Certificate[] certs,
                                   String authType)
    throws java.security.cert.CertificateException {
      return;
    }
```

```
} // end TrustAllTrustManager
```

4. Read command line arguments.

```
public static void main(String [] args) throws Exception {
  String serverName = args[0];
  String userName = args[1];
  String password = args[2];
  String vmName = args[3];
  String url = "https://"+serverName+"/sdk/vimService";
```

5. Create SSL context and session context for the connection.

```
  javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
  javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
  trustAllCerts[0] = tm;
  // Create the SSL context
  javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL");
  // Create the session context
  javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();
  // Initialize the contexts; the session context takes the trust manager.
  sslsc.setSessionTimeout(0);
  sc.init(null, trustAllCerts, null);
  // Use the default socket factory to create the socket for the secure connection

javax.net.ssl.HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
```

6. Set a host name verifier that will enable any connection.
   The host name verifier is invoked during the SSL handshake.

```
  HostnameVerifier hv = new HostnameVerifier() {
    public boolean verify(String urlHostName, SSLSession session) {
      return true;
    }
  };
  HttpsURLConnection.setDefaultHostnameVerifier(hv);
```

7. Create a VimService object to obtain a VimPort binding provider.

The BindingProvider provides access to the protocol fields in request/response messages. Retrieve the request context which will be used for processing message requests.

```
VimService vimService = new VimService();

methods = vimService.getVimPort();

Map<String, Object> ctxt = ((BindingProvider) methods).getRequestContext();
```

8. Store the Server URL in the request context.

   Specify **`true`** to maintain the connection between the client and server. The client API will include the Server's HTTP cookie in its requests to maintain the session. If you do not set this to true, the Server will start a new session with each request.

```
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);

ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);
```

9. Create a MOref for the ServiceIntsance and retrieve the ServiceContent.

```
// -- ManagedObjectReference for the ServiceInstance on the Server

ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();

SVC_INST_REF.setType("ServiceInstance");

SVC_INST_REF.setValue("ServiceInstance");

serviceContent = methods.retrieveServiceContent(SVC_INST_REF);
```

10. Use the SessionManager to log in.

```
methods.login(serviceContent.getSessionManager(),

              userName,

              password,

              null);
```

11. Get references to the PropertyCollector and the ViewManager.

```
pCollector = serviceContent.getPropertyCollector();

viewMgr = serviceContent.getViewManager();
```

With an authenticated session, you can use the PropertyCollector to retrieve the MOref of the virtual machine, then create clones and monitor the `Task` objects:

```
ManagedObjectReference vmRef = getVmRef( vmName );

cloneVM( vmRef );
```

A best practice is to close the session when done:

```
methods.logout(serviceContent.getSessionManager());

  } // end main

} // end cloneVMTask
```

Find Virtual Machine MOref

Use the PropertyCollector to get a managed object reference for a virtual machine named in the command line.

For this task you need:
- A virtual machine name is referenced by the String variable `vmName`.
- An authenticated Web Services session with the vSphere server that manages the virtual machine.
- A VimPort binding provider is referenced by the variable `methods`, which is attached to the vSphere server connection context.
- The ServiceInstance content is referenced by the variable `serviceContent`.
- The ViewManager object is referenced by the variable `viewMgr`.
- The PropertyCollector object is referenced by the variable `pCollector`.

Given the name of a virtual machine, this function retrieves references to all of the virtual machines in the datacenter and looks for a match to the specified name.

1. Declare a function that accepts a virtual machine name and returns a managed object reference.

```
private static ManagedObjectReference getVmRef( String vmName )

throws Exception

{

  ManagedObjectReference vmRef = null;
```

2. Use a container view to collect references to all virtual machines in the datacenter.

   Start the collection from the root folder at the top of the managed object hierarchy.

```
List<String> vmList = new ArrayList<String>();

vmList.add("VirtualMachine");

ManagedObjectReference cViewRef =

   methods.createContainerView(viewMgr,

                               serviceContent.getRootFolder(),

                               vmList,

                               true);
```

3. Create an **`ObjectSpec`** to define the beginning of the traversal.

   Use the **`setObj`** method to specify that the container view is the starting object for this traversal. Set the **`skip`** property to `true` to indicate that you don't want the PropertyCollector to collect properties from the container.

```
ObjectSpec oSpec = new ObjectSpec();

oSpec.setObj(cViewRef);

oSpec.setSkip(true);
```

4. Create a traversal spec to traverse from the starting object to the objects in the view.

   Set the **`skip`** property to `false` to indicate that you don't want the PropertyCollector to skip the objects reached by this traversal.

```
TraversalSpec tSpec = new TraversalSpec();
tSpec.setName("traverseEntities");
tSpec.setType("ContainerView");
tSpec.setPath("view");
tSpec.setSkip(false);
```

5. Add the traversal spec to the object spec.

```
oSpec.getSelectSet().add(tSpec);
```

6. Specify the property for retrieval (virtual machine name).

```
PropertySpec pSpec = new PropertySpec();
pSpec.setType("VirtualMachine");
pSpec.getPathSet().add("name");
```

7. Create a PropertyFilterSpec and add the object and property specs to it.

```
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);
```

8. Create a list for the filters and add the spec to it.

```
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);
```

9. Get the data from the server.

```
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(pCollector,fSpecList,ro);
```

10. Go through the returned list and look for a match to the specified **`vmName`**.

```
if (props != null) {
  for (ObjectContent oc : props.getObjects()) {
    String vmname = null;
    List<DynamicProperty> dps = oc.getPropSet();
    if (dps != null) {
      for (DynamicProperty dp : dps) {
        vmname = (String) dp.getVal();
        // If the name of this virtual machine matches
        // the specified name, save the managed object reference.
```

```
            if (vmname.equals(vmName)) {
              vmRef = oc.getObj();
              break;
            }
          }
          if (vmRef != null) { break; }
        }
      }
    }
    if (vmRef == null) {
      System.out.println("Specified Virtual Machine not found.");
      throw new Exception();
    }
    return vmRef;
  } // end getVmRef
```

Given the MOref of the named virtual machine, you can use the PropertyCollector to retrieve ia MOref to its parent folder, which you will need to clone the virtual machine.

Get_Parent_Property

Given the managed object reference for a virtual machine object, use the PropertyCollector to get the managed object reference for its parent object.

For this task you need:
- A virtual machine managed object reference in a variable named `vmRef`.
- An authenticated Web Services session with the vSphere server that manages the virtual machine.
- A VimPort binding provider referenced by the variable `methods`, which is attached to the vSphere server connection context.
- A PropertyCollector instance referenced by the variable `pCollector`.

Given the MOref of a virtual machine, this function uses the PropertyCollector to retrieve the MOref of the parent of the virtual machine object. The MOref of the parent folder can be used to clone the virtual machine.
A `VirtualMachine` can be a child of either a `Folder` object or a `VirtualApp` object. If it is a child of a `Folder` object, the `parent` property contains the MOref of the `Folder` object and the `parentVApp` property is null. If the `VirtualMachine` is a child of a `VirtualApp` object, the `parent` property is null and the `parentVApp` property contains the MOref of the `VirtualApp` object. The method in this example returns a key-value pair from which the caller can determine which kind of parent the `VirtualMachine` has.

1. Declare a function that accepts a virtual machine MOref and returns a MOref to its parent object.

```
private static DynamicProperty getVMParentProperty(ManagedObjectReference vmRef)
throws Exception {
```

2. Create an **`ObjectSpec`** to define the property collection. Use the **`setObj`** method to specify that the vmRef is the starting object for this property collection. Set the **`skip`** property to `false` to indicate that you want to collect properties from the starting object. Omit the `selectSet` property because the property collection does not need to traverse away from the starting object.

```
ObjectSpec oSpec = new ObjectSpec();
oSpec.setObj(vmRef);
oSpec.setSkip(false);
```

3. Specify the properties for retrieval (`VirtualMachine.parent` and `VirtualMachine.parentVApp`).

   Because the `VirtualMachine` has two mutually exclusive parent properties, depending on the parent type, this code collects both properties.

```
PropertySpec pSpec = new PropertySpec();
pSpec.setType("VirtualMachine");
pSpec.getPathSet().add("parent");
pSpec.getPathSet().add("parentVApp");
```

4. Create a PropertyFilterSpec and add the object and property specs to it.

```
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);
```

5. Create a list for the filters and add the property filter spec to it.

```
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);
```

6. Use the filter spec to retrieve the property collection from the server.

```
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(pCollector,fSpecList,ro);
```

7. Unwrap the parent folder property.

```
if (props != null) {
  for (ObjectContent oc : props.getObjects()) {
    List<DynamicProperty> dps = oc.getPropSet();
    if (dps != null) {
      for (DynamicProperty dp : dps) {
        if (dp.getName().equals("parent") || dp.getName().equals("parentVApp")) {
          return dp;
        }
      }
```

```
        }
      }
    }
    System.out.println("Parent not found.");
    throw new Exception();
  }
```

Now that you have the reference information for the virtual machine that you specified on the command line (**`vmRef`**) and a reference for the parent (in the **`dp`** property), you are ready to extract the type and MOref of the parent.

Create Clones of the Virtual Machine

Create two clones of the virtual machine and build a PropertyFilterSpec that uses a ListView managed object to track the Tasks to completion.

For this task you need:
- A virtual machine managed object reference in a variable named `vmRef`.
- An authenticated Web Services session with the vSphere server that manages the virtual machine.
- A VimPort binding provider referenced by the variable `methods`, which is attached to the vSphere server connection context.
- The ViewManager object is referenced by the variable `viewMgr`.
- A PropertyCollector instance referenced by the variable `pCollector`.

Given the MOref of a virtual machine and the MOref of its parent folder, this function initiates two clone operations, resulting in two `Task` managed objects that you can use to monitor progress of the clone operations. The function uses the `Task` objects to build a `PropertyFilterSpec` that you can use to report status updates for the two `Task` objects.

1. Declare a method that accepts a virtual machine MOref to clone it and monitor the `Task` objects.

   ```
   private static void cloneVM(ManagedObjectReference vmRef) throws Exception {
   ```

2. Use the `getVMParent` method to get a MOref for the parent folder of the virtual machine.

   If the parent object is a `Folder`, the `CloneVMTask` method will create the clones in the same folder that contains the specified virtual machine. If the parent object is a `VirtualApp`, the `VirtualMachine` cannot be cloned.

   ```
   DynamicProperty parent = getVMParentProperty( vmRef );
   if (parent.getName().equals("parentVApp")) {
     System.out.println("Will not clone VM in a VApp.");
     throw new Exception();
   } else {
     ManagedObjectReference folder = (ManagedObjectReference) parent.getVal();
   }
   ```

3. Create a clone specification. Use default values whenever possible.

```
VirtualMachineCloneSpec cloneSpec = new VirtualMachineCloneSpec();
VirtualMachineRelocateSpec vmrs = new VirtualMachineRelocateSpec();
cloneSpec.setLocation(vmrs);
cloneSpec.setPowerOn(true);
cloneSpec.setTemplate(false);
```

4. Create two clone virtual machines.

```
ManagedObjectReference cloneTask = methods.cloneVMTask( vmRef,
                                                         folder,
                                                         "clone__1",
                                                         cloneSpec);
ManagedObjectReference cloneTask2 = methods.cloneVMTask( vmRef,
                                                         folder,
                                                         "clone__2",
                                                         cloneSpec);
```

   The `CloneVMTask` method returns a `Task` object from each invocation. You can use PropertyCollector to track task completion.

5. Create a list view to reference the clone tasks.

```
List<ManagedObjectReference> taskList = new ArrayList<ManagedObjectReference>();
taskList.add(cloneTask);
taskList.add(cloneTask2);
ManagedObjectReference cloneTaskList = methods.createListView(viewMgr,
                                                              taskList);
```

6. Create an object spec to describe the property collection.

   Add the list of `Task` objects to the object spec. Set the `skip` property to `true` so the PropertyCollector will ignore the properties of the `ListView` object itself.

```
ObjectSpec oSpec = new ObjectSpec();
oSpec.setObj(cloneTaskList);
oSpec.setSkip(true);
```

7. Create a traversal spec to tell the PropertyCollector how to move from the `ListView` object to the `Task` objects.

   Set the `skip` property to `false` so the PropertyCollector will try to collect properties from the managed objects at the end of this traversal step.

```
TraversalSpec tSpec = new TraversalSpec();
tSpec.setName("traverseTasks");
```

```
tSpec.setPath("view");

tSpec.setSkip(false);

tSpec.setType("ListView");
```

8. Add the traversal spec to the object spec.

```
oSpec.getSelectSet().add(tSpec);
```

9. Create a property spec to collect the `Task.info.state` and `Task.info.result` properties.

```
PropertySpec pSpec = new PropertySpec();

pSpec.setType("Task");

pSpec.setAll(false);

pSpec.getPathSet().add("info.state");

pSpec.getPathSet().add("info.result");
```

10. Create a filter spec from the object spec and the property spec, and use the filter spec to create a `PropertyFilter` object to guide the property collection.

    The filter managed object needs to persist on the server until the PropertyCollector finishes reporting task status.

```
PropertyFilterSpec fSpec = new PropertyFilterSpec();

fSpec.getObjectSet().add(oSpec);

fSpec.getPropSet().add(pSpec);

ManagedObjectReference pFilter = methods.createFilter(pCollector,
                                                       fSpec,
                                                       true);
```

Two cloning operations are underway, with status of the operations reflected in `Task` objects. A `PropertyFitler` managed object is created and configured to monitor the task status.

Use the `PropertyCollector.WaitForUpdatesEx` method to monitor the status of the `Task` objects.

Monitor Tasks with WaitForUpdatesEx

Use the `PropertyCollector.WaitForUpdatesEx` method to monitor and report `Task` state changes.

For this task you need:
- An authenticated Web Services session with the vSphere server that manages the virtual machine.
- Managed object references for two `Task` objects in the variables `cloneTask` and `cloneTask2`.
- The ViewManager object is referenced by the variable `viewMgr`.
- The PropertyCollector object is referenced by the variable `pCollector`.

This example uses the **`waitForUpdatesEx`** method to look for a change in the **`info.state`** property and the **`info.result`**.property of `cloneTask` and `cloneTask2`. When the state is `"success"`, **`info.result`** is the managed object reference of the clone virtual machine.

NOTE
The order of property retrieval is not guaranteed, and it may take more than one call to **`waitForUpdatesEx`** to retrieve both properties for a task

.

1. Initialize variables for the wait loop.

```
String version = "";

Boolean wait = true;

WaitOptions waitOptions = new WaitOptions();


while ( wait ) {
```

2. Call WaitForUpdatesEx.

   If `waitForUpdatesEx` returns `null`, that indicates that the call has timed out.

```
UpdateSet uSet = methods.waitForUpdatesEx(pCollector,
                                          version,
                                          waitOptions);


if (uSet == null) {
  wait = false;
} else {
```

3. Save the `version` for subsequent calls to `WaitForUpdatesEx`.

```
  version = uSet.getVersion();
```

4. Get the list of property updates.

```
  List<PropertyFilterUpdate> pfUpdates = uSet.getFilterSet();
  for (PropertyFilterUpdate pfu : pfUpdates) {
```

5. Get the list of object updates produced by the filter.

```
    List<ObjectUpdate> oUpdates = pfu.getObjectSet();
    for (ObjectUpdate ou : oUpdates) {
```

6. Look for `ObjectUpdate.kind=="MODIFY"` (property modified).

```
      if (ou.getKind() == ObjectUpdateKind.MODIFY) {
        String name = "";
        TaskInfoState state;
```

```
ManagedObjectReference cloneRef = new ManagedObjectReference();
```

7. Get the changed data.

```
List<PropertyChange> pChanges = ou.getChangeSet();
```

8. Output the property changes.

   If the task completes successfully, the `result` property contains the MOref of the new clone.

```
for (PropertyChange pc : pChanges) {
  name = pc.getName();
  if (name.equals("info.state")) {
    state = (TaskInfoState)pc.getVal();
    System.out.println("State is "+state.value());
  } else if (name.equals("info.result")) {
    cloneRef = (ManagedObjectReference)pc.getVal();
    System.out.println("Clone reference is "+cloneRef.getValue());
  } // end if name.equals
 } // end for pc
 } // end if ou.getKind
 } // end for ou
 } // end for pfu
 } // end if user
 } // end while wait
} // end cloneVM
```

This code does not set a time-out (**`WaitOptions.maxWaitSeconds`** is unset), so after it has retrieved all of the property changes, **`waitForUpdatesEx`** will block the thread, waiting for the TCP connection with the vSphere Server to time out.

How a client application handles the session depends on the particular context. For example, the client can call **`WaitForUpdatesEx`** from its own thread, look for specific updates and then stop calling the method. For the preceding example, you might choose to watch for the `Task.info.state` to transition to a value of `success` or `error`.

For more information about **`WaitOptions`** and the **`waitForUpdatesEx`** method, see Client Data Synchronization with WaitForUpdatesEx.

Gathering Data with a TaskManager Interface

TaskManager is a service interface that you can also use for accessing and manipulating **Task** objects. This approach uses a **PropertyCollector** that includes the **recentTask** property of the **TaskManager** managed object that corresponds to the Recent Tasks pane at the bottom of the vSphere client User Interface.

You can use the following **TaskManager** properties in your client application.

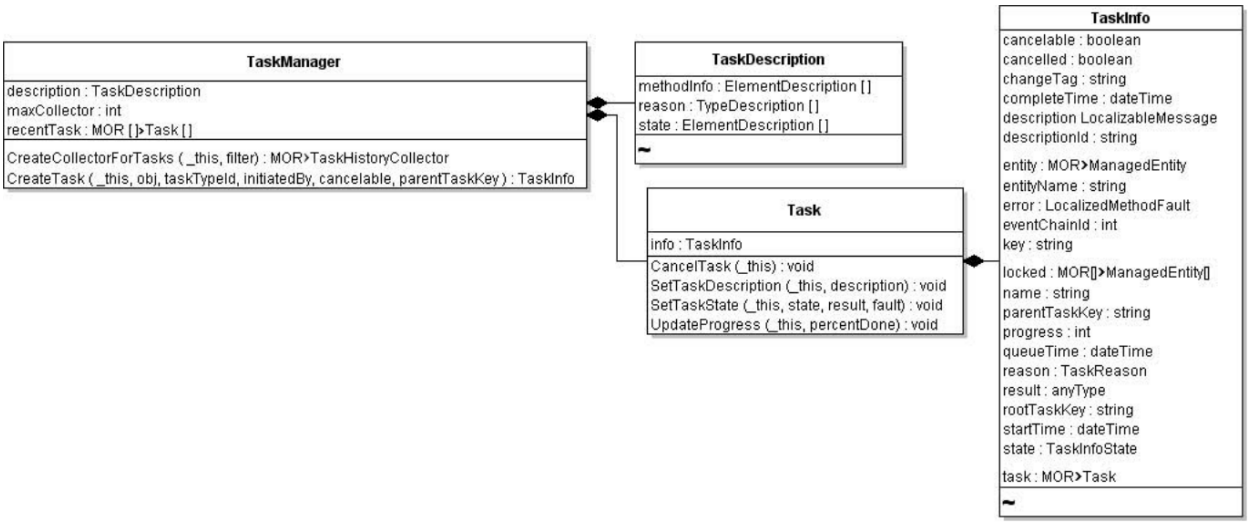
- **description** – **TaskDescription** object that includes a **methodInfo** property. **methodInfo** contains a key-based array that **TaskManager** uses to populate the value of a **TaskInfo** data object's **descriptionId** property with the name of the operation. Examples of two elements from this key-based array are **methodInfo["Folder.createVm"]** and **methodInfo["Folder.createClusterEx"]**.
- **recentTask** – Array of **Task** managed object references that are queued to run, running, or completed within the past 10 minutes. On ESX/ESXi hosts that are managed by a vCenter Server, a completed task must also be one of the 200 most recent tasks to be included in the array. A vSphere Client connected to a vSphere Server displays queued, running, and completed tasks in the Recent Tasks pane.

In addition to these properties, **TaskManager** has the following methods:

- **CreateTask** – Used by other methods to create a custom **Task** object. Developers creating extensions can use this method to create custom **Task** objects.
- **CreateCollectorForTasks** – Creates an object that contains all tasks from the vCenter Server database that meet specific criteria. You cannot run this method against an ESX/ESXi system. See Using a TaskHistoryCollector.

Figure: TaskManager and Task Managed Objects shows a UML class diagram for **TaskManager** and associated objects.

Figure 55: TaskManager and Task Managed Objects



Examining Recent Tasks with TaskManager

To obtain the list of recent tasks, use a **PropertyCollector** to obtain references to the **TaskManager** and to all **Task** objects from the **recentTask** property of the **TaskManager**.

The following example shows an excerpt from the `TaskList.java` sample that creates the **ObjectSpec**, **PropertySpec**, and a **TraversalSpec** to obtain references to all **Task** objects on the server from the **TaskList**. See also Property Collector.

PropertyFilterSpec Definition to Obtain recentTask Property Values

```
private PropertyFilterSpec[] createPFSForRecentTasks(ManagedObjectReference
taskManagerRef) {

     PropertySpec pSpec = new PropertySpec();

          pSpec.setAll(Boolean.FALSE);

          pSpec.setType("Task");

          pSpec.setPathSet(new String[] {"info.entity", "info.entityName", "info.name",

              "info.state", "info.cancelled", "info.error"});

     ObjectSpec oSpec = new ObjectSpec();

          oSpec.setObj(taskManagerRef);

          oSpec.setSkip(Boolean.FALSE);

     TraversalSpec tSpec = new TraversalSpec();

          tSpec.setType("TaskManager");

          tSpec.setPath("recentTask");

          tSpec.setSkip(Boolean.FALSE);

     oSpec.setSelectSet(new SelectionSpec[]{tSpec});

     PropertyFilterSpec pfSpec = new PropertyFilterSpec();

          pfSpec.setPropSet(new PropertySpec[]{pSpec});

          pfSpec.setObjectSet(new ObjectSpec[]{oSpec});

     return new PropertyFilterSpec[]{pfSpec};

}
```

For ESXi hosts managed by vCenter Server, use a **`TaskHistoryCollector`**. See Using a TaskHistoryCollector.

Understanding the ScheduledTaskManager Interface

You can use the **`ScheduledTaskManager`** to schedule tasks. In the vSphere Client, scheduled tasks display in the Task & Events tab.

You can define actions to occur on vCenter Server at different times:

- When a vCenter Server system starts up operations, such as after a reboot
- At a specific time and day
- At hourly, daily, weekly, or monthly intervals

You can schedule scripts to be run or methods to be invoked on the server. You apply the action to an entity in the inventory, such as a virtual machine or a host.

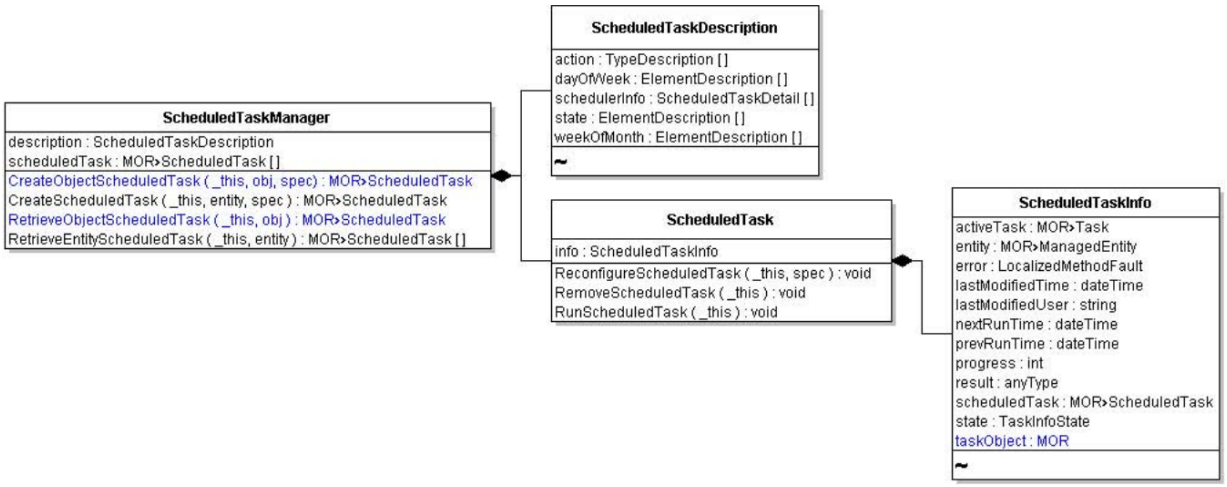
You can perform the following actions with **`ScheduledTaskManager`**.

- Retrieve scheduled tasks for a specific managed entity by calling the **`ScheduledTaskManager.RetrieveEntityScheduledTask`** method.

- Create a scheduled task by calling the **ScheduledTaskManager.CreateScheduledTask** method. See Scheduling Tasks.

Figure: ScheduledTaskManager and ScheduledTask Managed Objects shows the **ScheduledTaskManager** service interface and associated data objects.

Figure 56: ScheduledTaskManager and ScheduledTask Managed Objects



The **ScheduledTaskManager.scheduledTask** property contains an array of the **ScheduledTask** objects configured for the server. If you have no actions scheduled, this property is empty. For any **ScheduledTask** objects in this array, you can use the **info** property of the **ScheduledTask** object to obtain information about the status of the scheduled action. Information includes the task's **progress**, **state**, previous and next runtimes, and other details contained in the **ScheduledTaskInfo** data object.

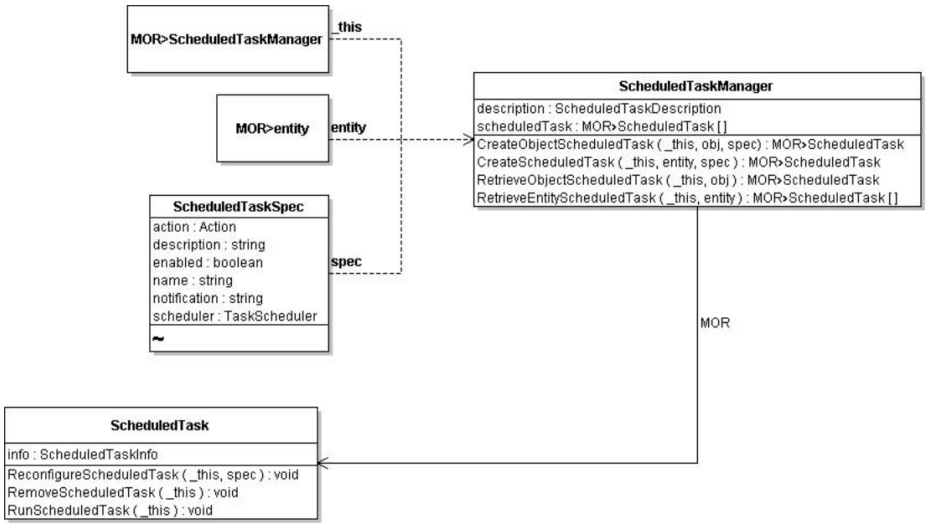
If the action specified for a **ScheduledTask** creates its own **Task** (such as with any of the asynchronous operations), the managed object reference to the **Task** populates the **activeTask** property of **ScheduledTaskInfo**.

Scheduling Tasks

You create a **ScheduledTask** by invoking the **ScheduledTaskManager.CreateScheduledTask** method. When you invoke the method, you include a **ScheduledTaskSpec** object that defines the schedule and specifies the action to take at the specified time. A scheduled action applies to an object based on these rules:

- If you specify a container object as the entity for the scheduled action, the schedule applies to all entities that are direct descendants of the container. You can set a **ScheduledTask** at the **Folder**, **Datacenter**, or **VirtualApp** level and have the scheduled action apply to all entities associated with the **Folder**, **Datacenter**, or **VirtualApp**.
- If you specify a node object in the inventory, such as a virtual machine, the action applies only to the virtual machine.

Figure 57: Using ScheduledTaskManager to Create a ScheduledTask



Defining the Schedule and Action

The **`ScheduledTaskSpec`** data object contains all the information to create a **`ScheduledTask`**.

- **`action`** – Action to take when the **`ScheduledTask`** runs. Specify an **`Action`** data object, which is an abstract type that is extended by several specific action types. The **`Action`** data objects are also used by the **`Alarm`** infrastructure. See Specifying Alarm Actions.
- **`notification`** – Specifies the email address for sending notification messages about the **`ScheduledTask`**. To use notifications, the vCenter Server system must have an SMTP email gateway configured. By default, **`notification`** is set to an empty string.
- **`scheduler`** – Specifies the time, frequency, and other details of the schedule. The **`TaskScheduler`** data object is the base type for several specific schedule objects. See Scheduling Recurring Operations.

Scheduling Recurring Operations

You can specify the times, days, or frequency of scheduled tasks by creating the appropriate instances of **`TaskScheduler`** subtypes and setting the **`scheduler`** property of the **`ScheduledTaskSpec`**.

The **`TaskScheduler`** base type has two properties:

- **`activeTime`** is the time at which the action should occur. If you leave this property unset, it defaults to the time when that specification for the scheduled task was submitted to the server.
- **`expireTime`** is the time after which the scheduled action should not occur. By default, this property is unset, so the scheduled task does not expire.

The following table provides some usage information about the **`TaskScheduler`** subtypes. The examples in the table are Java code fragments.

TaskScheduler Subtype	Usage
`AfterStartupTaskScheduler`	Schedule a task to start as soon as the vCenter Server system is started, or at a defined time after startup. The value must be zero (task triggered at startup) or higher.

Table continued on next page

Continued from previous page

TaskScheduler Subtype	Usage
	Example: Schedule a task to run 10 minutes after vCenter Server startup.

	`AfterStartupTaskScheduler asts =`
	`new AfterStartupTaskScheduler();`
	`asts.setMinute(10);`

`OnceTaskScheduler`	Schedule an action to run once only at the specified date and time. Example: Schedule a task to run 30 minutes after the schedule is submitted to the server.

	`Calendar runTime = Calendar.getInstance();`
	`runtime.add(Calendar.MINUTE, 30);`
	`OnceTaskScheduler ots = new OnceTaskScheduler();`
	`ots.setRunAt(runTime);`

`RecurrentTaskScheduler`	Base type for **`HourlyTaskScheduler`**, **`DailyTaskScheduler`**, **`WeeklyTaskScheduler`**, and **`MonthlyTaskScheduler`** objects. Set the interval property to define how frequently a task should run. For example, setting the interval property of an hourly task to 4 causes the task to run every 4 hours.
`HourlyTaskScheduler`	Schedule a task to run once every hour (or every specified number of hours) at a specified time. Set the interval property to run the task after a specified number of hours. Example: Schedule a task to run every 4 hours at half-past the hour.

	`HourlyTaskScheduler hts =`
	`new HourlyTaskScheduler();`
	`hts.setMinute(30);`
	`hts.setInterval(4);`

`DailyTaskScheduler`	Schedule a task to run daily or a specified number of days at a specified time (hour and minutes). Use in conjunction with the interval property to run the task after a specified number of days. Example: Schedule a task to run daily at 9:30 am (EST).

	`DailyTaskScheduler dts =`
	`new DailyTaskScheduler();`
	`dts.setMinute(30);`
	`dts.setHour(14);`

`WeeklyTaskScheduler`	Schedule a task to run every week (or every specified number of weeks) on a specified day (or days) at a specific time. The hours and minutes are set as UTC values. At least one of the boolean values must be set to **`true`**. You can also set the interval property to run the task after a specified number of weeks. Example: Schedule a task to run every Tuesday and Sunday at 30 minutes past midnight.
	`WeeklyTaskScheduler wts =`

Table continued on next page

Continued from previous page

TaskScheduler Subtype	Usage

	`new WeeklyTaskScheduler();`
	`wts.setMonday(true);`
	`wts.setTuesday(true);`
	`...`
	`wts.setSaturday(false);`
	`wts.setSunday(true);`
	`dts.setMinute(30);`
	`dts.setHour(4);`

MonthlyByDayTaskScheduler	Schedule a task to run every month (or every specified number of months) on a specified day at a specified time (hour and minutes). You can also set the interval property to run the task after a specified number of months. Example: Schedule a task to run every 3 months (on the last day of the month) at 12:30 p.m.

	`MonthlyByDayTaskScheduler mbdts =`
	`new MonthlyByDayTaskScheduler();`
	`mbdts.setDay(31);`
	`mbdts.setInterval(3);`
	`mbdts.setMinute(30);`
	`mbdts.setHour(14);`

MonthlyByWeekdayTaskScheduler	Schedule a task to run every month (or every specified number of months) on a specified week, weekday, and time (hour: minutes). You can also set the interval property to run the task after a specified number of months. Example: Schedule a task to run on the last Wednesday of each month at 12:30 a.m.

	`MonthlyByWeekdayTaskScheduler mbwts =`
	`new MonthlyByWeekdayTaskScheduler();`
	`mbwts.setOffset(WeekOfMonth.last);`
	`mbwts.setWeekday(DayOfWeek.wednesday);`
	`mbwts.setHour(4);`
	`mbwts.setMinute(30);`

The **hour** and **minute** properties of all objects that extend the **RecurrentTaskSchedule** data object are specified in Coordinated Universal Time (UTC) values rather than the local time of the server. When you define the schedule, convert your local time to a UTC value.

The following code fragment defines a **ScheduledTask** that powers on virtual machines daily at 4:15 a.m., if the server local time is in the Pacific Standard Time (PST) time zone. For a server in the Eastern European Summer Time (EEST) zone, the setting is read by the system as 3:15 pm.

Scheduled Task for Powering-on Virtual Machines

```
...
// Set the schedule using the DailyTaskScheduler subtype.
DailyTaskScheduler dTScheduler = new DailyTaskScheduler();
dTScheduler.setHour(12);
dTScheduler.setMinute(15);
ScheduledTaskSpec tSpec = new ScheduledTaskSpec();
tSpec.setDescription("Start virtual machine as per schedule.");
tSpec.setEnabled(Boolean=TRUE);
tSpec.setName("Power On Virtual Machine");
tSpec.setAction(ma);
tSpec.setScheduler(dTScheduler);
tSpec.setNotification("admin@vmware.com");
my_conn.createScheduledTask(_sic.getScheduledTaskManager, vmRef, tSpec);
...
```

Cancelling a Scheduled Task

You can cancel a scheduled task in several ways.

- To cancel the current run of a scheduled task, call **ScheduledTask.RemoveScheduledTask**. This method does not cancel subsequent runs of the **ScheduledTask**.
- To cancel an upcoming run of a **ScheduledTask**, call **ScheduledTask.ReconfigureScheduledTask** with a new **ScheduledTaskSpec** data object containing the new specifications for the schedule.
- To cancel a **ScheduledTask** that spawns a second task, create a **PropertyCollector** to obtain the reference to the **Tasks** and call its **CancelTask** method. The task must be cancellable.

Using a TaskHistoryCollector

A **TaskHistoryCollector** lets you gather information about tasks. You create a **TaskHistoryCollector** using the **TaskManager.CreateCollectorForTasks** method.

To create a TaskHistoryCollector

1. Identify the type of **Task** objects that you want to collect, and create an instance of a **TaskFilterSpec** data object that specifies your filter criteria.
   The **TaskFilterSpec** includes an **taskTypeId** property, which you use to limit the set of collected task objects to specific types. You can also provide a time range in the **TaskFilterSpec** by defining an **TaskFilterSpecByTime** data object for its **time** property. See the *vSphere API Reference*.
2. Obtain the managed object reference to the **TaskManager** on your server instance.

3. Submit the **filter** and the reference to the server in the **CreateTaskHistoryCollector** method. The server returns a reference to a **TaskHistoryCollector** object.

After a **HistoryCollector** has been created, the server appends new objects that meet the filter criteria to the collection as they occur. The system appends the new object to the collection by placing it in the first position of the **latestPage** and removes the oldest object from the collection. The **latestPage** property of the **TaskHistoryCollector** object has a property that consists of the 1000 most recent objects in the collection. Use a **PropertyCollector** to obtain the items from the **latestPage** property.

A **HistoryCollector** exists only for the duration of the session that instantiated it. Call the **HistoryCollector.DestroyCollector** method to delete the collector before the session ends.

Creating a TaskHistoryCollector Filter

When you create a **TaskHistoryCollector**, you can define filters. For example, rather than returning all **Task** objects associated with virtual machines, you might create a filter to collect only **Task** objects associated with virtual machines that were started by the backup-administrator between 2:00 and 4:00 a.m. on a specific date.

The **TaskFilterSpec** object allows you to specify the collection criteria. Most of the properties are optional and can be submitted as **null** values. The **TaskFilterSpec** lets you collect tasks based on user name, entity type, time, and state of the **Task**.

Managing the HistoryCollector

The **HistoryCollector** managed object provides operations for managing the life-cycle and scrollable view of a collection.

- **DestroyCollector** – A **HistoryCollector** exists only for the current session. Invoke the **DestroyCollector** operation to explicitly destroy the collector before the session ends.
- **ResetCollector** – Adjusts the starting position for the subset of objects from the collector to the object immediately preceding the current **latestPage**.
- **RewindCollector** – Positions the **latestPage** to the oldest item in the array. When a **HistoryCollector** is created, this is the default location.
- **SetCollectorPageSize** – Accepts an integer parameter to set the size of the **latestPage** property of a **HistoryCollector**. The default size of a **HistoryCollector** is an array that consists of at most 1000 objects of the appropriate type (**Task**, **Event**). The array is sorted by creation date and time of the objects.

Sample Code for Tasks

The following table lists the sample applications included with the vSphere Web Services SDK that demonstrate **Task** monitoring.

Java
vsphere-ws\java\JAXWS\samples\com\vmware\general\ TaskList.java
vsphere-ws\java\JAXWS\samples\com\vmware\scheduling\ DeleteOneTimeScheduledTask.java
vsphere-ws\java\JAXWS\samples\com\vmware\scheduling\ OneTimeScheduledTask.java
vsphere-ws\java\JAXWS\samples\com\vmware\scheduling\ WeeklyRecurrenceScheduledTask.java

Events and Alarms

Events are sent by vSphere to convey information about things that happen in the system. You can monitor events directly or use an **`EventHistoryCollector`** to retrieve events from a certain period.

Alarms are sent by vSphere to alert users to problems. You can also create your own alarm to monitor the system and set up follow-up actions. Alarm setup includes specifying the trigger condition and defining the action that should result.

Event and Alarm Management Objects

`EventManager` is the service interface for working with the event infrastructure.

`Event` subtypes define the events that the system generates. See Event Data Objects and Creating Custom Events.

`EventHistoryCollector` allows you to monitor events. You can create a filter to limit the number of events your code retrieves. You can monitor both system events and your own events. See Using an EventHistoryCollector.

The **`AlarmManager`** is the service interface for creating, setting, and managing alarms. You create an alarm, specifying trigger conditions and the action to take. When the conditions defined for the **`Alarm`** occur on the system, the **`Action`** specified for the alarm starts. The alarm also generates an **`Event`** that you can retrieve with an **`EventHistoryCollector`**.

Understanding Events

An **`Event`** is a data object type that contains information about state changes of managed entities and other objects on the server. Events include user actions and system actions that occur on datacenters, datastores, clusters, hosts, resource pools, virtual machines, networks, and distributed virtual switches. For example, these common system activities generate one or more **`Event`** data objects:

- Powering a virtual machine on or off
- Creating a virtual machine
- Installing VMware Tools on the guest OS of a virtual machine
- Reconfiguring a compute resource
- Adding a newly configured ESXi system to a vCenter Server system

In the vSphere Client, information from **`Event`** objects generated on a standalone ESXi hosts displays in the Events tab. For managed hosts, information from **`Event`** objects displays in the Tasks & Events tab.

Persistence of **`Event`** objects depends on the system setup.

- Standalone ESXi hosts – **`Event`** objects are not persistent. Events are retained only for as long as the host system's local memory can contain them. Rebooting a standalone ESXi host or powering off a virtual machine removes **`Event`** objects from local memory.
  A standalone ESXi host might keep about 15 minutes worth of **`Event`** data, but this can vary depending on the processing load of the host, the number of virtual machines, and other factors.
- Managed ESXi systems. **`Event`** objects are persistent. Managed ESXi systems send **`Event`** data to the vCenter Server system that manages them, and the vCenter Server system stores the information its database.

You can use the event sample applications included in the SDK package with either managed or standalone ESXi systems and with vCenter Server systems.

Using an **`EventHistoryCollector`**, you can obtain information about these objects as they are being collected on a specific ESXi system, or from a specific historical period from the database. See Using an EventHistoryCollector.

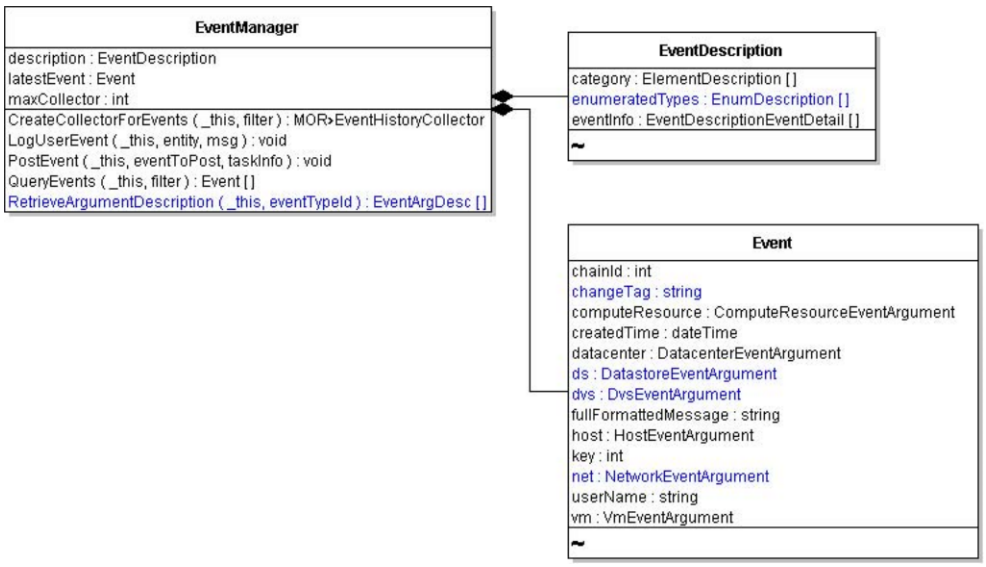
Managing Events with EventManager

`EventManager` is the service interface for working with the event infrastructure.

Figure: EventManager Managed Object and Associated Objects shows **`EventManager`** and related objects. An **`EventManager`** has these properties:

- A **`description`** property, defined as an instance of an **`EventDescription`** data object, which contains an event category and other information.
- A **`latestEvent`** property that contains the most recent **`Event`** data object in memory.
- A **`maxCollector`** property that specifies the number of **`EventHistoryCollector`** objects per client session that can be created. This value is set by the vCenter Server system.

Figure 58: EventManager Managed Object and Associated Objects



Event Data Objects

`Event` subtypes define the events that the system generates.

Figure: Event Data Object and Sample Subtypes shows only a few of the subtypes that extend the **`Event`** data object. For example, **`TaskEvent`** inherits all **`Event`** properties and includes an **`info`** property that is an instance of a **`TaskInfo`** object (see Monitoring TaskInfo Properties).

The following event objects are commonly generated by a console-style client application:

`com.vmware.vim.VmPoweredOnEvent`

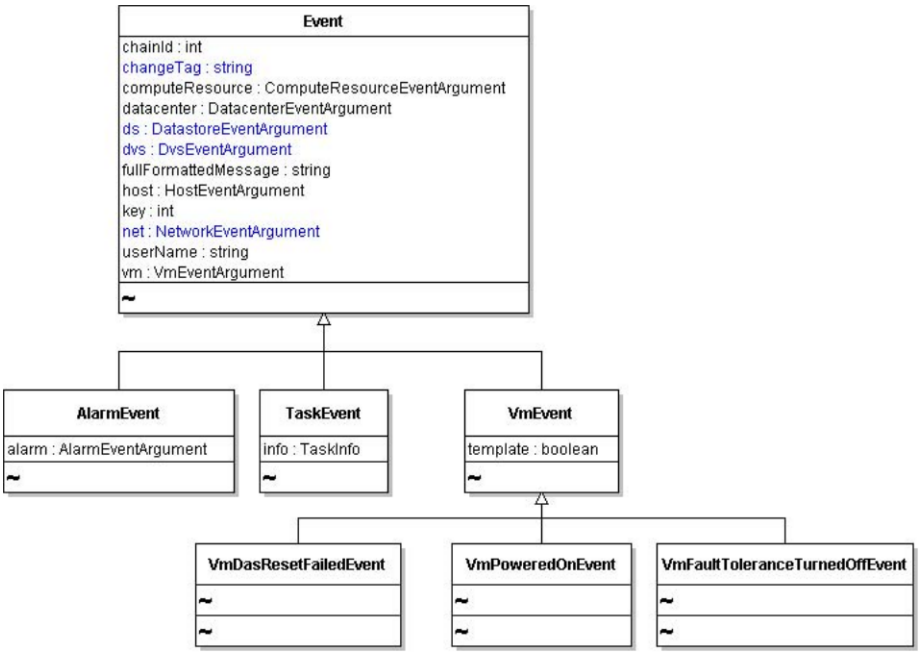
`com.vmware.vim.VmStartingEvent`

`com.vmware.vim.VmReconfiguredEvent`

`com.vmware.vim.VmCreatedEvent`

`com.vmware.vim.VmBeingCreatedEvent`

Figure 59: Event Data Object and Sample Subtypes



Formatting Event Message Content

When displayed at the console, **`Event`** data objects are not formatted and do not provide context information. You can format an **`Event`** message using the predefined string in the **`Event.fullFormattedMessage`** property.

You can also format an **`Event`** message based on contextual information. At runtime, the **`Event`** data object is populated with values that contain information associated with the source of an event, for example, the **`Event`** data object's **`computeResource`**, **`datacenter`**, **`ds`**, **`dvs`**, **`host`**, **`net`**, and **`vm`** properties.

You can use the properties of an **`Event`** object with the information in the **`EventDescriptionEventDetail`** in **`EventManager.description.eventInfo`** to format event messages.

Creating Custom Events

The Web Services API allows you to create custom Event objects that convey information specific to your application. There are two types of custom events, the user log event and the extended event.

Creating User Log Events

The **`EventManager.LogUserEvent`** method allows you to create user log **`Event`** objects. You can associate your custom **`Event`** with any managed entity. User log events are useful for marking actions or status associated with the objects your application deals with.

The following steps show how to create a user log Event.

1. Obtain the managed object reference to the **`EventManager`**.

```
..

ManagedObjectReference _svcRef = new ManagedObjectReference();

ServiceContent _sic = my_conn.retrieveServiceContent(_svcRef);
```

```
ManagedObjectReference eMgrRef = _sic.getEventManager();

...
```

2. Obtain the managed object reference to the entity with which you are associating the **`Event`**.

   For example, suppose you have a reference to a virtual machine (`myVMRef`) and you want to log a message to record the fact that a virus check completed. You want to use `myVMRef` as a parameter to the `LogUserEvent` method in the next step.

3. Call the `LogUserEvent` method, passing in the `EventManager` and the `Event` reference and a string consisting of the Event message for the **`msg`** parameter of the operation.

   ```
   LogUserEvent(eMgrRef, myVMRef, "Completed virus check at 1:05 AM on Sunday December 21.");
   ```

User-defined Event objects display in the vSphere Client among the other events on the system, with the prefix User logged event: followed by the text submitted in your **`msg`** parameter. In other client applications, such as in the console-based Event sample applications, custom events display as `com.vmware.vim.GeneralUserEvent` objects.

Creating Extended Events

The **`EventManager.EventEx`** method allows you to create an event that contains an arbitrary dictionary of key-value pairs. This kind of custom event allows greater flexibility to store application data that is not associated with a managed entity, and is not limited to a single string value. The custom event structure also contains more sophisticated metadata than user log events.

The following steps show pseudocode examples of the operations you need to do in your client code.

1. Obtain the managed object reference to the **`EventManager`**.

   ```
   si = connection.retrieveServiceContent(svc_ref);

   em = si.eventManager;
   ```

2. Choose a severity for the custom event.

   ```
   severity = EventEventSeverity.warning;
   ```

3. Create a local copy of an extended event with metadata.

   ```
   e = vim.event.EventEx(severity,

                         eventTypeId=”com.example.events.Total_System_Backup”,

                         createdTime = si.CurrentTime(),

                         chainId=0,

                         key=0,

                         userName=local_account_name);
   ```

4. Create a set of key-value pairs to store the information you want to associate with the event.

```
arg_list = {};
arg1 = vmodl.KeyAnyValue(key="reason", value="Upcoming governance audit");
arg2 = vmodl.KeyAnyValue(key="datacenter", value="Washington South");
arg3 = vmodl.KeyAnyValue(key="organization", value="IT Cloud Services");
arg_list.append(arg1, arg2, arg3);
```

5. Add the set of key-value pairs to the event object.

   ```
   e.arguments = arg_list;
   ```

6. Use the Event Manager to post the event to the server.

   ```
   em.PostEvent(e);
   ```

Using an EventHistoryCollector

An **`EventHistoryCollector`** lets you gather information about events that the server has generated. You create an **`EventHistoryCollector`** using the **`EventManager.CreateCollectorForEvents`** method.

The following steps show how to create an `EventHistoryCollector`.

1. Identify the type of **`Event`** objects that you want to collect, and create an instance of an **`EventFilterSpec`** data object that specifies your filter criteria. See Creating an EventHistoryCollector Filter.

   The **`EventFilterSpec`** includes an **`eventTypeId`** property, which you use to limit the set of collected event objects to specific types. You can also provide a time range in the **`EventFilterSpec`**, by defining an **`EventFilterSpecByTime`** data object for its **`time`** property. See the *`vSphere API Reference`* for details.

2. Obtain the managed object reference to the **`EventManager`** on your server instance.
3. Submit the **`filter`** and the reference to the server in the **`CreateEventHistoryCollector`** operation. The server returns a reference to an **`EventHistoryCollector`** object.

After you have created the **`HistoryCollector`**, the server appends new objects that meet the filter criteria to the collection as they occur. The system appends the new object to the collection by placing it in the first position of the **`latestPage`** and it removes the oldest object from the collection. The **`latestPage`** property of the **`EventHistoryCollector`** object has a property that consists of the 1000 most recent objects in the collection. Use a **`PropertyCollector`** to obtain the items from the **`latestPage`** property.

A **`HistoryCollector`** exists only for the duration of the session that instantiated it. You invoke the **`DestroyCollector`** operation to explicitly eliminate the collector before the session ends.

Creating an EventHistoryCollector Filter

When you create an **`EventHistoryCollector`**, you can define filters. For example, rather than returning all **`Event`** objects associated with virtual machines, you might create a filter to collect only those **`Event`** objects associated with virtual machines that were started by the backup-administrator between 2:00 and 4:00 a.m. on a specific date.

The **EventFilterSpec** object allows you to specify the collection criteria. Most of the properties are optional and can be submitted as **null** values. The **EventFilterSpec** lets you collect events based on user name, entity type, time, and state of the **Event**.

Managing the HistoryCollector

The **HistoryCollector** managed object provides operations for managing the life-cycle and scrollable view of a collection.

- **DestroyCollector** – A **HistoryCollector** exists only for the current session. Invoke the **DestroyCollector** operation to explicitly destroy the collector before the session ends.
- **ResetCollector** – Adjusts the starting position for the subset of objects from the collector to the object immediately preceding the current **latestPage**.
- **RewindCollector** – Positions the **latestPage** to the oldest item in the array. When a **HistoryCollector** is created, this is the default location.
- **SetCollectorPageSize** – Accepts an integer parameter to set the size of the **latestPage** property of a **HistoryCollector**. The default size of a **HistoryCollector** is an array that consists of at most 1000 objects of the appropriate type (**Task**, **Event**). The array is sorted by creation date and time of the objects.

Using Alarms

The vSphere alarm infrastructure supports automating actions and sending different types of notification in response to certain server conditions. Many **Alarms** exist by default on vCenter Server systems. You can also create alarms yourself. For example, an Alarm can send an alert email message when CPU usage on a specific virtual machine exceeds 99% for more than 30 minutes.

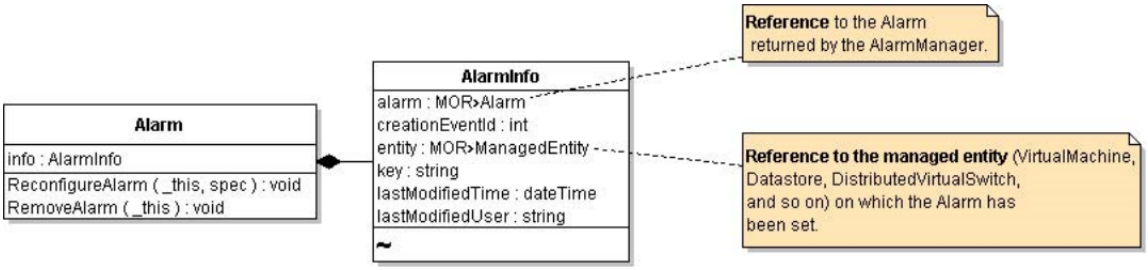
The alarm infrastructure integrates with other server components, such as events and performance counters.

The **AlarmManager** is the service interface for creating, setting, and managing alarms. You create an alarm, specifying trigger conditions and the action to take. When the conditions defined for the **Alarm** occur on the system, the **Action** specified for the alarm starts. The alarm also generates an **Event** that is posted to the **Event** history database. In addition, the action initiated by the **Alarm** might also post a second **Event** to the database, depending on the **Action** type.

Obtaining a List of Alarms

Use the **AlarmManager.GetAlarm** method to obtain an array of references to all **Alarm** managed objects defined for a specific managed entity. When you call the method, you can pass in an optional reference to a managed entity. Without a reference to a managed entity, the **GetAlarm** operation returns all **Alarm** objects for all entities that are visible to the principal associated with the session invoking the operation.

Figure 60: Alarm Managed Object



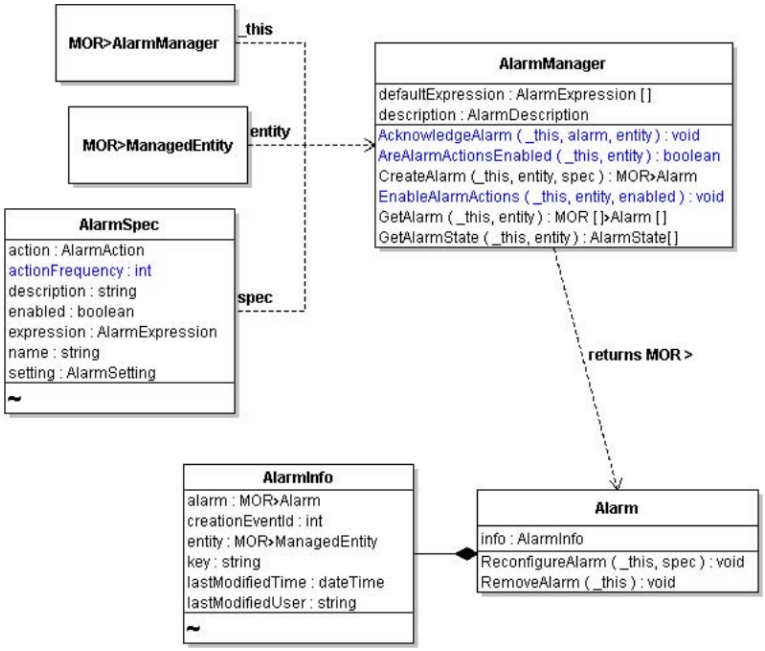
The **Alarm.info** property is an **AlarmInfo** data object. You can obtain information about active **Alarms** by collecting the properties of the **AlarmInfo** data object.

Creating an Alarm

You create an alarm with the **`AlarmManager.CreateAlarm`** method. In the simplest case, you specify the trigger condition in the **`AlarmSpec.expression`** property and the action to perform in the **`AlarmSpec.action`** property. When the expression evaluates to **`true`**, the alarm performs the action.

Figure: CreateAlarm Method Inputs and Outputs shows the **`CreateAlarm`** method.

Figure 61: CreateAlarm Method Inputs and Outputs



The following steps show how to create an alarm.

1. Obtain a managed object reference to the **`AlarmManager`** associated with the vCenter Server.
2. Obtain a managed object reference of the entity on which you want to set the **`Alarm`**.
3. Create an **`AlarmSpec`** data object and specify the alarm details in its properties. See Defining Alarms Using the AlarmSpec Data Object.
4. Call **`AlarmManager.CreateAlarm`**, passing in the references and the **`AlarmSpec`** data object. The system returns a managed object reference to the **`Alarm`** (see Figure: CreateAlarm Method Inputs and Outputs).

The state of an alarm is contained in an **`AlarmState`** data object.

Defining Alarms Using the AlarmSpec Data Object

The **`AlarmSpec`** data object has properties for all aspects of an **`Alarm`**, including its expression and the action to take when the expression evaluates to true. The following properties define the alarm; see the *`API Reference`* for a complete list.

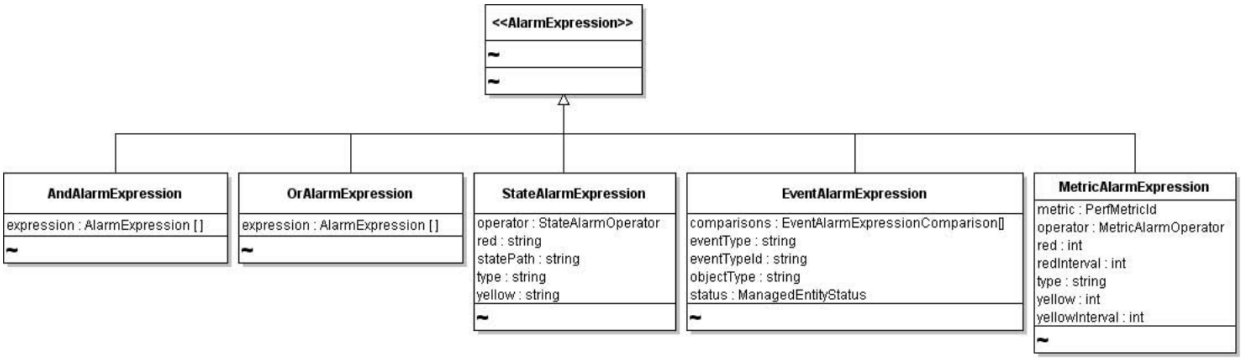
- **`action`** – Action to initiate when the **`Alarm`** becomes active. Specify one of the **`Action`** subtypes. See Specifying Alarm Actions.
- **`actionFrequency`** – Number of seconds that the **`Alarm`** remains in the state required to initiate the specified action.

- **`expression`** – One or more **`AlarmExpression`** data objects combined in a way that evaluates to a true-false expression. See Specifying Alarm Trigger Conditions with AlarmExpression.
- **`setting`** – Tolerance and frequency limits for the **`Alarm`** defined in the **`AlarmSetting`** data object. **`AlarmSetting`** contains two integer properties:
  - **`reportingFrequency`**, which specifies the number of seconds between activation of an alarm. Use 0 to specify that the alarm can activate as frequently as required.
  - **`toleranceRange`**, which specifies the acceptable range (measured in hundredth percentage) above and below the specified value defined in a **`MetricAlarmExpression`**.

Specifying Alarm Trigger Conditions with AlarmExpression

You use the **`AlarmExpression`** data object to specify the conditions under which you want the **`Alarm`** to become active. The **`AlarmExpression`** data object is an abstract type with several subtypes, which allow you to specify thresholds on objects, state of objects, or specify specific events to monitor.

Figure 62: AlarmExpression and Its Subtypes



AlarmExpression Types

By using the appropriate type of **`AlarmExpression`**, you can set alarms for different conditions, states, or events.

AlarmExpression	Description	Example
`StateAlarmExpression`	Specifies thresholds that trigger the alarm.	Triggered by a power state change of a virtual machine or state change of a distributed virtual switch.
`MetricAlarmExpression`	Specifies levels at which the alarm changes state. See Using MetricAlarmExpression.	Triggered when resource utilization metrics exceed a specified limit.
`EventAlarmExpression`	Specifies a type of event as the basis for the alarm.	Triggered by power on or power off events of primary or secondary virtual machines in a fault-tolerant cluster.
`EventAlarmComparison`	Specifies the property of the **`Event`** that should trigger the alarm and the operator to use as the basis for comparison.	
`AndAlarmExpression` **`OrAlarmExpression`**	Combines one or more instances of the **`AndAlarmExpression`** and the **`OrAlarmExpression`** data objects into	

Table continued on next page

Continued from previous page

AlarmExpression	Description	Example
	an expression that evaluates to **`true`** or **`false`**.	

Using MetricAlarmExpression

The **`MetricAlarmExpression`** data object lets you set an alarm to monitor performance metrics. The vSphere Client uses the data object to indicate when hosts or clusters do not have sufficient resources in a DAS or DRS cluster environment. See the *Resource Management Guide.*

You set the **`metric`** property to the **`PerfMetricId`** of a performance metric that you want to monitor on the system. Set the **`red`** or **`yellow`** properties to identify the level at which the metric value moves from green, to yellow, to red. You must define **`red`**, **`yellow`**, or both properties. Use each of these properties with the **`isAbove`** or **`isBelow`** **`MetricAlarmOperator`** enumerations to complete the definition of the threshold.

In conjunction with **`red`** and **`yellow`** properties, you can use the **`redInterval`** or **`yellowInterval`** properties. These properties enable you to set the number of seconds that the performance metric must be in **`red`** or **`yellow`** state before the expression becomes **`true`** and triggers the defined action.

Specifying Alarm Actions

You specify the actions that the system should take by setting the **`action`** property of the **`AlarmSpec`** data object to the **`AlarmAction`** data object defined for the purpose.

The **`AlarmAction`** data object is an abstract type that has two descendent objects.

- The **`AlarmTriggeringAction`** data object has an **`action`** property and a **`transitionSpecs`** property. **`AlarmTriggeringActionTransitionSpec`** allows you to define a starting state and a final state for the **`Alarm`**. You can limit the number of **`Alarm`** objects actually triggered to a single **`Alarm`** by specifying **`false`** for the **`repeats`** property of the **`AlarmTriggeringActionTransitionSpec`**.
- The **`GroupAlarmAction`** data object is an array version of the **`AlarmAction`** base type. You can create a single **`AlarmAction`** instance or an array of **`AlarmAction`** instances to take effect when the conditions specified for your alarm are met on the system.

The system can respond to an alarm in several ways:

- Invoking an operation. To invoke an operation, create a **`MethodAction`** data object.
- Running a Script. To run a script, create an instance of the **`RunScriptAction`** data object that specifies the fully qualified path to the shell script on the vCenter Server.
- Send an email message. To send an email message to a system administrator, use the **`SendEmailAction`** data object.

Figure 63: AlarmAction and Related Objects



For example, you can use the **`MethodAction`** data object type to invoke an operation on the server. The **`MethodAction`** data object contains the following properties:

- **`name`**—Name of the operation that you want to invoke at the scheduled time.
- **`argument`**—Specifies required parameters, if any, as an array of **`MethodArgumentAction`** data objects. Depending on the entity associated with the alarm, the **`MethodAction.argument`** property might not be needed.

Deleting or Disabling an Alarm

An **`Alarm`** remains active until you delete it or disable it. To delete the alarm, obtain a managed object reference to the **`Alarm`** and invoke its **`RemoveAlarm`** operation.

To disable the **`Alarm`**, obtain managed object references to the **`AlarmManager`** and to the entity on which the **`Alarm`** is set. Call **`AlarmManager.EnableAlarmActions`** operation, passing the value **`false`** for the **`enabled`** parameter.

Sample Code for Events and Alarms

The following table lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

Java
`\samples\alarms\MPowerStateAlarm.java`
`\samples\events\EventFormat.java`
`\samples\events\EventHistoryCollectorMonitor.java`
`\samples\events\VMEventHistoryCollectorMonitor.java`

vSphere Performance

VMware vSphere servers use performance counters to track resource use. At runtime, vSphere components generate performance data which the vSphere servers store in performance counters. You can use the **`PerformanceManager`** interface to retrieve the data.

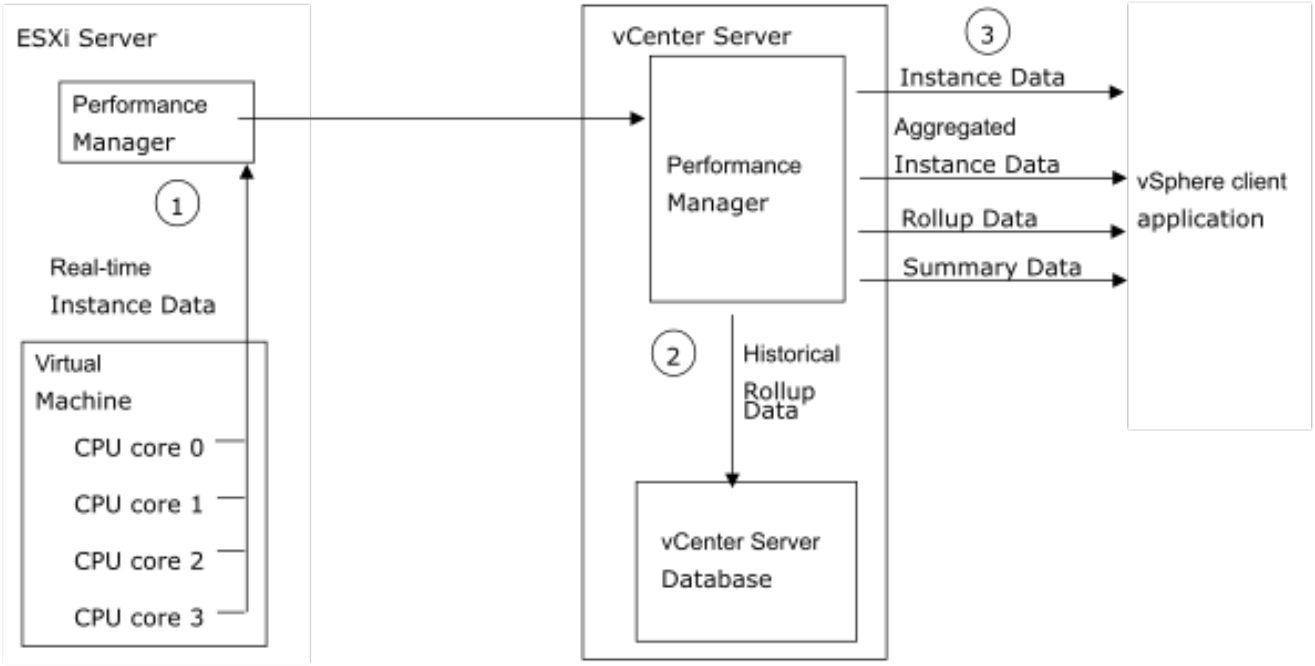
vSphere Performance Data Collection

In a vSphere environment, virtual and physical components generate performance data. To track the use of resources, ESXi Servers perform real-time data collection and vCenter Servers store the data in the vCenter database. vCenter Servers also store a historical rollup of the data according to defined performance intervals.

- Real-time data collection – An ESXi Server collects data for each performance counter every 20 seconds and maintains that data for one hour.
- Historical data rollup – A vCenter Server collects data from all of the hosts that the vCenter Server manages. The **`PerformanceManager`** defines performance intervals that specify time periods for performance data rollup, a methodology for combining data values. The server stores the rolled up performance counter data in the vCenter database.

The following figure represents vSphere performance data collection and retrieval.

Figure 64: vSphere Performance Data Collection and Retrieval



1. ESXi Servers sample performance counter instances every 20 seconds and maintain the real-time instance data for one hour. For example, the figure shows collection of CPU statistics for four CPU cores.
2. The vCenter Server retrieves and stores data from the servers that it manages. The Server produces rollup data according to the settings of the historical intervals.
3. vSphere client applications can retrieve real-time instance data, aggregated instance data, historical rollup data, and summary data.

The following table defines terms that are used to describe vSphere performance management.

Term	Definition
performance providers	Performance providers include managed entities, such as hosts, virtual machines, compute resources, resource pools, datastores, and networks.
performance counter	Unit of statistical data collected on a vSphere server. For example, a vCenter server collects the average CPU utilization for hosts, virtual machines and clusters (the counter **`cpu.usage.average`**).
counter ID	System-generated identifier for a performance counter.
instance	An identifier derived from device configuration names. Examples of counter instances are the name of a virtual Ethernet adapter such as “vmnic0:”, or a number that identifies a CPU core, such as 0, 1, 2, or 3. Performance data is retrieved as specific instances of performance counters.
instance data	Performance data collected at 20-second intervals.
metric ID	Combination of a counter ID and an instance. You use metric IDs – **`PerfMetricId`** objects – when you construct a performance query specification to identify the data to be collected. There are two system-defined instances that you can use to specify aggregate retrieval. See the description of aggregate performance data below. • **`"*"`** – An asterisk directs the vSphere Server to return all instances plus rollup data. This is not supported for some disk-related counters. • **`""`** – A string of length zero directs the vSphere Server to return only aggregated instance data or rollup type data. The vSphere Server returns metric IDs embedded in the data objects that it returns as a response to performance queries.
performance interval	Data object (**`PerfInterval`**) which defines the time interval between collection events, the collection level, and the time period that the data will be stored on the Server. • ESXi Servers define a built-in performance interval that specifies data collection every 20 seconds for each performance counter. ESXi Servers also define a single historical interval (**`PerformanceManager.historicalInterval`**) that defines aggregate performance data. This system-defined performance interval specifies aggregate data collection every 300 seconds for each counter. You cannot modify the performance intervals on an ESXi Server. • vCenter Servers define four performance intervals that determine how collected instance data is aggregated and stored. You can modify the system-defined intervals on a vCenter Server to a limited extent.
collection level	Number between one and four that is assigned to a performance interval (**`PerformanceManager.historicalInterval[].level`**). The interval collection level corresponds to the level specified for individual performance counters (**`PerfCounterInfo.level`**). A vCenter Server uses a performance interval to perform performance data aggregation, using data for the counters with levels that match the performance interval collection level.
rollup type	Methodology for producing a single value from a set of statistical values (**`PerformanceManager.perfCounter[].rollupType`**). Examples of rollup types are average, latest, and summation.
aggregate performance data	A single value that represents a set of instance data values collected for a performance counter. The single value is derived using one of the rollup types.

PerformanceManager Objects and Methods

PerformanceManager provides methods for obtaining statistical data about various aspects of system performance, as generated and maintained by the performance providers. It also defines historical performance intervals and it identifies the set of performance counters that you can use to obtain performance data. The following table shows the **PerformanceManager** properties.

Property	Description
description	Composite object that includes information about the types of counters (**counterType**) and statistics (**statsType**) available on the system.
historicalInterval	Array of system-defined performance intervals (**PerfInterval** data objects). Each object defines the interval between rollup events, the collection level, and the time period that the data is stored on the system. • For an ESXi system, the array contains a single performance interval. You cannot modify the ESXi performance interval. • For vCenter Server systems, the **PerfInterval** objects control how ESXi performance data are rolled up and stored in the database. You can modify some of the **PerfInterval** properties on a vCenter Server.
perfCounter	Array of **PerfCounterInfo** data objects. The array identifies all of the performance counters known to the vCenter Server at the time a client accesses the array. The set of counters may change as ESXi hosts are added or removed from vCenter management. Each **PerfCounterInfo** object contains metadata associated with a performance counter.

The **PerformanceManager** methods allow you to retrieve performance statistics and to retrieve metadata that defines the statistics. The following table classifies the methods and describes their purposes.

Method Type	Method	Purpose
Performance data availability	QueryAvailablePerfMetric	Returns **PerfMetricId** objects which identify the counter data available on the specified entity. For example, a virtual machine provides the memory counter **granted**, which indicates the amount of physical memory that is mapped for the virtual machine. The **PerfMetricId** object for the **mem.granted.average** counter specifies the system-defined counter ID. Since this is a memory counter, the **PerfMetricId.instance** property is empty.
Performance data retrieval	QueryPerf	Returns statistics for a specific list of managed entities that provide performance data.
	QueryPerfComposite	Returns statistics for a host and its virtual machines. This method accepts the **refreshRate** for current statistics or the **intervalId** of one of the historical intervals as a parameter. Supported for the **HostSystem** managed entity only.
Performance counter metadata retrieval	QueryPerfCounter	Returns **PerfCounterInfo** data objects for the specified list of counter IDs.
	QueryPerfCounterByLevel	Returns **PerfCounterInfo** data objects for the specified collection level.

Table continued on next page

Continued from previous page

Method Type	Method	Purpose
Performance provider information	QueryPerfProviderSummary	Returns the **PerfProviderSummary** data object for the specified managed object.
Collection parameters	ResetCounterLevelMapping	Restores a set of performance counters to their default collection levels.
	UpdateCounterLevelMapping	Changes the collection level for a set of performance counters.
	UpdatePerfInterval	Modifies the system-defined performance intervals.

Retrieving vSphere Performance Data

To retrieve collected data, your client application creates a query specification and passes the specification to a performance query method. The query specification is composed of one or more **PerfQuerySpec** objects. Each object identifies the following:

- Performance provider – managed entity for which the Server will return performance data (**PerfQuerySpec.entity**).
- Performance counters – **PerfMetricId** objects that identify performance counter instances (**PerfQuerySpec.metricId**).
- Performance interval – the sampling period that defines the data rollup (**PerfQuerySpec.intervalId**).
- Amount of data to be returned – start and end times (**PerfQuerySpec.startTime**, **PerfQuerySpec.endTime**) and maximum number of values (**PerfQuerySpec.maxSample**) to limit the amount of data to be returned.
- Output data format (**PerfQuerySpec.format**) – one of two kinds:
  - Normal output returned as values contained in data objects.
  - Formatted output returned as strings containing comma-separated values.

The combination of the **entity** and **metricID** properties determine the set of counters for which the server will return performance data. The combination of the **interval**, **startTime**, **endTime** properties produce instance, aggregated instance, rollup, or summarized data. The following table summarizes the different classifications of performance data that you can retrieve from a vCenter Server.

Performance Data	Description
Instance	ESXi Servers sample performance data every 20 seconds. 20-second interval data is called instance data or real-time data. To retrieve instance data, specify a value of 20 seconds for the **PerfQuerySpec.intervalId** property.
Aggregated Instance	A vSphere client can retrieve aggregated instance data. To obtain aggregated instance data, specify the following **PerfQuerySpec** properties. • **intervalId** – Specify 20 seconds to indicate instance data. • **metricId[].instance** – specify a zero-length string ("") for aggregated instance data.
Rollup	The vCenter Server uses the historical intervals to rollup performance data from the servers that it manages. To retrieve historical performance data, specify the following **PerfQuerySpec** properties. • **intervalId** – Specify a value that corresponds to one of the historical intervals (**PerformanceManager.historicalInterval[].samplingPeriod**) . • **startTime**/**endTime** – If specified, use time values that are not within the last 30 minutes of the current time. If you do not specify a starting time, the Server will return values starting with

Table continued on next page

Continued from previous page

Performance Data	Description
	the earliest data. If you do not specify an end time, the Server will return values that include the latest data.
Summary	When you call the **QueryPerf** method and specify a performance interval (**PerfQuerySpec.intervalId**) that does not match one of the historical intervals (**PerformanceManager.historicalInterval[].samplingPeriod**), the Server will attempt to summarize the stored data for the specified interval. In this case, the Server may return values that are different from the values that were stored for the historical intervals.

Performance Counter Example (QueryPerf)

The following code fragments are part of an example that uses the **PerformanceManager.QueryPerf** method to obtain performance statistics for a virtual machine.

The example code in this section does not include server connection code and it does not show the code for obtaining the managed object reference for the virtual machine. See Client Applications for the Web Services API for an example of server connection code.

This example retrieves the following statistics:

- **disk.provisioned.LATEST** – virtual machine storage capacity.
- **mem.granted.AVERAGE** – amount of physical memory mapped for the virtual machine.
- **power.power.AVERAGE** – current power usage.

The example creates a query specification (**PerfQuerySpec**) to identify the data to be retrieved, calls the **QueryPerf** method, and prints out the retrieved performance data and corresponding performance counter metadata. The following sections describe the basic steps involved in retrieving performance statistics.

- Map the performance counters – Mapping Performance Counters (Counter Ids and Metadata).
- Create a performance query specification and call the **QueryPerf** method – Retrieving Statistics.
- Process the returned data – Handling Returned Performance Data.

Mapping Performance Counters (Counter Ids and Metadata)

Performance counters are represented by string names, for example **disk.provisioned.LATEST** or **mem.granted.AVERAGE**. A vSphere server tracks performance counters by using system-generated counter IDs. When you create a performance query, you use counter IDs to specify the statistics to be retrieved, so it is useful to map the names to IDs.

The example must specify counter IDs in the calls to `QueryPerf`, and it will use performance counter metadata when it prints information about the returned data. To obtain performance counter IDs and the corresponding performance counter metadata, the example creates two hash maps. This example maps the entire set of performance counters to support retrieval of any counter.

HashMap Declarations

The following code fragment declares two hash maps.

- **countersIdMap** – Uses full counter names to index performance counter IDs. A full counter name is the combination of counter group, name, and rollup type. The example uses this map to obtain counter IDs when it builds the performance query specification.

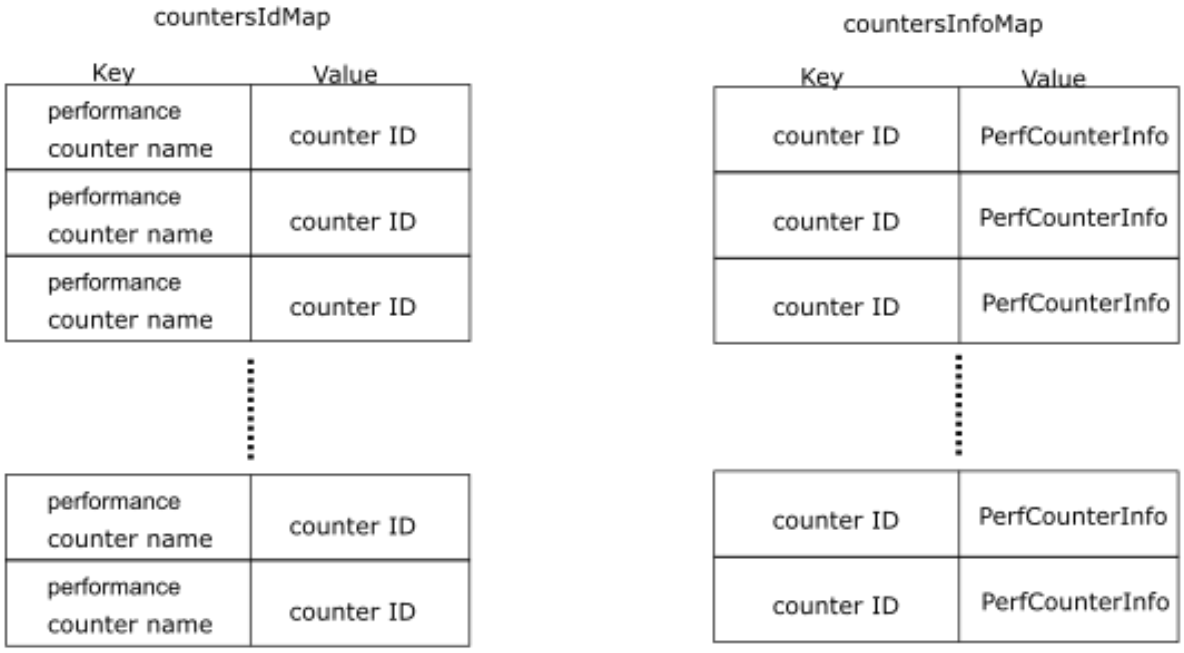
- **countersInfoMap** – Uses performance counter IDs to index **PerformanceCounterInfo** data objects. The example uses this map to obtain metadata when it prints the returned performance data.

```
/*
 * Map of counter IDs indexed by counter name.
 * The full counter name is the hash key - group.name.ROLLUP-TYPE.
 */
private static HashMap<String, Integer> countersIdMap = new HashMap<String, Integer>();


/*
 * Map of performance counter data (PerfCounterInfo) indexed by counter ID
 * (PerfCounterInfo.key property).
 */
private static HashMap<Integer, PerfCounterInfo> countersInfoMap =
    new HashMap<Integer, PerfCounterInfo>();
```

The following figure shows a representation of the hash maps.

Figure 65: Performance Counter Hash Maps



Creating the Map

The example uses the Property Collector to retrieve the array of performance counters (**PerfCounterInfo**) known to the vCenter Server (**PerformanceManager.perfCounter[]**). It then uses the data to create the maps. The code fragment uses the variable **apiMethods**, which is a **VimPortType** object that provides access to the vSphere API

methods. For information about the **VimPortType** object, see Overview of a Java Sample Application for the Web Services SDK.

The following code fragment performs these steps:

1. Create an **ObjectSpec** to define the property collector context. This example specifies the Performance Manager.
2. Create a **PropertySpec** to identify the property to be retrieved. This example retrieves the **perfCounter** property, which is an array of **PerfCounterInfo** objects.
3. Create a **PropertyFilterSpec** for the call to the `PropertyCollector`. The **PropertyFilterSpec** creates the association between the **ObjectSpec** and `PropertySpec` for the operation.
4. Call the **PropertyCollector.RetrievePropertiesEx** method. This method blocks until the server returns the requested property data.
5. Cast the returned **xsd:anyType** value into the array of **PerfCounterInfo** objects.
6. Cycle through the returned array and load the maps. The counter-name to counter-ID map uses a fully qualified counter name. The qualified name is a path consisting of counter group, counter name, and rollup type – *group.counter.ROLLUP-TYPE* The rollup type must be coded in uppercase letters. Examples of qualified names are **disk.provisioned.LATEST** and **mem.granted.AVERAGE**.

```
/*

 * Create an object spec to define the context to retrieve the PerformanceManager
property.

 */

ObjectSpec oSpec = new ObjectSpec();

oSpec.setObj(performanceMgrRef);


/*

 * Specify the property for retrieval

 * (PerformanceManager.perfCounter is the list of counters the vCenter Server is aware
of.)

 */

PropertySpec pSpec = new PropertySpec();

pSpec.setType("PerformanceManager");

pSpec.getPathSet().add("perfCounter");


/*

 * Create a PropertyFilterSpec and add the object and property specs to it.

 */

PropertyFilterSpec fSpec = new PropertyFilterSpec();

fSpec.getObjectSet().add(oSpec);

fSpec.getPropSet().add(pSpec);
```

```
/*
 * Create a list for the filter and add the spec to it.
 */
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);

/*
 * Get the performance counters from the server.
 */
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = apiMethods.retrievePropertiesEx(pCollectorRef,fSpecList,ro);

/*
 * Turn the retrieved results into an array of PerfCounterInfo.
 */
List<PerfCounterInfo> perfCounters = new ArrayList<PerfCounterInfo>();
if (props != null) {
   for (ObjectContent oc : props.getObjects()) {
      List<DynamicProperty> dps = oc.getPropSet();
      if (dps != null) {
         for (DynamicProperty dp : dps) {
            /*

/*
 * Cycle through the PerfCounterInfo objects and load the maps.
 */
             *  DynamicProperty.val is an xsd:anyType value to be cast
             *  to an ArrayOfPerfCounterInfo and assigned to a List<PerfCounterInfo>.
             */
            perfCounters = ((ArrayOfPerfCounterInfo)dp.getVal()).getPerfCounterInfo();
         }
```

```
      }
   }
}
for(PerfCounterInfo perfCounter : perfCounters) {
   Integer counterId = new Integer(perfCounter.getKey());

   /*
    * This map uses the counter ID to index performance counter metadata.
    */
   countersInfoMap.put(counterId, perfCounter);

   /*
    * Obtain the name components and construct the full counter name,
    * for example - power.power.AVERAGE.
    * This map uses the full counter name to index counter IDs.
    */
   String counterGroup = perfCounter.getGroupInfo().getKey();
   String counterName = perfCounter.getNameInfo().getKey();
   String counterRollupType = perfCounter.getRollupType().toString();
   String fullCounterName = counterGroup + "." + counterName + "." + counterRollupType;

   /*
    * Store the counter ID in a map indexed by the full counter name.
    */
   countersIdMap.put(fullCounterName, counterId);

}
```

Retrieving Statistics

The following code fragment calls the **`QueryPerf`** method to retrieve statistics. It performs these tasks:

1. Create a list of qualified performance counter names for retrieval. The name is a path consisting of *`group-name.counter-name.ROLLUP-TYPE`*, for example **`mem.granted.AVERAGE`**. The rollup type must be coded in uppercase letters to match the character case of the rollup type in the performance counter metadata

(**PerfCounterInfo.rollupType**). See the *vSphere API Reference* for tables of available counters. The *vSphere API Reference* page for the **PerformanceManager** managed object contains links to the tables.

2. Create a list of **PerfMetricId** objects, one for each counter to be retrieved. The metric ID is a combination of the counter ID and the instance. To fill in the **PerfMetricId** properties, the example does the following:
   - Use the **countersIdMap** to translate a full counter name into a counter ID.
   - Specify an asterisk (*) for the **PerfMetricId.instance** property. The asterisk is the system-defined instance specification for combined instance and rollup retrieval.
3. Build a query specification for the method call. This query specifies the following:
   - Virtual machine for which performance data is being retrieved (**entityMor**);
   - Interval ID of 300 to collect 5-minute rollup data.
   - Comma-separated value (CSV) format for the retrieved data.
4. Call the **QueryPerf** method.

```
/*
 * Use <group>.<name>.<ROLLUP-TYPE> path specification to identify counters.
 */
String[] counterNames = new String[] {"disk.provisioned.LATEST",
                                      "mem.granted.AVERAGE",
                                      "power.power.AVERAGE"};


/*
 * Create the list of PerfMetricIds, one for each counter.
 */
List<PerfMetricId> perfMetricIds = new ArrayList<PerfMetricId>();
for(int i = 0; i < counterNames.length; i++) {
   /*
    * Create the PerfMetricId object for the counterName.
    * Use an asterisk to select all metrics associated with counterId (instances and
rollup).
    */
   PerfMetricId metricId = new PerfMetricId();
   /* Get the ID for this counter. */
   metricId.setCounterId(countersIdMap.get(counterNames[i]));
   metricId.setInstance("*");
   perfMetricIds.add(metricId);
}
```

```
/*
 * Create the query specification for queryPerf().
 * Specify 5 minute rollup interval and CSV output format.
 */
int intervalId = 300;
PerfQuerySpec querySpecification = new PerfQuerySpec();
querySpecification.setEntity(
querySpecification.setIntervalId(intervalId);
querySpecification.setFormat("csv");
querySpecification.getMetricId().addAll(perfMetricIds);


List<PerfQuerySpec> pqsList = new ArrayList<PerfQuerySpec>();
pqsList.add(querySpecification);


/*
 * Call queryPerf()
 *
 * QueryPerf() returns the statistics specified by the provided
 * PerfQuerySpec objects. When specified statistics are unavailable -
 * for example, when the counter doesn't exist on the target
 * ManagedEntity - QueryPerf() returns null for that counter.
 */
List<PerfEntityMetricBase> retrievedStats = apiMethods.queryPerf(performanceMgrRef,
pqsList);
```

Performance Data Returned by a vSphere Server

The query methods return sampling information and performance data. The sampling information indicates the collection interval in seconds and the time that the data was collected. When you call performance query methods, you pass in query specifications (**`PerfQuerySpec`**) to identify the performance data to be retrieved. To indicate the format of the output data, specify either `"normal"` or `"csv"` for the **`PerfQuerySpec.format`** property.

The query methods return **`PerfEntityMetricBase`** objects which you must cast into the appropriate type that corresponds to the **`PerfQuerySpec.format`** value specified in the call to the method.

- The **`QueryPerf`** method returns a list of **`PerfEntityMetricBase`** objects.

- The **QueryPerfComposite** method returns a **PerfCompositeMetric** object, which contains **PerfEntityMetricBase** objects.

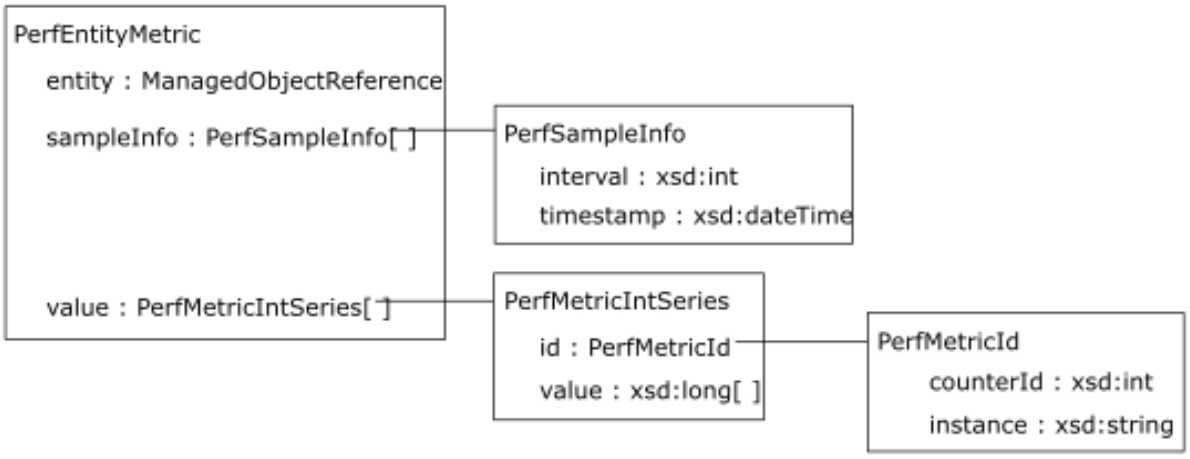
Normal Output Format

When you specify "normal" format, you must cast the returned **PerfEntityMetricBase** objects into **PerfEntityMetric** objects. Each **PerfEntityMetric** object contains the following properties:

- **entity** – Reference to the performance provider.
- **sampleInfo** – Array of sample information (**PerfSampleInfo** data objects), encoded as **xsd:int** and **xsd:dateTime** values.
- **value** – Array of data values (**PerfMetricIntSeries** data objects). Each object in the array contains the following properties:
  - **id** – Performance metric ID that identifies the counter instance.
  - **value** – Array of integers that corresponds to the array of sample information (**PerfEntityMetric.sampleInfo**).

The following figure shows a representation of the data object hierarchy returned by the query methods for normal format.

Figure 66: PerfEntityMetric Object Hierarchy



CSV Output Format

When you specify "csv" format, you must cast the returned **PerfEntityMetricBase** objects into **PerfEntityMetricCSV** objects. Both the sampling information and the collected data are encoded as comma-separated values suitable for display in tabular format.

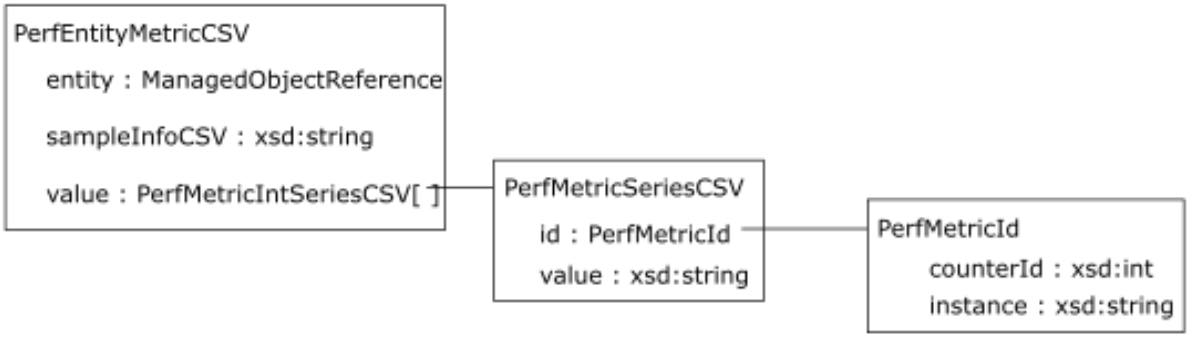
The **PerfEntityMetricCSV** object contains the following properties:

- **entity** – Reference to the performance provider.
- **sampleInfoCSV** – String containing a set of interval and date-time values. The property contains string representations of **PerfSampleInfo xsd:int** and **xsd:dateTime** values. The string values are encoded in the following CSV format:
  interval1, *date1*, *interval2*, *date2*
- value – Array of data values (**PerfMetricSeriesCSV** data objects). Each object in the array contains the following properties:
  - id – Performance metric ID that identifies the counter instance.

  - value – Set of sample values in CSV format, corresponding to the list of sample information (**`PerfEntityMetricCSV.sampleInfoCSV`**).

The following figure shows a representation of the data object hierarchy returned by the query methods for CSV format.

Figure 67: PerfEntityMetricCSV Object Hierarchy



Handling Returned Performance Data

The following code fragment prints out the returned performance data. This example uses CSV formatted data. The code fragment performs these tasks:

- Loop through the list of **`PerfEntityMetricBase`** objects returned by the **`QueryPerf`** method (**`retrievedStats`**).
  - Cast the **`PerfEntityMetricBase`** object to a **`PerfEntityMetricCSV`** object to handle the CSV output specified in the **`PerfQuerySpec`**.
  - Retrieve the sampled values.
  - Retrieve the interval information (**`csvTimeInfoAboutStats`**). The **`sampleInfoCSV`** string (**`PerfEntityMetricCSV.sampleInfoCSV`**) is **`PerfSampleInfo`** data formatted as interval,time pairs separated by commas – **`interval-1,time-1,interval-2,time-2`**. The list of pairs embedded in the string corresponds to the list of sampled values (**`PerfEntityMetricCSV.value[]`**).
  - Print the time and interval information.
  - Loop through the sampled values (**`metricsValues`**).
    - Use the counter metadata to print out identifying information about the counter along with the returned sampled value for the counter.
    - Use the **`countersInfoMap`** to translate the counter ID returned in the **`PerfMetricSeriesCSV`** object into the corresponding **`PerfCounterInfo`** object.

```
/*

 * Cycle through the PerfEntityMetricBase objects. Each object contains

 * a set of statistics for a single ManagedEntity.

 */

for(PerfEntityMetricBase singleEntityPerfStats : retrievedStats) {


  /*

   * Cast the base type (PerfEntityMetricBase) to the csv-specific sub-class.

   */

  PerfEntityMetricCSV entityStatsCsv = (PerfEntityMetricCSV)singleEntityPerfStats;
```

```
/* Retrieve the list of sampled values. */
List<PerfMetricSeriesCSV> metricsValues = entityStatsCsv.getValue();

if(metricsValues.isEmpty()) {
  System.out.println("No stats retrieved. " +
                     "Check whether the virtual machine is powered on.");
        throw new Exception();
}

/*
 * Retrieve time interval information (PerfEntityMetricCSV.sampleInfoCSV).
 */
String csvTimeInfoAboutStats = entityStatsCsv.getSampleInfoCSV();

/* Print the time and interval information. */
System.out.println("Collection: interval (seconds),time (yyyy-mm-ddThh:mm:ssZ)");
System.out.println(csvTimeInfoAboutStats);

/*
 * Cycle through the PerfMetricSeriesCSV objects. Each object contains
 * statistics for a single counter on the ManagedEntity.
 */
for(PerfMetricSeriesCSV csv : metricsValues) {
  /*
   * Use the counterId to obtain the associated PerfCounterInfo object
   */
  PerfCounterInfo pci = countersInfoMap.get(csv.getId().getCounterId());

  /* Print out the metadata for the counter. */
  System.out.println("---------------------------------------");
  System.out.println(pci.getGroupInfo().getKey() + "."
```

```
                        + pci.getNameInfo().getKey() + "."
                        + pci.getRollupType() + " - "
                        + pci.getUnitInfo().getKey());
    System.out.println("Instance: "+csv.getId().getInstance());
    System.out.println("Values: " + csv.getValue());
  }
}
```

Large-Scale Performance Data Retrieval

The example described in the previous sections shows how to retrieve performance data for a single entity. When you design your application to retrieve performance data on a large scale, take the following information into consideration for more efficient processing.

- Use CSV formatted output. CSV format provides a more compact representation of the output data which can save on meta-data overhead.
- Create query specifications to reference a set of vSphere entities.
  - Using one **`QueryPerf`** method call per entity is not efficient.
  - Using a single call to **`QueryPerf`** to retrieve all of the performance data is not efficient.
  - As a general rule, specify between 10 and 50 entities in a single call to the **`QueryPerf`** method. This is a general recommendation because your system configuration may impose different constraints.
- Do not retrieve statistics more frequently than they are refreshed. For example, when you retrieve 20-second interval data, the data will not change until the next 20-second data collection event.
- Use **`QueryAvailablePerfMetric`** only when you intend to send a query for a specific counter using a specific performance interval. The method will return **`PerfMetricId`** objects that you can use for the query.
  In all other cases, create the **`PerfMetricId`** objects for the query.
  - For the **`counterId`** property, use the counter IDs from the **`PerformanceManager`** counter list (`PerformanceManager.perfCounter[].key`).
  - For the instance property, specify an asterisk (“*”) to retrieve instance and aggregate data or a zero-length string (““) to retrieve aggregate data only.

Using the QueryPerf Method as a Raw Data Feed

The **`QueryPerf`** method can operate as a raw data feed that bypasses the vCenter database and instead retrieves performance data from an ESXi host. You can use a raw data feed to obtain real-time instance data associated with 20-second interval collection and aggregate data associated with the 5-minute intervals.

You can use a raw data feed on vCenter Server 2.5 and later.

Performance Interval	Description
20-second	ESXi servers collect data for each performance counter every 20 seconds and maintain that data for an hour. When you specify a 20-second interval in the query specification for the **`QueryPerf`** method (**`PerfQuerySpec.intervalId`**), the method operates as a raw data feed. The Server ignores the historical interval collection levels and retrieves data for all of the requested counters from the ESXi servers. When you send a query for 20-second instance

Table continued on next page

Continued from previous page

Performance Interval	Description
	data, the server returns the most recent data collected for the 20-second interval. The server does not perform additional, unscheduled data collection to satisfy the query.
5-minute	ESXi servers aggregate performance data according to the system-defined performance interval which specifies data collection every 300 seconds. To use a raw data feed for this data, specify the following **`PerfQuerySpec`** properties in the call to the **`QueryPerf`** method. • **`intervalId`** – Specify 300 seconds to match the system-defined performance interval. • **`startTime`**/**`endTime`** – Specify time values within the last 30 minutes of the current time. The **`QueryPerf`** method checks the performance interval collection level on the vCenter Server. The method returns aggregated statistics for performance counters that specify a collection level (**`PerfCounterInfo.level`**) at or below the vCenter Server performance interval for the 300 second sampling period (**`PerfInterval.level`**). For example, if the vCenter Server performance interval is set to level one, and your query specification requests only performance counters that specify level four, the **`QueryPerf`** method will not return any data.

Comparison of Query Methods

The following table presents a comparison of performance query methods.

Method	Notes
QueryPerf	• Specify an array of **`PerfQuerySpec`** objects. • An unset **`PerfQuerySpec.metricId`** property produces results for all counters defined for **`PerfQuerySpec.entity`**. • **`PerfQuerySpec.maxSample`** is ignored for historical statistics. You can use this method to retrieve historical statistics; you can also use it as a raw data feed. For information about retrieving the raw data collected on ESXi servers, see Using the QueryPerf Method as a Raw Data Feed.
QueryPerfComposite	• Method works only at the host level. You can use a single call to the **`QueryPerfComposite`** method to retrieve performance data for a host and its virtual machines. • Specify a single **`PerfQuerySpec`** object. • You must specify a list of performance metrics to identify the data to be retrieved (**`PerfQuerySpec.metricId`**). • You cannot specify **`PerfQuerySpec.maxSample`**. This method is designed for efficient client-server communications. **`QueryPerfComposite`** usually generates less network traffic than **`QueryPerf`** because it returns a large-grained object, a **`PerfCompositeMetric`** data object, that contains all the data.

Retrieving Summary Performance Data

You can obtain near real-time summary information about performance or utilization without using the `PerformanceManager` methods. vSphere servers maintain "quick stats" data objects for hosts (**`HostListSummaryQuickStats`**), virtual machines (**`VirtualMachineQuickStats`**), and resource pools (**`ResourcePoolQuickStats`**). For more information about these objects, see the *`vSphere API Reference`*.

Performance Counter Metadata

Performance counters are organized by groups of system resources. Examples of performance counter groups are memory, CPU, and disk. The counter groups and specific counters used on any vSphere server depend on the server configuration. The *vSphere API Reference* contains a table for each counter group. The table includes the counter name, type of statistics being collected, unit of measurement, level, and so on. The *vSphere API Reference* page for the **PerformanceManager** managed object contains links to the tables.

PerfCounterInfo

The **PerformanceManager.perfCounter** property is an array of **PerfCounterInfo** data objects. Each object provides metadata for the collected data. A **PerfCounterInfo** object has a unique key, the counter ID. The actual performance data collected at runtime are identified by this counter ID. The following table lists the **PerfCounterInfo** properties.

Property	Description
groupInfo	Name of the resource group to which this counter belongs, such as disk, cpu, or memory.
key	Unique integer that identifies the counter. Also called the counter ID. The value is unique and it is not static—it might, for example, change between system reboots. The counter key on an ESXi system might not be the same as the counter key for the same counter on the vCenter Server system managing the ESXi system. However, the system maps the keys from ESXi to vCenter Server systems automatically.
level	Number from 1 to 4 that identifies the level at which data values for this counter are aggregated.
nameInfo	Descriptive name for the counter. The name component of a fully qualified counter name, for example `"granted"` is the **nameInfo** property for the **mem.granted.AVERAGE** counter.
rollupType	Indicates how multiple samples of a counter are transformed into a single statistical value. Examples of rollup types are average, summation, and minimum. No conversion of values occurs for counters that specify absolute values, such as the total number of seconds that the system has been running continuously since startup. The **PerfSummaryType** is an enumeration containing valid constants for this property.
statsType	Type of statistical data that the value represents over the course of the interval, such as an average, a rate, the minimum value, and so on. The **PerfStatsType** is an enumeration containing valid constants for this property.
unitInfo	Unit of measure, such as megahertz, kilobytes, kilobytes per second, and so on. The **ElementDescription**'s **key** property is populated using one of the constants available in the **PerformanceManagerUnit** enumeration.

Performance Intervals

The **PerformanceManager** defines performance intervals which specify the period of time between collection events, how much data will be collected, and how long the collected data will be saved.

- An ESXi server has a built-in performance interval that produces discrete data values from counter instances sampled every 20 seconds. The server will maintain this instance data for one hour.
- Additional data collection is specified by historical performance intervals which produce data aggregated from counter instances according to the individual intervals.

The **PerformanceManager.historicalInterval** property is an array of **PerfInterval** objects. The following table lists the **PerfInterval** properties.

Property	Description
samplingPeriod	Number of seconds for the interval. You can modify this property on a vCenter Server only.
length	Period of time for which the server will save the data that it collects. You can modify this property on a vCenter Server only.
level	Level at which the Server collects data. The interval level corresponds to the performance counter level (**`PerfCounterInfo.level`**). The Server will collect data for all counters with levels that match **`PerfInterval.level`**, and for all counters with levels lower than **`PerfInterval.level`**. You can modify this property on a vCenter Server only.
enable	Enable/disable performance data collection. You can modify this property on a vCenter Server only.
key	Unique identifier for the interval. You cannot modify this property.
name	Label for the historical interval; one of the following strings: • `"Past Day"` • `"Past Week"` • `"Past Month"` • `"Past Year"` The **`PerformanceManager`** uses the **`samplingPeriod`**, **`level`**, and **`length`** properties to determine its collection behavior. It does not interpret the name string. You cannot modify this property.

ESXi Server Performance Intervals

An ESXi server collects performance data for each performance counter every 20 seconds. The **`PerformanceManager.historicalInterval`** array for an ESXi Server contains a single, readonly **`PerfInterval`** object that specifies rollup data collection every 5 minutes. You cannot retrieve 5-minute rollup data from an ESXi Server directly. You can use a vCenter Server connection to obtain 5-minute rollup data for an ESXi Server. The following table shows the historical interval property values on an ESXi server. You cannot modify this performance interval.

Property	Value	Description
`key`	**`1`**	Numeric identifier for the **`PerfInterval`**.
`name`	**`PastDay`**	Name of the **`PerfInterval`**.
`samplingPeriod`	**`300`**	Time interval between data sampling events.
`length`	**`129600`**	Number of seconds that statistics associated with the interval are kept by the vCenter Server.
`enabled`	**`true`**	This **`PerfInterval`** is enabled on the system.
`level`	**`null`**	Statistics collection level. For an ESXi system, this property is **`null`**. The **`PerfInterval`** object on an ESXi system defines the baseline interval.

vCenter Server Performance Intervals

A vCenter Server system aggregates performance data from all ESXi systems that it manages. The amount of data aggregated depends on the level setting configured for the vCenter Server. The level settings are reflected in the **`PerformanceManager.historicalInterval`** property for the vCenter Server system. **`historicalInterval`** is an array of **`PerfInterval`** data objects that define four different level settings, 1 through 4.

The following table lists the default values for the performance intervals on a vCenter Server system.

Key	Name	Sampling Period	Length	Enabled	Level
1	Past Day	300	86400	TRUE	1
2	Past Week	1800	604800	TRUE	1
3	Past Month	7200	2592000	TRUE	1
4	Past Year	86400	31536000	TRUE	1

By default, the collection level is set to 1 for each of the four intervals. Using the default level, a vCenter Server will collect data for all performance counters that specify collection level 1. Using the default length value, a vCenter Server will save collection data for the following time periods:

- 5-minute samples for the past day
- 30-minute samples for the past week
- 2-hour samples for the past month
- 1-day samples for the past year

Data older than a year is purged from the vCenter Server database.

vSphere Performance and Data Storage

The following sections provide information about modifying the operation of the **`PerformanceManager`** and vSphere Server performance data collection and storage.

- Modifying Historical Intervals
- Modifying Performance Counter Collection Levels

Modifying Historical Intervals

Changes to a vCenter performance interval are global and apply to all entities in the system. VMware recommends that you do not modify the historical intervals. The **`PerfInterval`** data objects in the **`PerformanceManager.historicalInterval`** array are related. Modifications to a performance interval affects the entire system and may cause problems.

If you must modify a performance interval, use the `PerformanceManager.UpdatePerfInterval` method and follow these guidelines.

- Performance data retention time (`PerfInterval.length`) must be a multiple of the collection interval (`PerfInterval.samplingPeriod`).
- Performance data retention length must increase in each interval compared to its predecessor. The `PerfInterval.length` value for each successive performance interval must be greater than the length property for the previous interval in the historical interval array.
- You cannot modify the value of the `PerfInterval.samplingPeriod` property on ESXi systems.

Modifying Performance Counter Collection Levels

The **`PerformanceManager`** provides the **`UpdateCounterLevelMapping`** method to change the collection level for individual performance counters (**`PerfCounterInfo.level`**). Consider carefully the performance and storage consequences of using the **`UpdateCounterLevelMapping`** method. If you use this method, you may cause a significant increase in data collection and storage, along with a corresponding decrease in performance. vCenter Server performance and database storage requirements depend on the collection levels defined for the performance intervals (**`PerformanceManager.historicalInterval`**) and the collection levels specified for individual performance counters (`PerfCounterInfo.level`).

Performance Counter Data Collection

vSphere defines four levels of data collection for performance counters. Each performance counter specifies a level for collection. The historical performance intervals (**`PerformanceManger.historicalInterval`**) define the sampling period and length for a particular collection level.

The amount of data collected for a performance counter depends on the performance interval and on the type of entity for which the counter is defined. For example, a datastore counter such as `datastoreIops` (the aggregate number of IO operations on the datastore) will generate a data set that corresponds to the number of datastores on a host. If a vCenter Server manages a large number of hosts with a large number of datastores, the Server will collect a large amount of data.

There are other counters for which the vCenter Server collects a relatively smaller amount of data. For example, memory counters are collected as a single counter per virtual machine and a single counter per host.

Performance Counter Data Storage

The performance interval collection level (**`PerfInterval.level`**) defines the set of counters for which the vCenter Server stores performance data. The Server will store data for counters at the specified level and for counters at all lower levels.

By default, all the performance intervals specify collection level one. Using these defaults, the vCenter Server stores performance counter data in the vCenter database for all counters that specify collection level one. It does not store data for counters that specify collection levels two through four.

Performance Manager Method Interaction

You can use the **`UpdateCounterLevelMapping`** method to change the collection level for individual counters. You can also use the **`UpdatePerfLevel`** method to change the collection level for the system-defined performance intervals. These methods can cause a significant increase in the amount of data collected and stored in the vCenter database.

- By default the system-defined performance intervals use collection level one, storing data for all counters that specify collection level one. If you use the **`UpdateCounterLevelMapping`** method to change the collection level of performance counters to level one, you will increase the amount of stored performance data.
- If you use the **`UpdatePerfLevel`** method to increase the collection level for the system-defined performance intervals, you will increase the amount of stored performance data.

To restore counter levels to default settings use the **`ResetCounterLevelMapping`** method.

vSphere Client Management of Performance Statistics

The vSphere Client displays the Performance Manager historical interval collection levels in the vCenter management statistics display. The vSphere Client also displays an estimate of the amount of storage that is required for data collection at the displayed levels. If individual counter levels are modified through the vSphere API (the **`UpdateCounterLevelMapping`** method), the vSphere Client will show a modified estimate. However, the vSphere Client cannot detect that the method has been called and it cannot display the current levels for individual counters. If you see a significantly increased estimate for storage, be aware that someone may have used the vSphere API to modify data collection.

Sample Code for Performance

The following table lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

Java (SDK\vsphere-ws\java\JAX-WS\samples\com\vmware\performance)
Basics.java
History.java
PrintCounters.java
RealTime.java
VITop.java
VIUsage.java

Diagnostics and Troubleshooting

vSphere includes several logs, which you can access and customize. You can also use the **DiagnosticManager** service interface for troubleshooting.

Troubleshooting Best Practices

Approach troubleshooting and problem-solving systematically, and take notes so you can trace your steps. Follow these guidelines to resolve issues with your client application.

- Do not change more than one thing at a time, and document each change and its result. Try to isolate the problem: Does it seem to be local, to the client? An error message generated from the server? A network problem between client and server?
- Use the logging facilities for your programming language to capture runtime information for the client application.
- Use the following VMware tools for analysis and to facilitate debugging.
  - vSphere Web Services API. The **DiagnosticManager** service interface allows you to obtain information from the server log files, and to create a diagnostic bundle that contains all system log files and all server configuration information. The vSphere Client and the MOB provide graphical and Web based access to the **DiagnosticManager**. **PerformanceManager** supports exploration of bottlenecks. See vSphere Performance.
  - Managed Object Browser (MOB). The MOB provides direct access to live runtime server-side objects. You can use the MOB to explore the object hierarchy, obtain property values, and invoke methods. See Managed Object Browser.
  - VMware vSphere Client GUI. The vSphere Client allows you to examine log files for ESXi, vCenter Server, and virtual machines, and to change log level settings. Use vSphere Client menu commands to create reports that summarize configuration information, performance, and other details, and to export diagnostic bundles. The vSphere Client maintains its own local log files.

Overview of Configuration Files and Log Files

ESXi and vCenter Server configuration files control the behavior of the system. Most configuration file settings are set during installation, but can be modified after installation. Log files capture messages generated by the kernel and different subsystems and services. ESXi and vCenter Server services maintain separate log files. The following table lists log files or reports, their locations and associated configuration files.

Description	Log Location	Filename or Names	Configuration File
ESXi service log	/var/log/vmware/	**hostd.log [hostd-0.log, ...hostd-9.log]**	See ConfigStore
vCenter Server agent log	/var/log/vmware/vpx/	vpxa.log	See ConfigStore

Table continued on next page

Continued from previous page

Description	Log Location	Filename or Names	Configuration File
Virtual machine kernel core file	`/root/`	**`vmkernel-core.<date>`** **`vmkernel-log.<date>`**	See ConfigStore
syslogd log	`/var/log/`	**`messages`** **`[messages.1,...`** **`messages.4]`**	See ConfigStore
Service console availability report	`/var/log/`	**`vmkernel`** **`[vmkernel.1, ...`** **`vmkernel.8]`**	See ConfigStore
VMkernel messages, alerts, and availability reports	`/var/log/vmkernel`		See ConfigStore
VMkernel warning	`/var/log/`	**`vmkwarning`** **`[vmkwarning.1 ... 4`** **`for history]`**	See ConfigStore
Virtual machine log file	`vmfs/volume/` `<vm_name>`	`vmware.log`	`<vm_name>/` `<vm_name>.vmx`

For developers, the following files are most relevant:

- `hostd.log` – Host daemon log, see ESXi Log File. Can be used as a SOAP monitor when set to trivia log level as in Generating Logs.
- `vpxa.log` – Agent log file found on each managed ESXi system.
- `vmware.log` – Virtual machine log. See Virtual Machine Log Files.

> **NOTE**
> Many configuration settings can no longer be edited directly with a web browser. To modify configuration settings in ConfigStore, use `/bin/configstorecli` or `esxcli`.

In addition to viewing log files in real time you can also generate reports and complete diagnostic bundles. See Generating Diagnostic Bundles.

ESXi Log File

The ESXi log (`hostd.log`) captures information of varying specificity and detail, depending on the log level. Each request to the server is logged.

You can view the file using the vSphere Client. The raw text form of an ESXi (**`hostd`**) log file is shown in Sample ESXi Log (hostd.log) Data.

Sample ESXi Log (`hostd.log`) Data

```
...

[2008-05-07 09:50:04.857 'SOAP' 2260 trivia] Received soap response from
[TCP:myservername.vmware.com:443]: GetInterfaceVersion

[2008-05-07 09:50:04.857 'ClientConnection' 2260 info] UFAD interface version is vmware-
converter-4.0.0

[2008-05-07 09:50:04.857 'SOAP' 2260 trivia] Sending soap request to
[TCP:myservername.eng.vmware.com:443]: logout
```

```
[2008-05-07 09:50:04.857 'ProxySvc Req00588' 3136 trivia] Client HTTP stream read error

[2008-05-07 09:50:04.872 'ProxySvc Req00612' 3136 trivia] Request header:

POST /vmc/sdk HTTP/1.1

User-Agent: VMware-client

Content-Length: 435

Content-Type: text/xml; charset=utf-8

Cookie: vmware_soap_session="F127B435-56C7-4580-BAC4-3034DA1E67B6"; $Path=/

Host: myservername.vmware.com



[2008-05-07 09:50:04.872 'ProxySvc Req00588' 3816 trivia] Closed

[2008-05-07 09:50:08.450 'App' 3560 verbose] [VpxdHeartbeat] Invalid heartbeat from
10.17.218.46

[2008-05-07 09:50:10.013 'App' 3560 verbose] [VpxdHeartbeat] Queuing 10.17.218.45:829
(host-55)

[2008-05-07 09:50:10.013 'App' 1928 verbose] [HeartbeatHandler] 50208862-2752-d94c-2a73-
fa2ec9e38ecc:829 (host-55)
```

Virtual Machine Log Files

Each running virtual machine has its own log file, `vmware.log`, stored on the VMFS volume. By default, the log file is rotated whenever the virtual machine is powered on, but file rotation is configurable.

- ESXi maintains six log files that rotate at each power-cycle (the default) or at a configured file size.
- ESXi can be configured to maintain a specific number of log files. When the limit is reached, the oldest file is deleted.
- VMware recommends a log file size of 500 KB.
- Messages that are generated by VMware Tools are logged separately.

VMkernel Availability Report

```
Availability Report for <servername>

Feb 27, 2008 - May  7, 2008


Availability: 99.949%

     Total time: 69 days, 15 hours

     Uptime: 69 days, 14 hours

     Downtime: 51 minutes


Note: Downtime is any time the system isn't capable of running
```

```
Virtual Machines.  This includes reboots, crashes, configuration and running linux

Downtime Analysis:
     0.1% (51 minutes) downtime caused by:
     13.1% (6 minutes) scheduled downtime
     86.9% (44 minutes) unscheduled downtime

Reasons for scheduled downtime:
     84.9% server rebooting (1 instance)
     9.4% VMkernel unloaded (1 instance)
     5.7% server booting (3 instances)

Reasons for unscheduled downtime:
100.0% unknown (powerfail / reset?) (1 instance)

Stats:
     Current uptime: 8 days, 11 hours
     Longest uptime: 61 days, 2 hours
     Shortest uptime: 38 minutes
     Average uptime: 23 days, 4 hours
     Longest downtime: 44 minutes
     Shortest downtime: 7 seconds
     Average downtime: 8 minutes
     Maximum VMs Sampled:  1
     Average VMs Sampled: 0.94

Server Information:   Number of CPUs:  4 logical 4 cores
      2 packages,  Intel(R) Xeon(R) CPU            5150  @ 2.66GHz
        Installed Memory:     2096416 kB
        Current Build:  78591
Report generated Wed May  7 04:02:04 PDT 2008
```

vCenter Server Log Files

vCenter Server log files are located by default in the Documents and Settings subdirectory of the Windows account used to install the software. For example:

```
C:\Documents and Settings\Administrator\Local Settings\Application Data\VMware\
```

> **IMPORTANT**
> VMware recommends creating a user account especially for vCenter Server installation.

By default, the log files are hidden files. See the procedure for your Windows operating system to make the files visible.

Modifying the Log Level to Obtain Detailed Information

The amount of information captured in the log files varies, depending on the level setting.

Log Level Setting	Description
None	Disables logging.
Error	Logging limited to error messages.
Warning	Error messages plus warning messages are logged.
Info	Default setting on ESXi and vCenter Server systems. Errors, warnings, plus informational messages about normal operations are logged. Acceptable for production environments.
Verbose	Can facilitate troubleshooting and debugging. Not recommended for production environments.
Trivia	Extended verbose logging. Provides complete detail, including content of all SOAP messages between client and server. Use for debugging and to facilitate client application development only. Not recommended for production environments.

For example, the **`hostd`** service running on ESXi systems has a default log level setting of **`info`**. The vCenter Server logs are controlled by settings through the vSphere Client.

Setting the Log Level on ESXi Systems

The default log level setting for the ESXi Host Agent is **`info`**. If you run into issues during development, you can set the log level to **`verbose`**, or to **`trivia`** to obtain SOAP message content to use in debugging.

To change the log level for hostd on an ESXi system, use the following steps.

1. Connect to the ESXi system using the vSphere Client.
2. On the **Host › Manage** screen, select the **System** tab.
3. On the **System** tab screen, select **Advanced Settings**.
4. Scroll to find the `Config.HostAgent.log.level` setting.
   By default this setting is **`info`**.
5. Select the `Config.HostAgent.log.level` setting and click **Edit option**.
   A dialog box opens, where you can enter a new setting.
6. Use the drop-down menu to select a new setting, such as **Verbose**, and click **Save**.
   The **Recent Tasks** pane updates to confirm that the change completed.

After the service restarts, the new log level is in effect.

Generating Logs

If you are connected to ESXi by SSH, you can use the **`tail`** command to explicitly create a log file that captures detail about actions that follow. For example, you can use the vSphere Client to create a new virtual machine and then use the content from the log as a model for how to create your own code.

To start the logging process and capture content to a file, use the following steps.

1. Navigate to the location of the `hostd.log` file:

   ```
   cd /var/log/vmware
   ```

2. Run the **`tail`** command, passing a filename in which to capture output:

   ```
   tail -f hostd.log > yourfilenamehere
   ```

3. Use the vSphere Client to perform whatever action you are having difficulty modeling in your own code. For example, create a new virtual machine and stop the **`tail`** process with Ctrl-C when the operation completes.

The file contains the SOAP message content and other log messages sent and received by **`hostd`** during the execution.

Setting the Log Level on vCenter Server Systems

To change log-level settings on vCenter Server, you must use the vSphere Client.

To set logging level for vCenter Server using the VMware vSphere Client, use the following steps.

1. Log in to the vSphere Client and connect to the vCenter Server instance.
2. Choose **Administration** and click **Server Settings › Logging Options**.
3. Choose **Trivia** from the pop-up menu and click **OK**.

Using DiagnosticManager

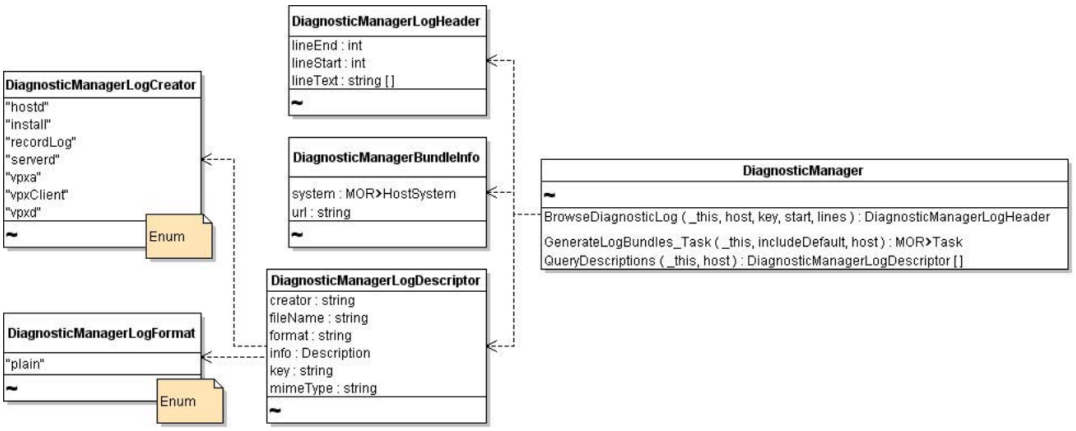
The vSphere API provides access to the **`DiagnosticManager`**, the service interface for obtaining information from the log files and for generating diagnostic bundles. The logs are populated based on configuration settings, such as **`info`**, **`trivia`**, and so on.

The **`DiagnosticManager`** is a managed object that works service-wide, rather than on a per-session basis. The **`DiagnosticManager`** has no properties, but provides operations for these tasks:

- Obtaining information about the logs and how they have been defined.
- Generating a diagnostic bundle that can be sent to VMware support for analysis.

Figure: DiagnosticManager Managed Object and Associated Data Objects shows a UML class diagram for **`DiagnosticManager`**, which is available on ESXi and vCenter Server systems.

Figure 68: DiagnosticManager Managed Object and Associated Data Objects



As shown in Figure: DiagnosticManager Managed Object and Associated Data Objects, **`DiagnosticManager`** supports these methods:

- **`BrowseDiagnosticLog`**
- **`GenerateLogBundleTask`**
- **`QueryDescriptions`**

The **`DiagnosticManagerLogDescriptor.creator`** property contains the creator of the log, which is the system or subsystem that controls a specific log.

The **`creator`** value is populated from the **`DiagnosticManagerLogCreator`** enumeration. The following table lists all string values currently available from the **`DiagnosticManagerLogCreator`** enumeration that can populate the **`creator`** property of the **`DiagnosticManagerLogDescriptor`** data object.

Name	Description
`hostd`	Host daemon
`install`	Installation
`recordLog`	System record log
`serverd`	Host server agent
`vpxa`	vCenter agent
`vpxClient`	vSphere Client
`vpxd`	vCenter service

Creator	File Name	Format	Info.label	Info.summary	Key	Mime Type
`hostd`	`/var/log/vmware/hostd.log`	plain	ESX Log	ESX log in plain format	**`hostd`**	text/plain
`hostd`	`/var/log/messages`	plain	ESX Log	ESX log in plain format	**`messages`**	text/plain
`hostd`	`/var/log/vmkernel`	plain	ESX Log	ESX log in plain format	**`vmkernel`**	text/plain
`hostd`	`/var/log/vmksummary.txt`	plain	ESX Log	ESX log in plain format	**`vmksummary`**	text/plain
`hostd`	`/var/log/vmkwarning`	plain	ESX Log	ESX log in plain format	**`vmkwarning`**	text/plain

Table continued on next page

Continued from previous page

Creator	File Name	Format	Info.label	Info.summary	Key	Mime Type
vpxa	/var/log/vmware/vpx/vpxa.log	plain	vCenter Agent Log	vCenter agent log in plain format	**vpxa**	text/plain

Using the MOB to Explore the DiagnosticManager

The Managed Object Browser (MOB) is a graphical interface that allows you to navigate the objects on a server and to invoke methods. You can access the **DiagnosticManager** using the MOB.

The following steps show how to explore DiagnosticManager.

1. Start the mob by typing the MOB URL (https://hostname.yourcompany.com/mob) into a Web browser.
2. In the **ServiceContent** data object, click the link (**ha-diagnosticmanager** or **DiagMgr**) in the **Value** column for the **diagnosticManager** property, to navigate to the **DiagnosticManager** for the system.
   - For ESXi, **ha-diagnosticsmanager** is the managed object ID.
   - For vCenter Server, **DiagMgr** is typically the managed object ID.
3. Click the link to the reference to display the managed object reference to the **DiagnosticManager** in the MOB.

   DiagnosticManager provides three operations that allow you to obtain information about the descriptions currently available in the log file and log file content.

   Because **DiagnosticManager** can track multiple ESXi systems, you can use the **QueryDescriptions** operation to return the names of keys used for all hosts. From this array, select the key for the host from which you want to obtain the log file.
4. On **QueryDescriptions**, click the **Invoke Method** link.

   The vCenter Server system returns the contents of the log file for the selected host as a string array for the **lineText** property of **DiagnosticManagerLogHeader**.

The string array returned through the MOB in this way is the content of the log file. The content contained in the log file is the same content that is available through the following other mechanisms:

- Displayed in the vSphere Client
- Included in a diagnostic bundle created through the **DiagnosticManager.GenerateLogBundles_Task** method.
- Available in the hostd.log file
- Returned to a client application that you write

For more information about using the DiagnosticManager, see Managed Object Browser.

Generating Diagnostic Bundles

Typically, customers create diagnostic bundles at the request of VMware technical support. Diagnostic bundles also allow developers to quickly obtain all configuration files and log files in a complete package.

The generated compressed files are packaged in a file having the following pattern:
<fqdn-hostname>-esxsupport-yyyy-mm-dd@hh-mm-ss.tgz

Export Diagnostic Data By Using the vSphere Client

You can use the vSphere Client interface to generate diagnostic bundles.

1. Connect to the ESXi system using the vSphere Client.
2. On the **Host › Monitor** screen, select the **Logs** tab.
   The **Recent Tasks** pane updates to indicate that the system is generating a diagnostic bundle.

When the task completes, the vSphere Client opens a dialog box that allows you to download the bundle.

Managed Object Browser

The Managed Object Browser (MOB) is a graphical interface that allows you to navigate the objects on a server and to invoke methods. Any changes you make through the MOB take effect on the server.

This appendix explains how to use the MOB. The examples invoke **`PerformanceManager`** query methods to demonstrate how to pass primitive data types, arrays, and complex data types (data objects, including managed object references) using the MOB.

Using the MOB to Explore the Object Model

The Managed Object Browser, or MOB, is a Web-based server application available for all ESXi and vCenter Server systems. The MOB lets you examine the objects that exist on the server and navigate through the hierarchy of live objects by clicking on links. The MOB populates the browser with actual runtime information, for example, the names of properties.

> **CAUTION**
> Despite the word "browser" in its name, the MOB is not a read-only mechanism. The MOB allows you to make changes on the server by clicking the **InvokeMethod** link associated with methods.

Accessing the MOB

The MOB runs in a web browser and is accessed by using the fully-qualified domain name or IP address for the ESXi or vCenter Server system.

To access the MOB, use the following steps.

1. Start a Web browser.
2. Enter the fully-qualified domain name (or the IP address) for the ESXi or vCenter Server system:

   ```
   https://hostname.yourcompany.com/mob
   ```

3. Enter the user account and password for the system.

   If warning messages regarding the SSL certificate appear, you can disregard them and continue to log in to the MOB, if VMware is the certificate authority and you are not in a production environment.

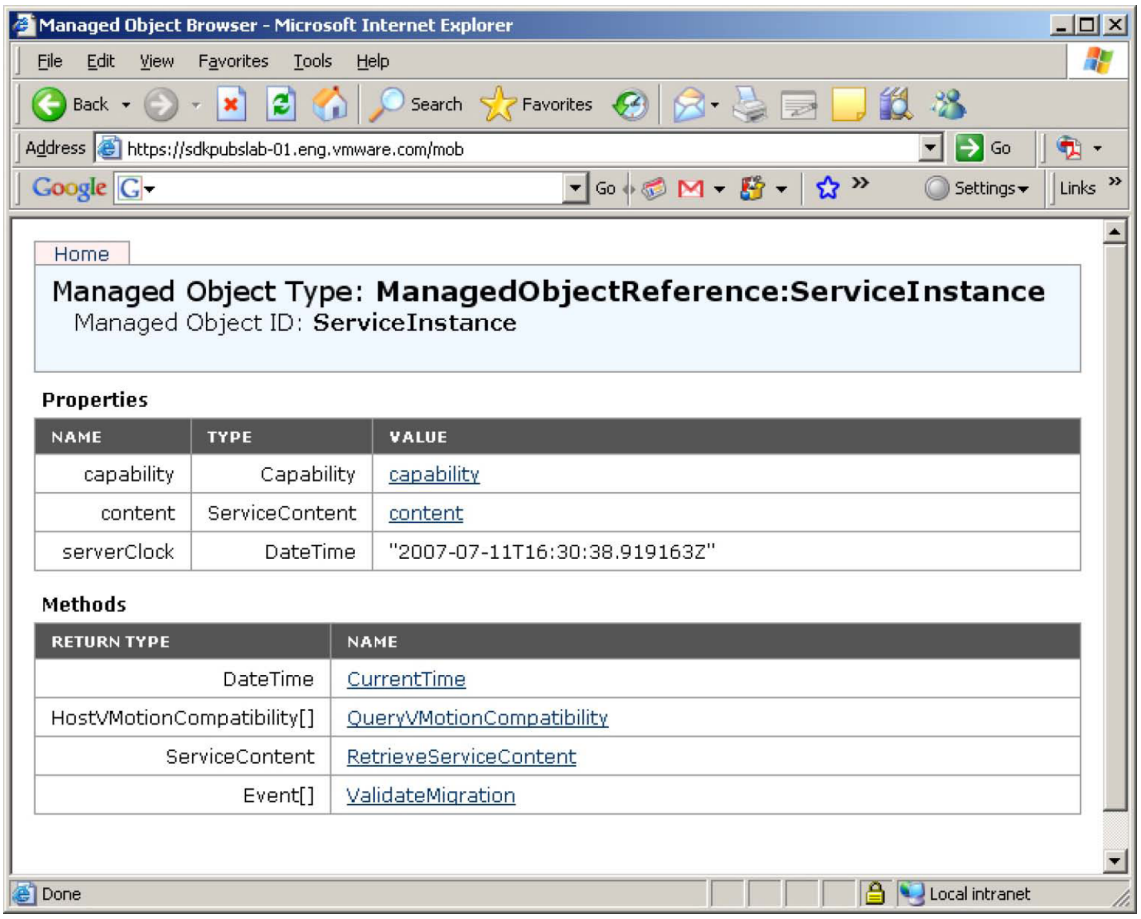
The MOB reveals the underlying structures of the object model. Seeing the structure in conjunction with the *`API Reference Guide`*, can help with understanding the model.

Using the MOB to Navigate the vSphere Object Model

Upon successful connection to the MOB, the browser displays the managed object reference for **`ServiceInstance`**. Client applications do not use managed objects directly, but interact with server-side managed objects by reference, using instances of the **`ManagedObjectReference`** data created for this purpose.

The page lists the properties and methods available through a `ServiceInstance` object. The **`ServiceInstance`** methods and properties provide access to the entire set of services and inventory objects available on the server. See Datacenter Inventory.

Figure 69: Managed Object Browser Connected to a vCenter Server



The MOB lets you examine the relationships among objects by looking at the properties and their values, and then drilling down into the objects. To explore the objects on the server, click the links in the **Value** column to navigate to the page that displays the object.

For example, to find out more about **`ServiceContent`**, click the **content** link to display the **`ServiceContent`** data object instance.

Using the MOB to Invoke Methods

You can use the MOB to invoke methods as follows:

1. In the display of the object in which the method lives, click the name of the method.

   A browser window displays information about the parameter name and type and allows you to specify parameter values.

2. Specify parameter values, using the method appropriate for the type, and click **Invoke Method**.

   The rest of this section discusses how to pass different types of parameters to the MOB.

Passing Primitive Datatypes to Method

vSphere Web Services SDK data types are defined in the WSDL using XML Schema markup. The primitive data types are specified using the **`xsd`** namespace. For example, a string value for a property is defined as data type **`xsd:string`**. Enter a primitive value in the MOB as plain text, without quotation marks or other markup. For example, to enter an integer value of 10, type **`10`** in the field.

To obtain information about the available performance counters at level 4 on the server, enter a **`4`** in the **`level`** field of the **`PerformanceManager.QueryPerfCounterByLevel`** method. (This method is available only on the vCenter Server **`PerformanceManager`** API, not from an ESXi system.)

In response to the query, the array of **`PerfCounterInfo`** data objects and nested objects, with populated values from the server, displays in the Web browser.

Passing Arrays of Primitives to Methods

For an array, use the name of the parameter as the name of the property. For example, the **`PerformanceManager.QueryPerfCounter`** method requires an array of integers for the **`counterId`** parameter, as follows:

```
<counterId>58</counterId><counterId>65603</counterId><counterId>65604</counterId>
```

Even if you want to submit a single value for a single array element, you must wrap the parameter name around the value in this way.

Passing Complex Structures to Methods

For complex datatypes, enter the value as defined by the XML Schema in the WSDL. You can obtain the WSDL definition from the *vSphere API Reference* using the **Show WSDL type definition** links. Each data object type has an associated link.

Simple Content

The data object type **`ManagedObjectReference`** is one of the most commonly required parameters to be passed to the server. For example, the MOB for the **`PerformanceManager.QueryPerfProviderSummary`** method shows that the method requires a single parameter, the managed object reference (an instance of **`ManagedObjectReference`**) of the entity for which you want to obtain the **`PerfProviderSummary`** object.

Using the *vSphere API Reference* for **`ManagedObjectReference`** type, you can obtain the schema information from the **Show WSDL type definition** link at the bottom of the documentation page for **`ManagedObjectReference`**.

XML Schema Definition of ManagedObjectReference Data Object

```
<complexType xmlns="http://www.w3.org/2001/XMLSchema" xmlns:vim25="urn:vim25"
name="ManagedObjectReference">

     <simpleContent>

          <extension base="xsd:string">

              <attribute name="type" type="xsd:string"/>
```

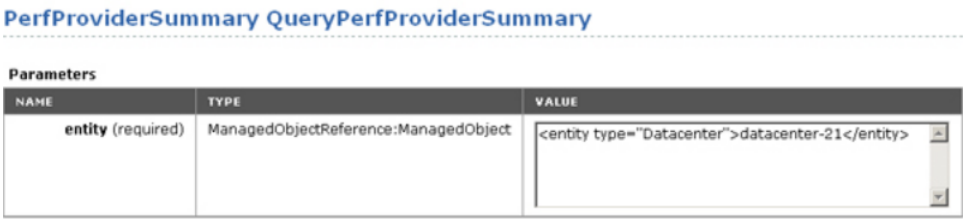
```
          </extension>
     </simpleContent>
</complexType>
```

XML Schema Definition of ManagedObjectReference Data Object shows that a managed object reference is defined as a **<SimpleContent>** element that consists of a string that specifies the attribute **type** with its associated value, also as string. Use this information to construct the appropriate structure by replacing **type** with the parameter name from the MOB, setting the value as needed, and submitting in the entry field of the MOB. (The value for the **Datacenter** is displayed in the MOB.)

```
<entity type="Datacenter">datacenter-21</entity>
```

Figure: Using the MOB to Pass Complex Types to a Method shows the result of using the definition listed in XML Schema Definition of ManagedObjectReference Data Object to specify the managed object reference for a target datacenter to the **PerformanceManager.QueryPerfProviderSummary** method.

Figure 70: Using the MOB to Pass Complex Types to a Method



As another example, one of the parameters required by the **VirtualMachine.CloneVM_Task** method is a **folder**. In this case, the parameter is defined as a managed object reference to a specific **Folder** object. Using the same definition shown in XML Schema Definition of ManagedObjectReference Data Object, the result is as follows:

```
<folder type="Folder">folder-87</folder>
```

Although both examples submit a **ManagedObjectReference** to the MOB, each is specific to the parameter name required by the method (**entity** type for **PerformanceManager.QueryPerfProviderSummary** method, **folder** type for the **VirtualMachine.CloneVM_Task** method).

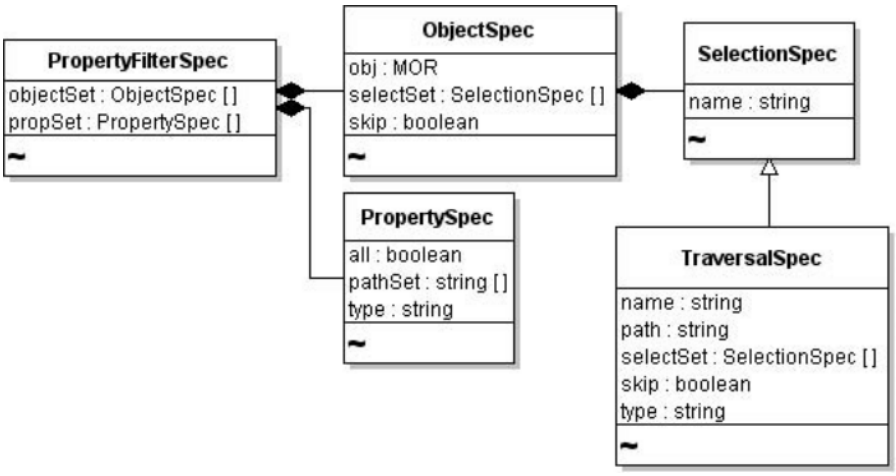
Complex Content

Many of the data objects required for method invocation consist of XML Schema elements defined as **<complexContent>** that can encompass many other elements.

For example, the **PropertyCollector.CreateFilter** method has a **spec** parameter that must be defined before method invocation. The **spec** parameter is defined as an instance of a **PropertyFilterSpec**.

Figure: PropertyFilterSpec and Associated Data Objects shows the relationships among several data objects that **PropertyFilterSpec** consists of.

Figure 71: PropertyFilterSpec and Associated Data Objects



To submit complex data structures such as this to the MOB, start by navigating the *vSphere API Reference*. Find the **PropertyFilterSpec** data object. Find the **Show WSDL type definition** link, and click it to display the XML Schema definition (see XML Schema Definition of PropertyFilterSpec Data Object Type).

XML Schema Definition of PropertyFilterSpec Data Object Type shows that the **PropertyFilterSpec** data object is a **<complexContent>** element that extends the **DynamicData** class with a sequence of two additional properties **propSet** (of type **PropertySpec)** and **objectSet** (of type **ObjectSpec**).

XML Schema Definition of PropertyFilterSpec Data Object Type

```
<complexType xmlns="http://www.w3.org/2001/XMLSchema" xmlns:vim25="urn:vim25"
name="PropertyFilterSpec">

     <complexContent>

          <extension base="vim25:DynamicData">

              <sequence>

                  <element name="propSet" type="vim25:PropertySpec" maxOccurs="unbounded"/
>

                  <element name="objectSet" type="vim25:ObjectSpec" maxOccurs="unbounded"/
>

              </sequence>

          </extension>

     </complexContent>

</complexType>
```

Because both elements are defined as a sequence, they must exist in the order listed. To obtain the definitions of **propSet** and **objectSet**, you must navigate further into the *vSphere API Reference*. XML Schema Extract for PropertySpec shows only the relevant parts of the XML Schema definition for **PropertySpec**. The **minOccurs="0"** attribute means that the element does not have to exist. The **maxOccurs="unbounded"** attribute means that the element can be populated as an array of any size. (When **minOccurs** is not set, but **maxOccurs** is set, the default for **minOccurs** defaults to 1, meaning one instance is required.)

XML Schema Extract for PropertySpec

```
<sequence>

     <element name="type" type="xsd:string"/><element name="all" type="xsd:boolean"
minOccurs="0"/><element name="pathSet" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"/>

</sequence>
```

Navigate through the *vSphere API Reference* to the **ObjectSpec** definition. ObjectSpec Definition as XML Schema shows the excerpt.

ObjectSpec Definition as XML Schema

```
...

<sequence>

     <element name="obj" type="vim25:ManagedObjectReference"/>

     <element name="skip" type="xsd:boolean" minOccurs="0"/>

     <element name="selectSet" type="vim25:SelectionSpec" minOccurs="0"
maxOccurs="unbounded"/>

</sequence>

...
```

Extrapolating from the WSDL definitions shown in XML Schema Definition of PropertyFilterSpec Data Object Type, XML Schema Extract for PropertySpec, and ObjectSpec Definition as XML Schema might produce results similar to those shown in CreateFilter Spec Property Entry.

CreateFilter Spec Property Entry

```
<spec>

     <propSet>

          <type>VirtualMachine</type>

          <all>false</all>

          <pathSet>config.guestFullName</pathSet>

     </propSet>

     <objectSet>

          <obj type="Folder">group-v4</obj>

          <skip>true</skip>

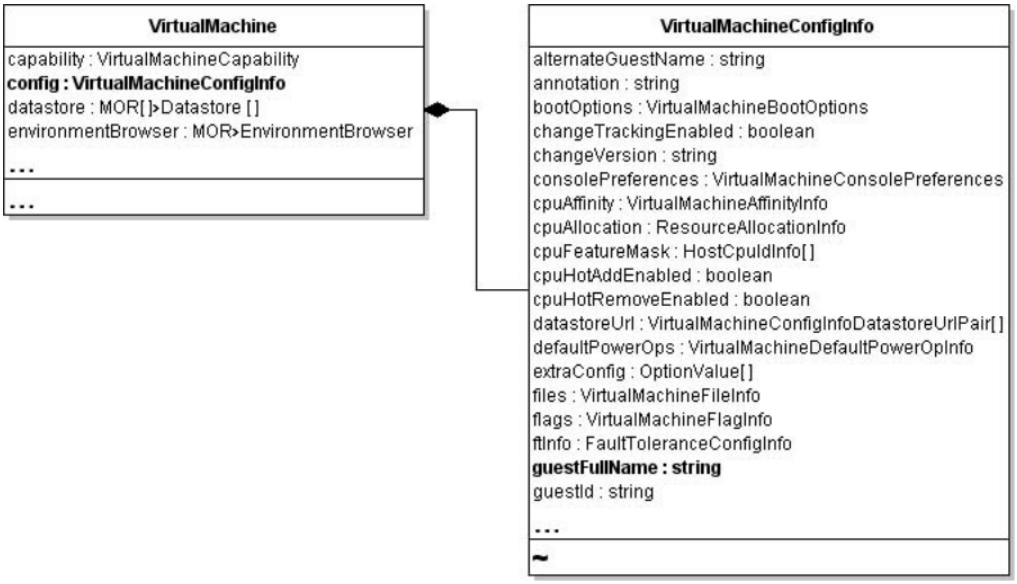
     </objectSet>

</spec>
```

In this example, the **<spec>** element identifies the **spec** parameter of the **CreateFilter** method. The order of the element tags is as defined in the XML Schema for the property (XML Schema Definition of PropertyFilterSpec Data Object Type). The **pathSet** property defines the full path to the nested data object of interest. In CreateFilter Spec Property

Entry, the **pathSet** property defines the path to the **guestFullName** property of the target virtual machine. ObjectSpec Definition as XML Schema shows the UML of these nested data objects.

Figure 72: Nested Data Objects



All of these details are available in the *vSphere API Reference*. By examining the WSDL definition, you can construct the strings needed to submit parameters through the MOB.

Using the MOB Along With the API Reference

The following table provides a brief summary of the steps involved when you use the MOB and the *vSphere API Reference* together.

Datatype	How to Input Values for Methods
Primitive	Enter the value as plain text regardless of its data type (**int**, **string**, **boolean**). Do not use quotes or other markup.
Array	Use the name of the parameter as the name of the element, wrap the values in a series of opening and closing tags for each array element.
Complex	Obtain XML Schema format information from the *vSphere API Reference* for the type (from the **Show WSDL type definition** link). Use the schema definition to construct the sequence of tags around the value (or values) you want to pass to the MOB.

HTTP Access to vSphere Server Files

In most cases, client applications interact with vSphere servers by using the vSphere Web Services SDK. In some cases, direct access to configuration files, log files, and other data on an ESXi or vCenter Server systems is more efficient.

Introduction to HTTP Access

ESXi and vCenter Server systems support file access using HTTP and secure HTTP. You can use HTTP/HTTPS for the following kinds of access.

- Datastore access on ESXi and vCenter Server systems.
- ESXi configuration and log file access on ESXi systems.

- Update bundle access on ESXi systems.

You can use the HTTP methods **GET**, **HEAD**, **PUT**, and **DELETE** to access files. The URL of the HTTP/HTTPS request must contain an embedded keyword that specifies the type of access. The following table shows the server access types with the corresponding URL keyword and HTTP methods.

Server Access	URL Keyword	HTTP Method or Methods
Datastore	folder	**GET**, **HEAD**, **PUT**, **DELETE**
ESXi configuration file	host	**GET**, **HEAD**, **PUT** (See Host File Access to vSphere Server Files (/host) for the specific methods supported for each file type.)
Update bundle	tmp	**PUT**

Use the **PUT** method to create new files or overwrite existing files. You can create a subdirectory by using a URL that is consistent with the supported top-level directories. You cannot create datastores or datacenters because the URL must refer to a valid datacenter or datastore.

You can use a Web browser to browse and download files. You cannot use a Web browser to post or delete files.

URL Syntax for HTTP Access

The URL specification in an HTTP request to a vSphere server includes one of the following keywords, which determines the type of access.

- Datastore Access (/folder)
- Host File Access to vSphere Server Files (/host)
- Update Package Access (/tmp)

Datastore Access (/folder)

An HTTP request for datastore access uses the following syntax:

```
http-method http[s]://server/folder[[/path]?dcPath=path[&dsName=name]]
```

http-method	One of the methods **GET**, **HEAD**, **PUT**, or **DELETE**.
http:// or https://	Access protocol (standard access or secure access).
server	ESXi or vCenter Server target system. The server value can be an IP address or a DNS name.
/**folder**	Specifies datastore access on an ESXi or vCenter Server system. The datastore URL can include the following optional elements: • *path* – Path to a file or directory in the datastore, relative to the root of the datastore. • **dcPath** – Inventory path to a datacenter. Specify the datacenter path as a name-value pair in the request. For a datacenter named **Datacenter** located in the root folder, the **dcPath** value is **MyDatacenter**. For a datacenter named **YourDatacenter** located in the folder **NorthAmerica** which is located in the root folder, the **cdPath** value is NorthAmerica/YourDatacenter. • **dsName** – Datastore associated with the datacenter. Specify the datastore name as a name-value pair in the request.

The following examples illustrate the syntax. If the target server is an ESXi system, **dcPath=DCPATH&** is optional and defaults to **dcPath=ha-datacenter**.

Example	Description
`/folder`	Directory listing of known datacenters on this server.
`/folder?dcPath=` **`path`**	Directory listing of all datastores available at the specified datacenter.
`/folder?dcPath=` **`path`** **`&dsName=`** **`name`**	Top-level directory listing of the datastore.
`/folder/` **`path`** **`?dcPath=`** **`path`** **`&dsName=`** **`name`**	Directory listing of all files in a datastore directory.

`/folder/`
`path`
`/disk-flat.vmdk?dcPath=`
`path`
`&dsName=`
`name`
	Access individual files.

Host File Access to vSphere Server Files (/host)

An HTTP request for access to ESXi configuration files uses the following syntax:

```
GET http[s]://my_system/host

http-method http[s]://my_system/host/file
```

Syntax Element	Description
http-method	One of **GET**, **HEAD**, or **PUT**, depending on the type of configuration file (see the following table).
http:// or https://	Access protocol (standard access or secure access).
esx-server	IP address or a DNS name.
/host	List of configuration files that you can access. (Use /host to retrieve the list.)
/host/**file**	A specific ESXi configuration file.

The following table shows ESXi host configuration files and the corresponding HTTP/HTTPS methods for access. The set of files might change from version to version.

Configuration File	HTTP Access Method(s)	Configuration File	HTTP Access Method(s)
esx.conf	**GET, HEAD, PUT**	ipmi0_sel.raw	**GET, HEAD**
openwsman.conf	**GET, HEAD, PUT**	**ipmi0_sel**	**GET, HEAD**
proxy.xml	**GET, HEAD, PUT**	ipmi0_sdr_content.raw	**GET, HEAD**
ssl_cert	**GET, HEAD, PUT**	ipmi0_sdr_header.raw	**GET, HEAD**
ssl_key	**PUT**	ipmi0_sensor_readings.raw	**GET, HEAD**
vmware_config	**GET, HEAD, PUT**	**ipmi1_sel**	**GET, HEAD**
vmware_configrules	**GET, HEAD, PUT**	ipmi1_sdr_content.raw	**GET, HEAD**
		ipmi1_sdr_header.raw	**GET, HEAD**
		ipmi1_sensor_readings.raw	**GET, HEAD**
		ipmi2_sel.raw	**GET, HEAD**
		ipmi2_sel	**GET, HEAD**
		ipmi2_sdr_content.raw	**GET, HEAD**
		ipmi2_sdr_header.raw	**GET, HEAD**
		ipmi2_sensor_readings.raw	**GET, HEAD**
		ipmi3_sel.raw	**GET, HEAD**
		ipmi3_sel	**GET, HEAD**
		ipmi3_sdr_content.raw	**GET, HEAD**
		ipmi3_sdr_header.raw	**GET, HEAD**
		ipmi3_sensor_readings.raw	**GET, HEAD**

Update Package Access (/tmp)

An HTTP request for update package access uses the following syntax:

```
PUT http[s]://esx-server/tmp/file-path
```

http:// or https://	Access protocol.
esx-server	IP address or a DNS name.
/tmp/**file-path**	Target file on an ESX/ESXi system.

Privilege Requirements for HTTP Access

HTTP access to a vSphere file is access to a datastore object that is associated with the folder structure in the vSphere inventory. HTTP access requires the same privileges needed to obtain these files using any other mechanism, such as the vSphere Client.

The following table shows the required privileges.

Object Associated with File	Portion of URL	Required Privileges
Root folder	/folder	System.View
Datacenter	**?dcPath**	**Datastore.Browse**
		Datastore.FileManagement
Datastore	**&dsName**	**Datastore.Browse**
		Datastore.FileManagement
Host	/host	**Host.Config.AdvancedConfig**
	/tmp/	**Host.Config.SystemManagement**

Sample Program Overview

The VMware vSphere Web Services SDK includes samples for the Java platforms. This chapter lists the available sample programs and provides some information about each program. The samples have been re-compiled with JAX-WS bindings, and they use JAX-WS credential store classes that allow you to ignore certificates when you connect to a server with the samples.

Java Sample Programs (JAXWS Bindings)

When you download the SDK, you can find the Java sample programs and related files in the following directories.

- SDK\vsphere-ws\java\JAXWS\samples\com\vmware – Top-level directory for Java samples. Details listed in the following table.
- SDK\vsphere-ws\java\JAXWS\samples\com\vmware\security – Credential store utilities
- SDK\vsphere-ws\java\JAXWS\samples\com\vmware\vim25 – Stub directory. The **vim25** directory contains stubs for release 2.5 and later.
- SDK\vsphere-ws\java\JAXWS\samples\com\vmware\vm – samples written for a single VM

Directory	Example	Description
alarms	**VMPowerStateAlarm**	Create an alarm to monitor a virtual machine's power state.
cim	**CIMReader**	Walks the Common Information Model (CIM) data associated with an ESXi host.
connection	**BasicConnection** (among others)	Shows how to set up a vSphere connection with user name and password.

Table continued on next page

Continued from previous page

Directory	Example	Description
events	**EventFormat**	Retrieve and format the last event from the host daemon or **vpxd**. Includes a function that formats the event message.
	EventHistoryCollectorMonitor	Demonstrates how to create and monitor an **EventHistoryCollector**. Uses the **latestPage** property of **EventHistoryCollector** to filter the events.
	VMEventHistoryCollectorMonitor	Standalone client that demonstrates how to perform the following tasks: (1) Logging into the web service. (2)Creating **EventHistoryCollector** filtered for a single virtual machine. (3) Monitoring events using the **latestPage** property of the **EventHistoryCollector**.
fcd	**FcdCreate**, **FcdDelete**	Create and delete a first class disk (FCD) virtual storage object.
	FcdAttachToVM, **FcdDetachFromVM**	Attach or detach FCD to or from VM.
	FcdRegisterLegacyDisk	Register legacy disk as a first class disk.
general	**Browser**	Print all managed entities and for each entity its type, reference value, property name, property value, inner object type, inner reference value and inner property value.
	Connect	Connect to an ESXi host or a vCenter Server system.
	Create	Create a managed entity such as a folder, datacenter, or cluster.
	Delete	Delete a managed entity from the inventory tree. The managed entity can be a virtual machine, a **ClusterComputeResource**, or a folder.
	GetCurrentTime	Retrieve the current time from the vSphere Server.
	GetHostName	Retrieve the hostname of the ESXi Server.
	GetUpdates	Demonstrates how to use the **PropertyCollector** to monitor one or more properties of one or more managed objects. In particular this sample monitors one or all virtual machines and all hosts or one host for changes to some basic properties.
	LicenseManager	Demonstrates uses of the licensing API using **License** managed object reference.
	Move	Move a managed entity from its current location in the inventory to a new location in a specified folder
	PropertyCollector	Illustrates the use of the **PropertyCollector** API.
	RemoveManagedObject	Destroy or unregister a managed inventory object like a **Host**, **VirtualMachine**, **or Folder**.
	Rename	Rename a managed entity object.
	SearchIndex	Illustrates the use of the **SearchIndex** API.

Table continued on next page

Continued from previous page

Directory	Example	Description
	SimpleClient	Connect to the server, log in, list the inventory contents (managed entities) at the console, and log out.
	TaskList	Display a list of tasks performed on a specified managed object.
guest	**CreateTemporaryFile**	Create a temporary file inside a virtual machine.
	DownloadGuestFile	Download a file from the guest to a specified path on the host where the client is running.
	RunProgram	Run a specified program inside a virtual machine. RunProgram re-directs output to a temporary file inside the guest and downloads the output file.
	UploadGuestFile	Upload a file from the client machine to a specified location inside the guest operating system.
hcl	**HCIBatchAddHostAndExtendCluster**	Call batch-add-host to add four or more new hosts to an HCI cluster.
	HCIComputeOnlyCluster	Call HCI configure to create a simple cluster configuration.
host	**AcquireSessionInfo**	Acquire session with a vCenter Server or ESXi host and print a CIM service ticket and related session information to a file.
	AddVirtualNic **RemoveVirtualNic**	Add a virtual NIC to a port group on a virtual switch. Remove a virtual NIC from a port group.
	AddVirtualSwitch **RemoveVirtualSwitch**	Add a virtual switch to a host. Remove a virtual switch from the host.
	AddVirtualSwitchPortGroup **RemoveVirtualSwitchPortGroup**	Add a port group to a virtual switch. Remove a port group from a virtual switch.
	DVSCreate	Create a distributed virtual switch.
	HostProfileManager	Demonstrates the use of **HostProfileManager** and **ProfileComplainceManager**.
	NIOCForDVS	Add a network resource pool to a distributed virtual switch.
httpfileaccess	**GetVMFiles**	Retrieve configuration files, snapshot files, log files, and virtual disk files of a virtual machine; place them on the system on which the program is run.
	PutVMFiles	Put virtual machine files into a specified datacenter and datastore; register and reconfigure the corresponding virtual machine.
performance	**Basics**	Display available performance counters or other metadata for an ESXi host.
	History	Read performance measurements from the current time, or from a specified start time, for a specified duration.
	PrintCounters	Write the available counters of a managed entity into the specified file at the specified location. The managed entity can be a host system, a virtual machine, or a resource pool.

Table continued on next page

Continued from previous page

Directory	Example		Description
	RealTime		Display performance measurements from the current time at the console.
	VItop		An **ESXtop**-like sample application that lets administrators specify the CPU and memory counters by name to obtain metrics for a specified host.
	VIUsage		Create a GUI for graphical representation of the counters.
scheduling	**DeleteOneTimeScheduledTask**		Demonstrates deleting a **ScheduledTask**.
	OneTimeScheduledTask		Demonstrates creating a **ScheduledTask** using the **ScheduledTaskManager**.
	WeeklyRecurrenceScheduledTask		Demonstrates creating a weekly recurrent scheduled task.
scsilun	**SCSILunName**		Display the **CanonicalName**,**Vendor**, **Model**, **Data**, **Namespace** and **NamespaceId** of the host's SCSI LUN.
security	**credstore**	**Base64**	A fast, memory efficient class that encodes and decodes to and from BASE64 in full accordance with RFC 2045.
		CredentialStore	Create an example credential store.
		CredentialStoreAdmin	A command-line tool that provides complete access to the credential store backing file on the local machine.
		CredentialStoreCipher	This class uses the JVM provided classes in javax.crypto to encrypt and decrypt text.
		CredentialStoreFactory	Factory class providing instances of a credential store.
		CredentialStoreImpl	Implementation class for **CredentialStoreAdmin**.
		CredentialStoreStorage	This class provides the same functionality as **FileInputStream**, except that the close() method is overridden so that **FileInputStream** class' close() method does not get called.
simpleagent	**CreateUser**		Create a user account and password and store them in the local credential store.
	SimpleAgent		Access the local credential store to obtain a single user account for login to a server.
storage	**CreateStorageDRS**		Creates storage DRS.
	SDRSRecommendation		Runs storage DRS on an SDRS cluster to obtain SDRS recommendations.
	SDRSRules		Configures rules for an SDRS cluster.
vApp	**OVFManagerExportVapp**		Demonstrates the **OvfManager** by exporting VMDKs and OVF Descriptors of all VMs in the vApps.
	OVFManagerExportVMDK		Demonstrates how the **OvfManager** exports VMDKs from a VM to the localSystem.
	OVFManagerImportLocalVapp		Use this class to import or deploy an OVF Appliance from a local drive.

Table continued on next page

Continued from previous page

Directory	Example	Description
	OVFManagerImportVAppFromUrl	Use this class to import or deploy an OVF Appliance from a specified URL.
vim25	This directory contains the many Java classes that define JAX-WS bindings to the vSphere API.	
vm	**VMApplyEvc**	Apply a per-VM extended vMotion compatibility (EVC) setting to an existing virtual machine.
	VMClone	Locate an existing virtual machine on the vCenter Server system, make a template from this virtual machine, and deploy instances of the template onto a datacenter.
	VMCreate	Create a virtual machine. Different command-line input creates the virtual machine in different ways.
	VMDeltaDisk	Create a delta disk on top of an existing virtual disk in a virtual machine, and simultaneously removes the original disk using the reconfigure API. Use delta disks in conjunction with linked virtual machines.
	VMDiskCreate	Create a virtual disk.
	VMLinkedClone	Create a linked virtual machine from an existing snapshot.
	VMManageCD	Configure a CDROM for a virtual machine. Also list information about the CDROMs associated with a virtual machine.
	VMManageFloppy	Configure a floppy drive for a virtual machine. Also list information about the floppy drives associated with a virtual machine.
	VMotion	Check whether migration with **VMotion** is feasible between two hosts. Perform a migration if the hosts are compatible.
	VMpowerOps	Perform power operations on a virtual machine.
	VMPromoteDisks	Consolidate a linked virtual machine by using the **VirtualMachine.PromoteDisks** method.
	VMReconfig	Reconfigure a virtual machine. Includes reconfiguring the disk size and disk mode.
	VMRelocate	Relocate a linked virtual machine using disk move type.
	VMSnapshot	Perform virtual machine snapshot operations
	XVCvMotion	Relocate virtual machine across from one vCenter Server to another.
	XVMotion	Relocate virtual machine to the computing resource recommended by the DRS.

VMware Storage Policy SDK Programming Guide

The *VMware Storage Policy Programming Guide* describes how to use the VMware® Storage Policy API.

The VMware Storage Policy SDK supports the development of vCenter clients that use storage profiles for virtual machine configuration. This method of storage administration is called storage policy based management (SPBM).

Revision History

This book is revised with each release of the product or when necessary. The following table summarizes significant changes in each version of this book.

Revision Date	Description
11 OCT 2022	vSphere 8.0 release. Minor cleanup.
02 APR 2020	Improved illustrations for vSphere 7.0.
17 Apr 2018	Revised for vSphere 6.7 with new chapter about VVol storage profiles.
16 Nov 2016	Revised for vSphere 6.5 with minor updates.
12 Mar 2015	Revised for vSphere 6.0 with new chapters about Tag, VM, and vSAN storage profiles.
12 Sep 2013	First version of this manual for vSphere 5.5 with information about storage profiles.

Intended Audience

This book is intended for anyone who needs to develop applications using the VMware Storage Policy SDK. An understanding of Web Services technology and some programming background in Java is required.

Sample Code

The VMware Storage Policy SDK includes Java programs to list storage profiles, view a specific profile, create a new storage profile, associate a storage profile with a VM (when creating, cloning, or relocating the VM), check compliance of the VM with its storage profile, and delete a storage profile.

API Reference

Many of the SPBM interfaces start with the letters Pbm, for policy based management. To find an API reference information, go to https://code.vmware.com/sdksand click **vSphere Management SDK** then click **VMware Storage Policy API Reference**. There are Pbm interfaces in all categories: managed objects, data objects, enumerated types, fault types, methods, and so forth.

VMware Storage Policies

Storage Policy Based Management (SPBM) helps administrators automate VM provisioning on appropriate storage devices with requested data services. This avoids laborious manual provisioning of storage for individual VMs. For object-based datastores like vSAN and VVol, vendors formulate metadata describing storage traits, and advertise storage capabilities for profile queries.

Storage Capabilities

The storage provider describes capabilities of a storage array, which the storage policy service obtains and presents to the administrator to assist with VM provisioning.

On object-oriented datastores such as vSAN and VVol datastores, storage capabilities originate from software written by the vendor, called a VASA provider. VASA is an abbreviation of vSphere API for storage awareness. The storage policy service can match storage capabilities with VM storage policies formulated in vSphere.

For VVol datastores: storage capabilities may include array type, vendor, RAID level, read latency, write latency, snapshot information, backup frequency, replication, caching, compression, deduplication, and high availability.

For vSAN datastores: storage capabilities may include RAID type, failures to tolerate, disk stripes per object, flash read cache, IOPS limit, encryption, compression, and deduplication.

For VMFS and NFS datastores: administrators can select the VMware encryption storage policy. VM encryption is implemented on the ESXi host rather than on the storage array.

For tag-based policies: administrators can use the vSphere Web Client to define storage policy tags.

Virtual Machine Storage

Virtual machine configuration and virtual disk data reside in datastores.

Virtual machine configuration is stored in a file with the `.vmx` file extension. VM home files also include other files for virtual machine operation, such as a log file (`.log`), virtual firmware (`.nvram`), paging file (`.vmem`), and snapshot data files (`.vmsd`).

Virtual machine data is stored in virtual disks, in files with the `.vmdk` file extension.

Storage policies allow you to distinguish between virtual machine configuration and data files and to specify storage locations based on the needs of each.

Storage Capability Profiles

To use a storage capability profile, you define storage requirements and associate a virtual machine with a storage policy. When you create a VM, vCenter Server correlates its VM storage policy with the storage policy service to determine an appropriate location for VM home and virtual disk files.

A storage policy consists of a set of subprofiles. Each subprofile defines a set of storage capabilities, and corresponds to a numbered Rule Set in the vSphere Web Client. Several storage policies pre-exist in vSphere, including:

- The VM Encryption Policy is a sample storage policy for VM Home and virtual disk encryption.
- The vSAN Default Storage Policy can be easily modified, but is the storage policy used by default for vSAN datastores.
- The VVol No Requirements Policy allows the storage array, through the VASA provider, to determine the best placement strategy for VM Home and virtual disks.

In the vSphere Web Client, all of these storage policies, and ones you create, have a **Check Compliance** button to verify that virtual machines assigned to that storage policy are correctly placed on storage devices.

Storage Policy Operations

The Storage Policy API is implemented with Web Services Description Language (WSDL), as an XML file describing how to communicate with it. The WSDL is usable with toolkit bindings by many programming languages, including Java.

SPBM Managed Objects

The Storage Policy API is structured like the vSphere API, and is likewise suitable for Java programming. Managed object types include the following:

- `PbmProfileProfileManager` supports operations on virtual machine storage profiles.
- `PbmPlacementSolver` identifies locations that support requested capabilities for storing virtual machine files.
- `PbmComplianceManager` verifies compliance of virtual machine and virtual disk requirement profiles.
- `PbmReplicationManager` deals with replication of virtual machine and virtual disk requirement profiles.
- `PbmServiceInstance` is the root object of Storage Policy service, created connecting to SPBM.

Storage Policy API and vSphere API

The following table shows the correspondence between SPBM managed objects and vSphere API methods.

Table 9: Storage Policy Operations and Virtual Machine Provisioning

SPBM Operation (Storage Policy API)	Virtual Machine Provisioning (vSphere API)
Use the **`PbmProfileProfileManager`** methods to create and update storage profiles.	Associate storage profiles with virtual machines and virtual disks. See the description of the vSphere API data object properties **`VirtualMachineConfigSpec.vmProfile`** and **`FileBackedVirtualDiskSpec.profile`** in the vSphere API Reference. You can also use the vSphere Web Client to associate a storage profile with a virtual machine or virtual disk.
Use the **`PbmPlacementSolver`** methods to identify candidate datastores for storage locations.	Specify the datastores when you create virtual machines and virtual disks. See the description of the vSphere API data object properties **`VirtualMachineFileInfo.vmPathName`** and **`VirtualDeviceFileBackingInfo.datastore`** in the vSphere API Reference.
Use the **`PbmComplianceManager`** methods to check compliance between storage requirements and capabilities.	After you associate a storage profile with a virtual machine or virtual disk, the Server will identify non-compliance if the datastore does not satisfy the requirements of the profile.

Access to the Storage Policy Server

You can access the VMware Storage Policy Server by connecting to a vCenter Server system and obtaining a vCenter session cookie.

The Storage Policy client API is described in the pbmService.wsdl file that is included in the VMware Storage Policy SDK. The API defines a set of request operations that you use to manipulate storage profiles. The VMware Storage Policy SDK includes Java bindings for the SPBM service WSDL.

To gain access to the Storage Policy Server, your client connects to a vCenter Server system and obtains the vCenter session cookie. Then you can use the vCenter session cookie to establish the connection with the Storage Policy Server. See Establish a Connection with the VMware Storage Policy Server.

After you establish a Storage Policy Server connection, your client uses language-specific Web Services access objects and the **`PbmServiceInstance`** and **`PbmServiceInstanceContent`** objects to access the Storage Policy managed objects and their methods.

The Storage Policy Web Services access objects are language-specific API binding objects that are generated from the Storage Policy WSDL. The VMware Storage Policy SDK contains JAXWS bindings to the Storage Policy API. The JAXWS bindings include the **`PbmService`** and **`PbmPortType`** Web Services access objects.

- **`PbmService`** – Provides access to the **`PbmPortType`** object and support for the Storage Policy Service connection.
- **`PbmPortType`** – Provides access to Storage Policy methods.

The following code fragment shows the sequence of calls that you use to obtain access to the Storage Policy API methods.

Access to Storage Policy API Methods

```
import com.vmware.pbm.PbmService;

import com.vmware.pbm.PbmPortType;

import com.vmware.pbm.PbmServiceInstanceContent;
```

```
[...]

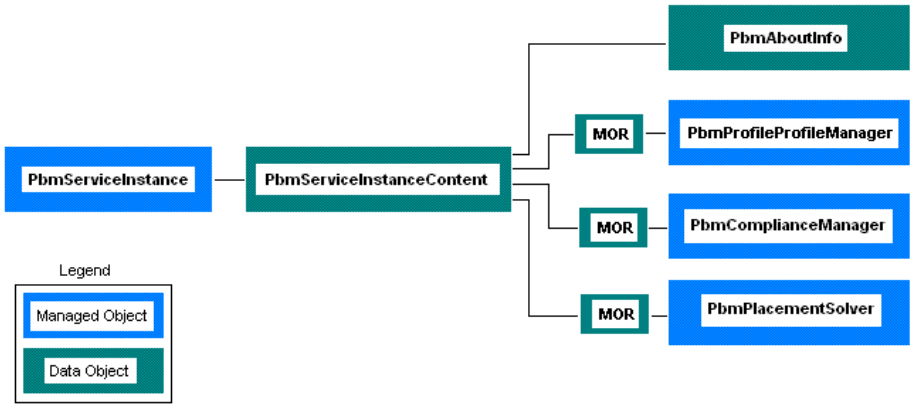
PbmService = new PbmService()

PbmPortType pbmPort = PbmService.getPbmPort()

PbmServiceInstanceContent pbmServiceContent = pbmPort.pbmRetrieveServiceContent
```

The following figure shows the resulting **`PbmServiceInstanceContent`** data object and the Storage Policy managed objects that provide access to Storage Policy services.

Figure 73: Storage Policy Service Instance Content



The **`PbmServiceInstanceContent`** object contains managed object references to the Storage Policy services. The set of Storage Policy services include the profile manager, placement solver, and compliance manager.

Service	Managed Object	Description
Profile Manager	`PbmProfileProfileManager`	Create and update VMware storage profiles to define storage requirements.
Placement Solver	`PbmPlacementSolver`	Identify candidate datastores for storage locations.
Compliance Manager	`PbmComplianceManager`	Check compliance between storage requirements and capabilities.

Storage Profile Queries

The Storage Policy API includes several methods that you can use to query for profiles and vSphere entities, such as datastores, virtual machines, and virtual disks.

The following table provides an overview of these methods. For more information, see the *`Storage Policy API Reference`*.

Table 10: Storage Profile API Query Methods

Method	Description
PbmQueryAssociatedEntities	Returns the virtual machine and disks that are associated with the given storage policies. With empty parameter, returns all virtual machine and disks that are associated with a storage policy.
PbmQueryAssociatedEntity	Returns references to entities associated with the specified profile.
PbmQueryAssociatedProfile	Returns profiles associated with the specified entity. The type of profile is determined by the type of entity that you specify. • If you specify a datastore, the method returns one or more capability (resource) profiles. • If you specify a virtual machine or virtual disk, the method returns one or more requirement profiles.
PbmQueryAssociatedProfiles	Returns **PbmQueryProfileResult** objects. Each result objet identifies an entity and one or more profiles. Profile type is determined by entity type. • If the entity is a datastore, the result object contains one or more capability (resource) profiles. • If the entity is a virtual machine or virtual disk, the result object contains one or more requirement profiles.
PbmQueryByRollupComplianceStatus	Returns all virtual machines for the given rollup compliance status.
PbmQueryDefaultRequirementProfile	Returns the default requirement profile ID for the given datastore. For legacy hubs, returns `null`.
PbmQueryDefaultRequirementProfiles	Returns the default profiles for the given datastores. For legacy datastores, the `defaultProfile` is set to null.
PbmQueryMatchingHub (deprecated)	Finds matching placement hubs for the specified requirements profile. Returns only hubs that match the profile. If this method is called for `VVolDefaultProfile`, then all VVol containers are returned as matching.
PbmQueryMatchingHubWithSpec (deprecated)	Finds matching placement hubs based on a profile creation specification. This method returns only those hubs that match the specification.
PbmQueryProfile	Returns requirement profiles or resource profiles, or both.
PbmQueryReplicationGroups	Returns identifiers for replication groups associated with virtual machines, virtual disks, or virtual machines and all their disks. If the query is performed for a virtual machine and all its disks are `virtualMachineAndDisks`, an entry per disk and one for the virtual machine configuration are returned.
PbmQuerySpaceStatsForStorageContainer	Retrieves space statistics of a datastore.

VMware Storage Policy SDK

The VMware Storage Policy SDK is distributed as part of the VMware vSphere Management SDK.

When you extract the contents of the distribution, the VMware Storage Policy SDK is located in the `spbm` sub-directory.

```
VMware-vSphere-SDK-build-num
     eam
     sms-sdk
     spbm
          docs
              java
                  JAXWS/samples/javadoc/index.html
              ReferenceGuide
                  index.html
          java
              JAXWS
                  lib
                  samples
                  build and run scripts
          wsdl
              pbmService.wsdl
              pbm.wsdl
     ssoclient
     vsphere-ws
```

The following table shows the locations of the contents of the VMware Storage Policy SDK.

Table 11: VMware Storage Policy SDK Contents

VMware Storage Policy SDK Component	Location
JAX-WS VMware Storage Policy client binding	`spbm/java/JAXWS/lib`
Java Storage Policy samples	`spbm/java/JAXWS/samples/com/vmware/spbm/samples/`
Java Storage Policy Server connection sample	`spbm/java/JAXWS/samples/com/vmware/spbm/connection/`
VMware Storage Policy API Reference	`spbm/docs/ReferenceGuide/index.html`
Documentation for example code	`spbm/docs/java/JAXWS/samples/javadoc/index.html`
WSDL files	`spbm/wsdl`

VMware Storage Policy SDK Examples

The VMware Storage Policy SDK contains Java examples that show how to create and use VMware storage policies.

This manual describes examples from the VMware Storage Policy SDK. It also describes examples from the vCenter Single Sign-On SDK that support the client connection to the Storage Policy Server. This manual includes the following single sign-on examples:

- vCenter Single Sign-On Client Example. This example shows how to obtain a holder-of-key token from the ESXivCenter Single Sign-On Server.
- vCenter LoginByToken Example. This example shows how to use the token to login to vCenter Server.

The following table lists the sample files in the VMware Storage Policy SDK.

Table 12: VMware Storage Profile SDK Sample File

Location	Examples	Description
SDK/spbm/java/JAXWS/samples/com/vmware/spbm/samples/		
	AboutInfo.java	Obtains identifying data about the Storage Policy Server.
	CheckCompliance.java	Checks the compliance of profiles associated with virtual machines and virtual disks.
	CreateProfile.java	Creates a requirement profile.
	CreateVSANProfile.java	Creates a new storage profile with one rule-set based on vSAN capabilities.
	DeleteProfile.java	Deletes a requirement profile.
	EditProfile.java	Adds or deletes subprofiles from a tag-based storage profile.
	FcdAssociateProfile.java	Attaches a first class disk (FCD) to a virtual machine and associates the given storage profile with the FCD.
	ListProfiles.java	Retrieves all of the storage profiles known to the system.
	ViewProfile.java	Prints the contents of a tag-based storage profile.
	VMClone.java	Deploys multiple instances of a virtual machine template to a datacenter. The clone specification has an associated storage profile.
	VMCreate.java	Creates a virtual machine. The virtual machine configuration specification has an associated storage profile.
	VMRelocate.java	Used to relocate a virtual machine's virtual disks to a datastore compliant with the given storage profile.
SDK/spbm/java/JAXWS/samples/com/vmware/spbm/connection/		
	BasicConnection.java	Establishes an authenticated session with a VMware SSO Server, vCenter Server, and Storage Policy Server.
	ConnectedServiceBase.java	Connection base class for client application implementations.
	Connection.java	Storage Policy sample support; utility class that sets up a Storage Policy Server connection.
	ConnectionException.java	Base exception class for exceptions thrown by connection classes.
	ConnectionMalformedUrlException.java	URL exception.

Table continued on next page

Continued from previous page

Location	Examples	Description
	KeepAlive.java	Keep-alive utility class; maintains the vCenter Server connection.
	VcSessionHandler.java	Utility class; inserts vCenter session cookie into SOAP header.

Storage Policy Server Connection

The connection between a Storage Policy client and the Storage Policy Server is based on the client's connection with a vCenter Server system.

About Storage Policy Server Sessions

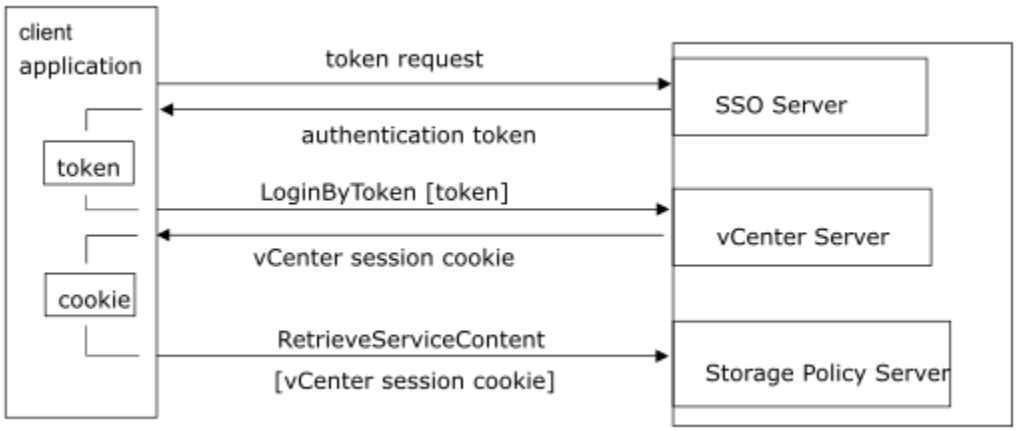
A vCenter Server client uses an HTTP session cookie to maintain a persistent connection with the Server. A Storage Policy client uses the vCenter Server session cookie to establish the connection with the Storage Policy Server.

A client performs the following operations to establish vCenter Server and Storage Policy Server sessions.

- Obtain a SAML token from the VMware SSO Server. See vCenter Single Sign-On Client Example.
- Use the SAML token to login to the vCenter Server. See vCenter LoginByToken Example.
- Use the **`RetrieveServiceContent`** method to send the session cookie to the Storage Policy Server and establish the connection with the Server.

The following figure shows a representation of the server connections and operations involved in establishing a Storage Policy Server connection.

Figure 74: Storage Policy Server Connection



Establish a Connection with the VMware Storage Policy Server

Use the session cookie from the vCenter Server session to establish the Storage Policy session. The session cookie represents the authenticated vCenter Server session, which is based on the SSO token.

The following code fragments establish connections both with the vCenter Server and the Storage Policy Server. These examples are based on the **`BasicConnection`** sample, which is located in the Storage Policy SDK connection sample directory.

`SDK/spbm/java/JAXWS/samples/com/vmware/spbm/connection/BasicConnection.java`

The **`BasicConnection`** sample uses an instance of the **`LoginByTokenSample`** class. See vCenter LoginByToken Example. The **`LoginByToken`** example saves the HTTP cookie produced during the intial connection sequence and then restores the cookie after the vCenter Server connection has been established. Although the **`LoginByToken`** example creates a vCenter Server connection, the **`BasicConnection`** sample establishes its own connection with the vCenter Server. A different implementation might integrate those capabilities to reduce the number of vCenter Server connections.

Server URLs for Basic Connection

The **`BasicConnection`** sample creates connections to three VMware Servers.

- SSO Server
- vCenter Server
- Storage Policy Server

In the example configuration, the SSO and Storage Policy Servers are located on the same system as the vCenter Server instance. In other configurations, the SSO Server might be located on a different server.

Table 13: VMware Server URLs

VMware Server	URL
vCenter Server	https://*server-name*|*IPaddress*/sdk/vimService
SSO Server	https://*server-name*|*IPaddress*/sts/STSService
Storage Policy Server	https://*server-name*|*IPaddress*/pbm

Create the Storage Policy Server Connection

The following code fragment uses a vCenter session cookie to create a Storage Policy Server session.

1. Extract the actual cookie value from the **`name=value`** expression in the cookie string obtained from the vCenter session connection.
2. Create a **`PbmService`** object.
3. Set up a header handler to support adding the vCenter session cookie to the Storage Policy Server connection.
4. Retrieve the **`PbmPort`** object for access to the Storage Policy API methods.
5. Retrieve the request context and set the endpoint to the Storage Policy Server URL.
6. Call the **`PbmRetrieveServiceContent`** method to establish the HTTP connection to the Storage Policy Server.

Storage Policy Server Connection

```
// 1. Set the extracted cookie in the PbmPortType

//

// Need to extract only the cookie value

String[] tokens = cookieVal.split(";");

tokens = tokens[0].split("=");

String extractedCookie = tokens[1];


// 2. Create a PbmService object.
```

```
pbmService = new PbmService();

// 3. Setting the header resolver for adding the VC session cookie to the
// requests for authentication
HeaderHandlerResolver headerResolver = new HeaderHandlerResolver();
headerResolver.addHandler(new VcSessionHandler(extractedCookie));
pbmService.setHandlerResolver(headerResolver);

// 4. Retrieve the PbmPort object for access to the Storage Policy API
pbmPort = pbmService.getPbmPort();

// 5. Set the Storage Policy Server endpoint
Map<String, Object> pbmCtxt = ((BindingProvider) pbmPort).getRequestContext();
pbmCtxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);
pbmCtxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, spbmurl.toString());

// 6. Retrieve the service content (creates the connection)
pbmServiceContent = pbmPort.pbmRetrieveServiceContent(getPbmServiceInstanceReference());
```

Establish the vCenter Session Connection for the Local Instance

The following code fragment sets up the HTTP connection with the vCenter Server instance.

1. Retrieve the **`VimPort`** interface. This provides access to the vSphere API methods.
2. Retrieve the request context and set the vCenter Server endpoint address in the request context.
3. Set the session cookie in the request context. The cookie (**`cookieVal`**) is obtained from the vCenter LoginByToken Example.
4. Call the **`RetrieveServiceContent`** method to establish the HTTP connection with the vCenter Server instance.

vCenter Server Connection

```
// 1. Retrieve the VimPort interface.
vimService = new VimService();
vimPort = vimService.getVimPort();

// 2. Retrieve the request context and set the vCenter Server endpoint.
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
```

```
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, vcurl.toString());
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);


// 3. Put the extracted vCenter session cookie into the VimPortType request header.
Map<String, List<String>> headers =
     (Map<String, List<String>>) ctxt.get(MessageContext.HTTP_REQUEST_HEADERS);
if (headers == null) {
     headers = new HashMap<String, List<String>>();
}
headers.put("Cookie", Arrays.asList(cookieVal));
ctxt.put(MessageContext.HTTP_REQUEST_HEADERS, headers);


// 4. Retrieve the vCenter Server service content. (Establishes the HTTP connection)
vimServiceContent = vimPort.retrieveServiceContent(this.getVimServiceInstanceReference());
```

vVol Based Storage Profiles

Virtual Volumes (vVols) help with granularity in shared storage by providing a way to designate a specific storage policy for each virtual machine or virtual disk. Vendors formulate metadata describing their storage arrays, advertise capability profiles in a VASA provider, so SPBM can query storage capabilities on request. VASA is an abbreviation of vSphere API for storage awareness.

vVol Storage Policy Architecture

Storage Policy Based Management (SPBM) helps vSphere and storage administrators automate the provisioning of virtual machines on shared vVol storage with desired data services.

Storage Management Service (SMS) and SPBM run as part of the Storage Policy Service (SPS), which is a daemon running in vCenter Server to handle storage subsystems. SPS communicates with vendor-provided VASA providers, which implement about a dozen SPBM related functions to advertise storage capabilities and manage storage requests.

On the storage array, a virtual machine comprises a Config vVol with descriptors, a Swap vVol for memory, and a Data vVol for each virtual disk. Data to and from a storage container flows through a protocol endpoint, which can support either SAN or NAS protocols. Snapshots, space efficiency, replication, and other storage features may be handled natively by the storage array.

Typically storage administration involves a set of rule sets where platinum or gold policy offers the best quality of service, silver policy in the middle, and bronze policy the fewest features. A vSphere administrator can configure these policies using the vSphere Client connected to vCenter Server. In vSphere there is a predefined No Requirements policy for vVols.

SPBM Calling Sequence for vVols

When managing vVol storage, SPBM uses a specific calling sequence.

After a VASA provider registers with vSphere, SPBM makes the following calls:

1. `queryCapabilityMetadata` to get the capability metadata exposed by the VASA provider, for example RAID level, compression, encryption, caching, or replication.
2. `queryResourceMetadata` to get a list of available storage resources, expressed as capability metadata.
3. `queryMatchingContainer` to get a list of compatible storage containers for a given policy. The response from the VASA provider refers to resources returned by the `queryResourceMetadata` call. There will also be provisioning requests before `queryComplianceResult` is called - not by SPBM, but from the ESXi host.
4. `queryComplianceResult` to check compliance with storage policies.
5. For version 3.0 VASA providers, `queryPolicy` is called if required to retrieve the Replication Group ID of the storage device for a vVol. ESXi hosts call it to query the policy of objects in the namespace so the policy can be applied to related objects.

SPBM Data Objects

SPBM data objects are numerous, and for replication can be complex, but the inheritance hierarchy is relatively flat. Some important objects are explained below

`CapabilityMetadata` represents the metadata for a single unique setting as defined by the VASA provider. A simple setting has one property, while a complex setting has more than one property.

`CapabilityMetadata` contains the `CapabilityId` (below), name and description of the capability, whether it is mandatory when creating a profile, whether the capability should affect placement and compliance, a key ID for the property, whether multiple constraints are allowed for a capability instance, and metadata for properties that comprise the capability. All contents but the first two and last are optional.

`CapabilityId` contains the unique identifier for a capability within its namespace, and the namespace to which the capability belongs.

`CapabilityInstance` contains the `CapabilityId` (above), an array of constraints on the properties that comprise the capability, and (as of VASA 3.0) the appropriate line of service for the capability.

`ConstraintInstance` contains the property constraints (in `PropertyInstance`) for a single occurrence of this capability. All properties must satisfy their respective constraints to be compliant. `PropertyInstance` includes the unique identifier for the property, and a value for its constraint.

The VASA provider is responsible for sending XML descriptions of capability metadata back to SMS. In the XML capability profiles, the <`subProfiles`> have a <`name`>, followed by a <`capability`> list, with each capability having an <`id`> for the <`namespace`> and a <`constraint`> for the <`propertyInstance`>.

`CapabilityMetadataPerCategory` contains an array of `CapabilityMetadata` (above) and the category to which the metadata belongs. The category is vendor-supplied; well-known terms are recommended.

`CapabilitySchema` describes a capability namespace, as reported by the VASA Provider in response to the `queryCapabilityMetadata` call. `CapabilitySchema` contains vendor information, namespace information, an array of `CapabilityMetadataPerCategory` (above), the line of service, and a schema ID string.

vSAN Based Storage Profiles

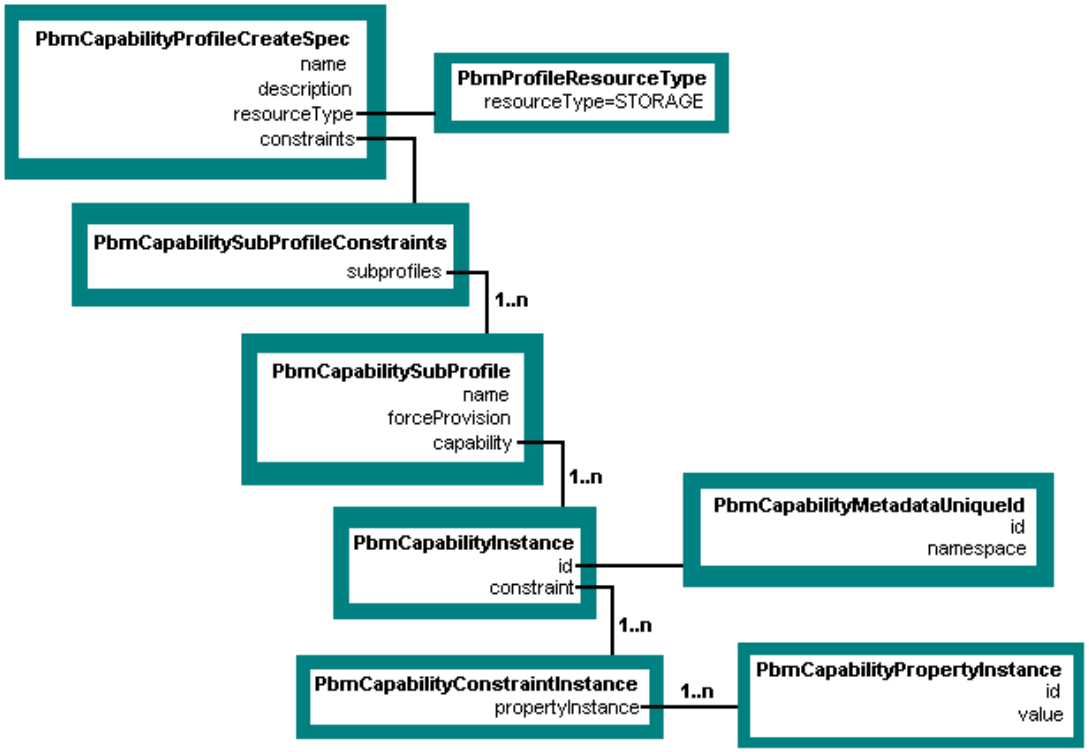
Storage requirements are based on storage capabilities available from storage provider, and vSphere offers vSAN storage capabilities. To create a requirements profile for vSAN capabilities, you retrieve metadata that describes vSAN capabilities and create a subprofile that expresses the storage requirements for virtual machine or virtual disk files. To perform these operations, you connect to the Storage Policy Server

Create a vSAN Requirements Profile

The following example demonstrates how to create a storage requirements profile based on vSphere vSAN storage capabilities. The example creates a requirement profile for vSAN stripe width.

The following figure shows the data objects used for a profile specification.

Figure 75: Storage Profile Specification



The following example is based on the Storage Policy SDK sample file **`CreateVSANProfile.java`**. This example is divided into two code fragments.

- Create an Individual Storage Requirement – The code fragment is a function that creates a single storage capability instance for a subprofile (rule).
- Create a Storage Profile – The code fragment builds a profile specification and creates the profile.

Create an Individual Storage Requirement

The following example builds a property instance for a capability. The property instance represents a single storage requirement.

The code performs the following steps.

1. Verifies that the capability exists.
2. Creates a property instance for the requirement (**`PbmCapabilityPropertyInstance`**).
3. Creates a capability constraint for the property instance (**`PbmCapabilityConstraintInstance`**).
4. Create a capability instance for the constraint and add the subprofile (rule) to the capability.

```
PbmCapabilityInstance buildCapability(String capabilityName, Object value,
```

```
         List<PbmCapabilityMetadataPerCategory> metadata)

         throws InvalidArgumentFaultMsg {


// Retrieve the metadata for the capability (stripeWidth)

PbmCapabilityMetadata capabilityMeta = PbmUtil.getCapabilityMeta(capabilityName,
metadata);

if (capabilityMeta == null)

throw new InvalidArgumentFaultMsg("Specified Capability does not exist", null);


// Create a New Property Instance based on the Stripe Width Capability

PbmCapabilityPropertyInstance prop = new PbmCapabilityPropertyInstance();

prop.setId(capabilityName);

prop.setValue(value);


// Associate Property Instance with a Rule (subprofile)

PbmCapabilityConstraintInstance rule = new PbmCapabilityConstraintInstance();

rule.getPropertyInstance().add(prop);


// Associate Rule (subprofile) with a Capability Instance

PbmCapabilityInstance capability = new PbmCapabilityInstance();

capability.setId(capabilityMeta.getId());

capability.getConstraint().add(rule);


return capability;

}
```

Create a Storage Profile

The example performs the following operations.

1. Retrieve a reference to the Storage Policy Profile Manger.
2. Verify that there is vSAN Storage Policy support.
3. Retrieve the vSAN storage capability metadata.
4. Add capabilities to be used as requirements.
5. Add the requirement capabilities to a subprofile. A subprofile corresponds to a rule set in the vSphere Web Client.
6. Specify the subprofile as capability constraints.
7. Build a profile specification.

8. Create the storage profile.

When you create a storage profile, the **PbmCreate** method returns a profile ID (**PbmProfileId**). The Profile Manager maintains a list of profiles. To obtain a profile from the list, use the **PbmQueryProfile** and **PbmRetrieveContent** methods. See Retrieve an Existing Storage Profile from the Server.

vSAN Storage Profile Creation

```
// 1: Get PBM Profile Manager & Associated Capability Metadata

spbmsc = connection.getPbmServiceContent();

ManagedObjectReference profileMgr = spbmsc.getProfileManager();


// 2: Verify that there is vSAN Storage Policy support

Boolean vSanCapabale = false;

List<PbmCapabilityVendorResourceTypeInfo> vendorInfo =

     connection.getPbmPort().pbmFetchVendorInfo(profileMgr, null);

for (PbmCapabilityVendorResourceTypeInfo vendor : vendorInfo)

     for (PbmCapabilityVendorNamespaceInfo vnsi : vendor.getVendorNamespaceInfo())

          if (vnsi.getNamespaceInfo().getNamespace().equals("vSan")) {

              vSanCapabale = true;

              break;

          }


if (!vSanCapabale)

     throw new RuntimeFaultFaultMsg( "Cannot create storage profile. vSAN Provider not
found.", null);


// 3: Get PBM Supported Capability Metadata

List<PbmCapabilityMetadataPerCategory> metadata =

     connection.getPbmPort().pbmFetchCapabilityMetadata(profileMgr,

             PbmUtil.getStorageResourceType(), null);

// 4: Add Provider Specific Capabilities

List<PbmCapabilityInstance> capabilities = new ArrayList<PbmCapabilityInstance>();

capabilities.add(buildCapability("stripeWidth", stripeWidth, metadata));


// 5: Add Capabilities to a RuleSet (subprofile)
```

```
PbmCapabilitySubProfile ruleSet = new PbmCapabilitySubProfile();

ruleSet.getCapability().addAll(capabilities);


// 6: Add Rule-Set (subprofile) to Capability Constraints

PbmCapabilitySubProfileConstraints constraints = new PbmCapabilitySubProfileConstraints();

ruleSet.setName("Rule-Set " + (constraints.getSubProfiles().size() + 1));

constraints.getSubProfiles().add(ruleSet);


// 7: Build Capability-Based Profile

PbmCapabilityProfileCreateSpec spec = new PbmCapabilityProfileCreateSpec();

spec.setName(profileName);

spec.setDescription("Storage Profile Created by SDK Samples. Rule based on vSAN
capability");

spec.setResourceType(PbmUtil.getStorageResourceType());

spec.setConstraints(constraints);


// 8: Create Storage Profile

PbmProfileId profile = connection.getPbmPort().pbmCreate(profileMgr, spec);

System.out.println("Profile " + profileName + " created with ID: " +
profile.getUniqueId());
```

Virtual Machine Storage Profiles

The Storage Policy Server maintains a list of storage profiles. To apply them, you retrieve an existing storage profile from the storage policy server, then associate it with a virtual machine. Code fragments in this chapter are based on sample program `VMCreate.java` in the Storage Policy SDK.

Retrieve an Existing Storage Profile from the Server

You can use a script to retrieve an existing storage profile from the storage policy server.

The following code fragment shows the example function **`getPbmProfileSpec`** that uses the **`PbmQueryProfile`** and **`PbmRetrieveContent`** methods to retrieve storage profiles. In the context of the Storage Policy SDK example `VMCreate.java`, the function returns a **`VirtualMachineDefinedProfileSpec`** to be used to configure storage for a virtual machine.

The function performs the following operations:

1. Uses the connection to the Storage Policy Server to retrieve a reference to the Profile Manager.
2. Calls the **`PbmQueryProfile`** method to obtain the list of storage profile identifiers.
3. Calls the **`PbmRetrieveContent`** method to obtain the list of storage profiles.

4. Finds the profile that matches the specified profile name.
5. Creates a **`VirtualMachineDefinedProfileSpec`** and assigns the identifier from the named profile to the **`VirtualMachineDefinedProfileSpec`**. You use the **`VirtualMachineDefinedProfileSpec`** when you configure the virtual machine. See Apply the Storage Profile to a Virtual Machine.

Retrieving a Storage Profile

```
    VirtualMachineDefinedProfileSpec getPbmProfileSpec(String name)
             throws InvalidArgumentFaultMsg, com.vmware.pbm.RuntimeFaultFaultMsg,
             RuntimeFaultFaultMsg {


// 1 Get PBM Profile Manager
        PbmServiceInstanceContent spbmsc = connection.getPbmServiceContent();
        ManagedObjectReference profileMgr = spbmsc.getProfileManager();


// 2 Retrieve the list of profile identifiers.
        List<PbmProfileId> profileIds =
            connection.getPbmPort().pbmQueryProfile(profileMgr,
                                                    PbmUtil.getStorageResourceType(),
                                                    null);


        if (profileIds == null || profileIds.isEmpty())
           throw new RuntimeFaultFaultMsg("No storage Profiles exist.", null);


// 3 Retrieve the list of storage profiles.
        List<PbmProfile> pbmProfiles =
            connection.getPbmPort().pbmRetrieveContent(profileMgr, profileIds);


// 4,5 Find the named profile and create a VirtualMachineDefinedProfileSpec
//    that will use the same profile identifier.
        for (PbmProfile pbmProfile : pbmProfiles) {
            if (pbmProfile.getName().equals(name)) {
                PbmCapabilityProfile profile = (PbmCapabilityProfile) pbmProfile;
                VirtualMachineDefinedProfileSpec spbmProfile =
                    new VirtualMachineDefinedProfileSpec();
```

```
                spbmProfile.setProfileId(profile.getProfileId().getUniqueId());


                return spbmProfile;
            }
        }


// Throw exception if none found
        throw new InvalidArgumentFaultMsg(
                "Specified storage profile name does not exist.", null);
    }
```

Apply the Storage Profile to a Virtual Machine

To use a storage profile for a virtual machine, specify a **`VirtualMachineDefinedProfileSpec`** object for the **`VirtualMachineConfigSpec.vmProfile`** property.

The following code fragment sets the storage profile and creates the virtual machine. The profile (**`spbmProfile`**) is a **`VirtualMachineDefinedProfileSpec`**. See Retrieve an Existing Storage Profile from the Server.

Associating a Storage Profile with a Virtual Machine

```
[...]

VirtualMachineConfigSpec configSpec = new VirtualMachineConfigSpec();

// Set SPBM profile

configSpec.getVmProfile().add(spbmProfile);

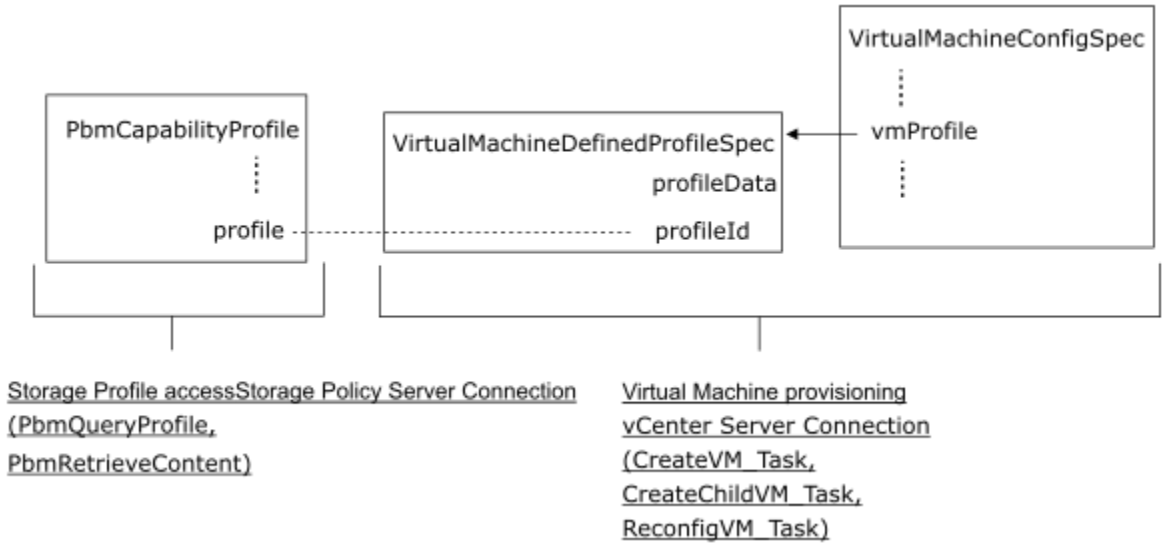
[...]

ManagedObjectReference taskmor =

     connection.getVimPort().createVMTask(vmFolderMor, vmConfigSpec, resourcepoolmor,
hostmor);
```

The following figure shows how a storage profile is integrated into a virtual machine configuration specification. Your client establishes the link between the storage profile (**`PbmCapabilityProfile`**) and the **`VirtualMachineDefinedProfileSpec`** by setting the **`profileId`** property in the **`VirtualMachineDefinedProfileSpec`**. The Server sets the **`profileData`** property when it configures the virtual machine.

Figure 76: Using a Storage Profile for Virtual Machine Provisioning



Tag-Based Storage Profiles

To use a tag-based storage profile, you assign a storage policy tag to a data center, define a storage requirement profile based on the tag, and associate the profile with a virtual machine.

Creating a Tag-Based Storage Profile

A storage profile specification has associated tag metadata, which you can use to create a storage profile.

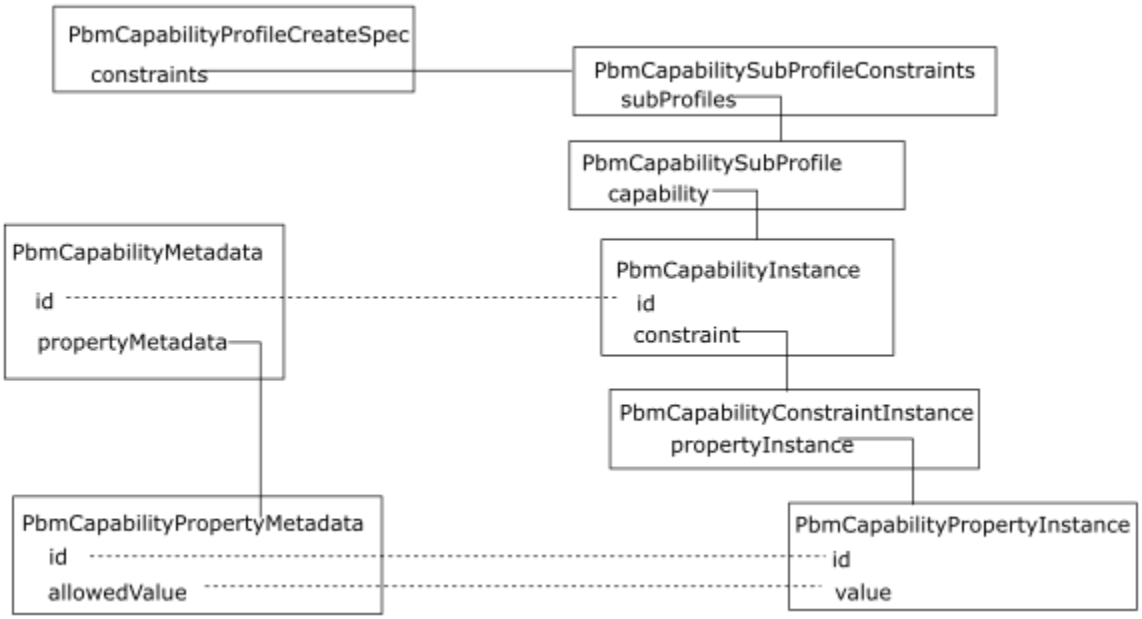
When you create the virtual machine, the vCenter Server instance will use the Storage Policy Server to resolve the tag reference in the profile and determine a datastore for virtual machine storage.

- To associate a storage policy tag with a datacenter, use the vSphere Web Client.
- To create a tag-based storage requirements profile, you retrieve metadata associated with a storage policy tag and create a storage profile that contains identifiers from the tag metadata.
- To associate the storage profile with a virtual machine, see Apply the Storage Profile to a Virtual Machine.

A tag-based storage profile involves the following metadata:

- The subprofile capability instance identifier (**`PbmCapabilityInstance.id`**) is set to the storage policy tag metadata identifier (**`PbmCapabilityMetadata.id`**).
- The capability property instance (**`PbmCapabilityPropertyInstance`**) specifies both an identifier and a value. Both properties are set to the tag metadata **`id`** and **`allowedValue`** properties.

Figure 77: Tag-Based Storage Profile Specification



The following example demonstrates how to create a storage requirements profile based on a storage policy tag. The example is divided into two sections.

- Retrieving Tag Metadata
- Creating a Storage Profile

Creating a Storage Profile

You can create a storage profile based on tag metadata.

The example performs the following operations.

1. Create a property instance with tags from the specified tag category.
2. Associate the property instance with a constraint (rule).
3. Associate the constraint with a capability instance.
4. Add the capability instance to a subprofile (rule set).
5. Add the subprofile to the list of subprofile constraints.
6. Build a profile specification.
7. Create the storage profile.

The following example is based on the Storage Policy SDK sample file **`CreateProfile.java`**.

Tag-Based Storage Profile Creation

```
// Get PBM Profile Manager and PBM Capability Metadata

spbmsc = connection.getPbmServiceContent();

ManagedObjectReference profileMgr = spbmsc.getProfileManager();
```

```
List<PbmCapabilityMetadataPerCategory> metadata =

     connection.getPbmPort().pbmFetchCapabilityMetadata(
profileMgr,PbmUtil.getStorageResourceType(), null);


// Step 1: Create Property Instance with tags from the specified Category

PbmCapabilityMetadata tagCategoryInfo = PbmUtil.getTagCategoryMeta(tagCategoryName,
metadata);


// Fetch Property Metadata of the Tag Category

List<PbmCapabilityPropertyMetadata> propMetaList = tagCategoryInfo.getPropertyMetadata();

PbmCapabilityPropertyMetadata propMeta = propMetaList.get(0);


// Create a New Property Instance based on the Tag Category ID

PbmCapabilityPropertyInstance prop = new PbmCapabilityPropertyInstance();

prop.setId(propMeta.getId());


// Fetch Allowed Tag Values Metadata; cast the xsd:any property (allowedValue) to a
discrete set

PbmCapabilityDiscreteSet tagSetMeta = (PbmCapabilityDiscreteSet)
propMeta.getAllowedValue();


// Create a New Discrete Set for holding Tag Values

PbmCapabilityDiscreteSet tagSet = new PbmCapabilityDiscreteSet();

for (Object obj : tagSetMeta.getValues()) {

   tagSet.getValues().add(((PbmCapabilityDescription) obj).getValue());

}

prop.setValue(tagSet);


// Step 2: Associate Property Instance with a Rule

PbmCapabilityConstraintInstance rule = new PbmCapabilityConstraintInstance();

rule.getPropertyInstance().add(prop);


// Step 3: Associate Rule with a Capability Instance

PbmCapabilityInstance capability = new PbmCapabilityInstance();
```

```
capability.setId(tagCategoryInfo.getId());

capability.getConstraint().add(rule);


// Step 4: Add Rule to a RuleSet

PbmCapabilitySubProfile ruleSet = new PbmCapabilitySubProfile();

ruleSet.getCapability().add(capability);


// Step 5: Add Rule-Set to Capability Constraints

PbmCapabilitySubProfileConstraints constraints = new PbmCapabilitySubProfileConstraints();

ruleSet.setName("Rule-Set " + (constraints.getSubProfiles().size() + 1));

constraints.getSubProfiles().add(ruleSet);


// Step 6: Build Capability-Based Profile

PbmCapabilityProfileCreateSpec spec = new PbmCapabilityProfileCreateSpec();

spec.setName(profileName);

spec.setDescription("Tag Based Storage Profile Created by SDK Samples. Rule based on tags
from Category "

            + tagCategoryName);

spec.setResourceType(PbmUtil.getStorageResourceType());

spec.setConstraints(constraints);


// Step 7: Create Storage Profile

PbmProfileId profile = connection.getPbmPort().pbmCreate(profileMgr, spec);
```

Retrieving Tag Metadata

You can retrieve tag metadata for a tag category.

The following example shows a code fragment that retrieves metadata for a tag category. Given the list of metadata obtained from the Storage Policy Server, the function traverses the list and returns the metadata associated with the specified category. This function is defined in the **`PbmUtil`** package in the Storage Policy SDK.

getTagCategoryMeta (PbmUtil Package)

```
public static PbmCapabilityMetadata getTagCategoryMeta(

     String tagCategoryName, List<PbmCapabilityMetadataPerCategory> schema) {

          for (PbmCapabilityMetadataPerCategory cat : schema)
```

```
            if (cat.getSubCategory().equals("tag"))
                for (PbmCapabilityMetadata cap : cat.getCapabilityMetadata())
                    if (cap.getId().getId().equals(tagCategoryName))
                       return cap;
        return null;
    }
```

Policy Rules

You can create custom storage policy strings to specify policy rules.

Create a Stand-Alone Virtual Disk with an Attached Storage Encryption Policy

The following example shows how to format an XML string to specify an I/O filter policy for a virtual disk.

- Verify that the name for the policy is user-friendly.
- Verify that the ID string to identify the policy is unique. The ID must be unique among the set of policies known to SPBM.

The format is specific to the current version of the VMware Web Services API. This format might not be compatible with other API versions.

1. Build the policy **<capability>** element, which specifies the filter.

```
<capability>
   <capabilityId>
      <id>vmwarevmcrypt@encryption</id>
      <namespace>IOFILTERS</namespace>
      <constraint></constraint>
   </capabilityId>
</capability>
```

2. Wrap the **<capability>** element singly in a **<subProfiles>** element, adding the rule name.

```
<subProfiles>
   <capability>
      <capabilityId>
         <id>vmwarevmcrypt@encryption</id>
         <namespace>IOFILTERS</namespace>
         <constraint></constraint>
      </capabilityId>
   </capability>
   <name>Rule-Set 1: IOFILTERS</name>
```

```
</subProfiles>
```

3. Wrap the **<subProfiles>** element singly in a **<constraints>** element.

```
<constraints>
  <subProfiles>
    <capability>
       <capabilityId>
          <id>vmwarevmcrypt@encryption</id>
          <namespace>IOFILTERS</namespace>
          <constraint></constraint>
       </capabilityId>
    </capability>
    <name>Rule-Set 1: IOFILTERS</name>
  </subProfiles>
</constraints>
```

4. Append profile metadata, such as name and creation date.

```
<constraints>
  <subProfiles>
    <capability>
      <capabilityId>
        <id>vmwarevmcrypt@encryption</id>
        <namespace>IOFILTERS</namespace>
        <constraint></constraint>
      </capabilityId>
    </capability>
    <name>Rule-Set 1: IOFILTERS</name>
  </subProfiles>
</constraints>
<createdBy>client</createdBy>
<creationTime>1999-12-31T23:59:59Z</creationTime>
<lastUpdatedTime>1999-12-31T23:59:59Z</lastUpdatedTime>
<generationId>1</generationId>
```

```
<name>IOfilter-Encrypt</name>
```

5. Wrap the **<constraints>** element and metadata in a **<storageProfile>** element, and prefix an XML header.

```
<?xml version='1.0' encoding='UTF-8'?>
<storageProfile xsi:type='StorageProfile'>
  <constraints>
    <subProfiles>
      <capability>
        <capabilityId>
          <id>vmwarevmcrypt@encryption</id>
          <namespace>IOFILTERS</namespace>
          <constraint></constraint>
        </capabilityId>
      </capability>
      <name>Rule-Set 1: IOFILTERS</name>
    </subProfiles>
  </constraints>
  <createdBy>client</createdBy>
  <creationTime>1999-12-31T23:59:59Z</creationTime>
  <lastUpdatedTime>1999-12-31T23:59:59Z</lastUpdatedTime>
  <generationId>1</generationId>
  <name>IOfilter-Encrypt</name>
  <profileId>I am Unique</profileId>
</storageProfile>
```

You can create a **ProfileSpec** to contain the XML data. See the *vSphere Web Services SDK Programming Guide*.

Legacy Storage Profiles

vSphere 5.0/5.1 systems support limited storage capability and requirement profiles.

In vSphere 5.5 and later, the Storage Policy Server supports more complex storage profiles. vSphere 5.5 and later upgrade legacy capability and requirement profiles for Storage Policy Server operations.

VASA 1.0 Storage Capability Upgrade

A Storage Policy Server can obtain storage capability data from VASA providers.

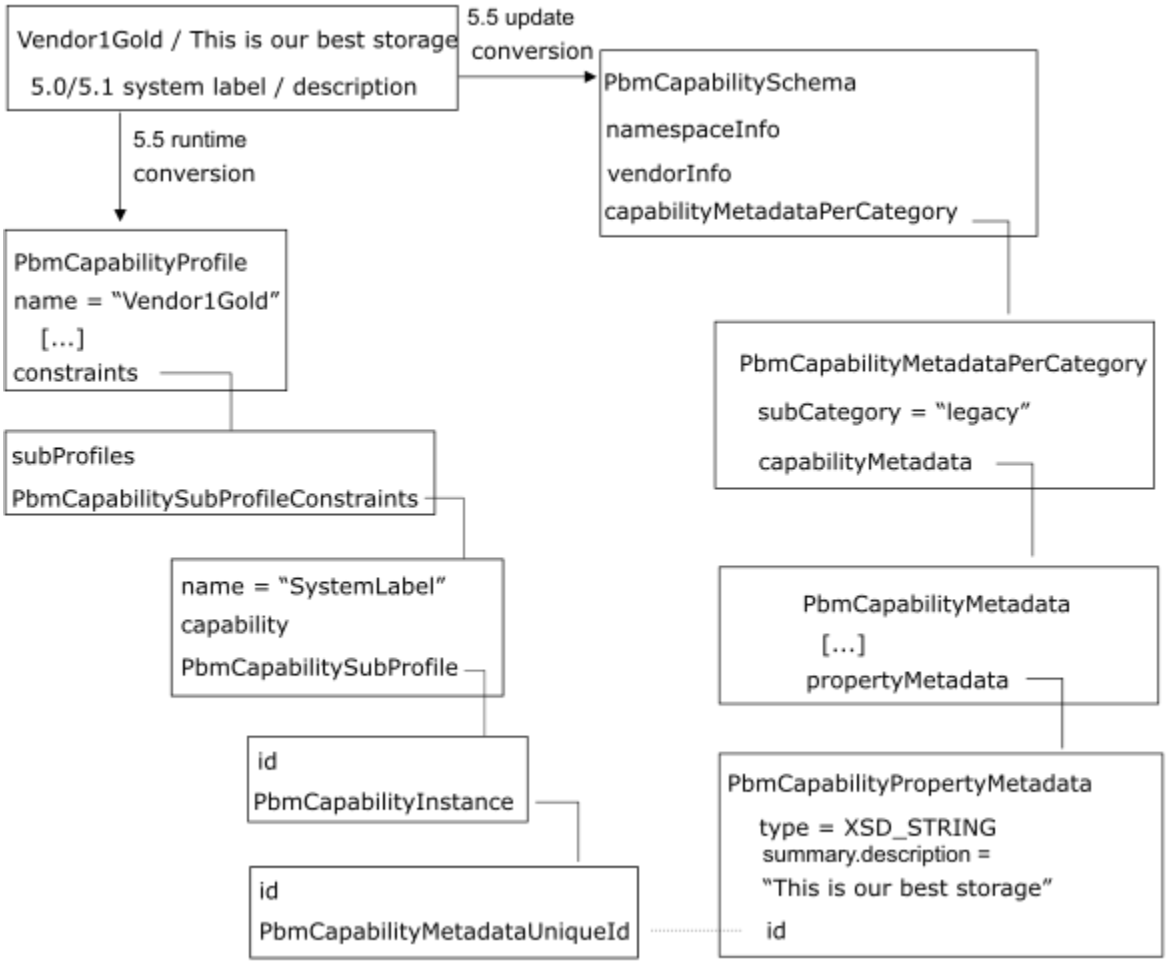
In vSphere 5.5, this generally implies VMware vSAN storage capabilities. A Storage Policy Server can also obtain capability data from a VASA provider that was implemented for the vSphere 5.0/5.1 environment.

The early architecture (vSphere 5.0/5.1) supports a simple expression of storage capability. A VASA 1.0 provider can advertise one system label per datastore. A system label has an associated description.

The Storage Policy Server performs a runtime conversion of VASA 1.0 system labels. The Storage Policy API presents the system label as a storage capability profile. The Server also generates a capability schema for the storage label. The generated storage capability profile references the generated schema.

The following figure shows the Storage Policy data objects that are generated from a vSphere 5.0/5.1 legacy profile.

Figure 78: Converted Legacy Capability Profile



vSphere Web Client User Label Conversion

A vSphere 5.0/5.1 storage profile can reference user labels displayed in the vSphere Web Client.

Users can associate a datastore with a user label. When you upgrade to vSphere 5.5, any existing 5.0/5.1 user labels will be converted into datastore tags.

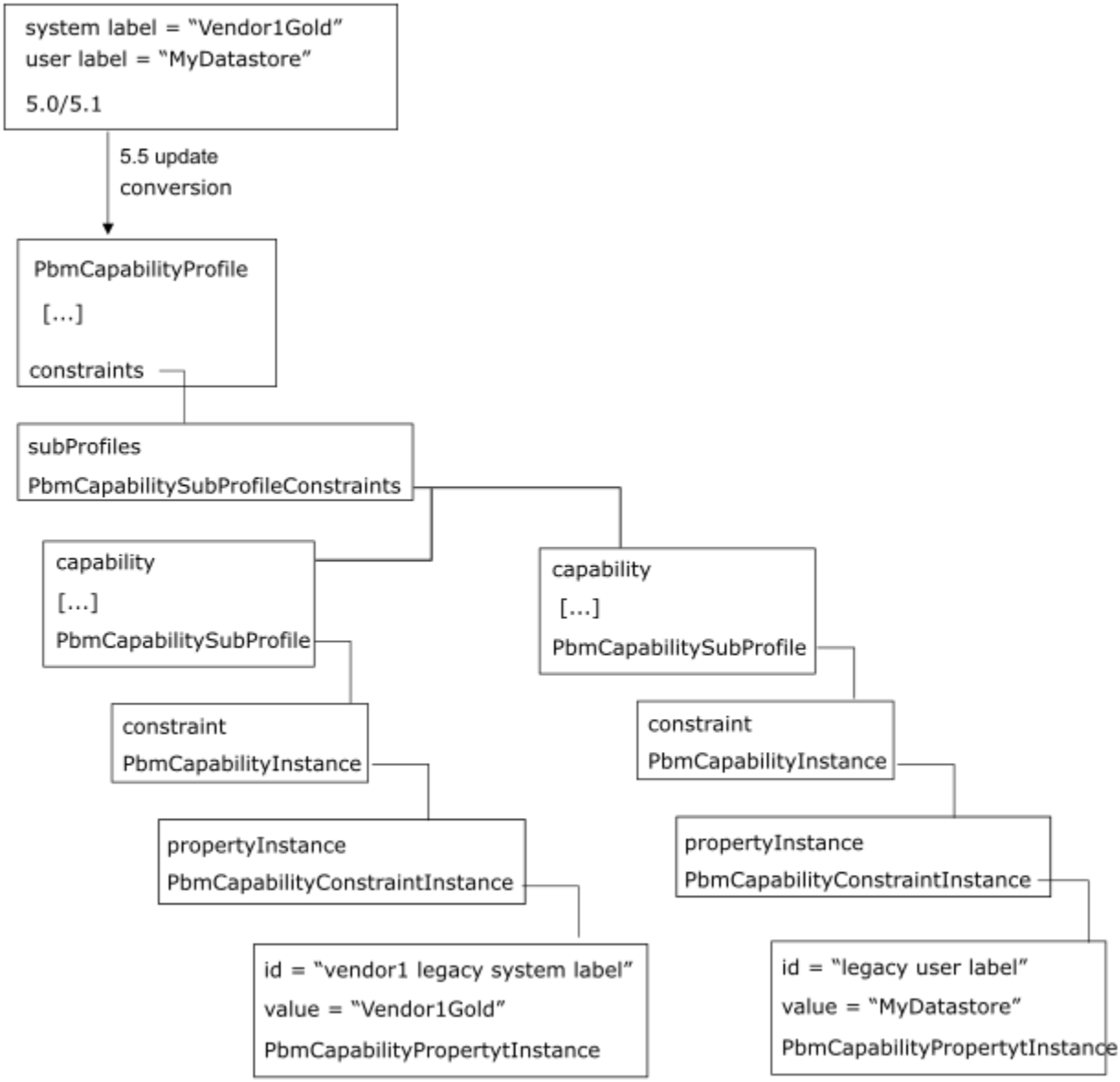
A converted policy profile contains one subprofile for each label referenced by the original vSphere 5.0/5.1 profile.

- A system label reference is converted to a reference to the appropriate vendor-specific “legacy system label” capability generated from that label.
- A user label reference is converted to a reference to the appropriate datastore tag generated from that label.

The following figure shows the conversion of a vSphere 5.0/5.1 profile that references a system label “Vendor1Gold” and a user label “MyDatastores”. When you upgrade to vSphere 5.5, the system converts the profile into Storage Policy API elements.

- The legacy system label identifies the original source as a VASA 1.0 provider.
- The legacy user label identifies the original source as a 5.0/5.1 user label in the vSphere Web Client.

Figure 79: Converted Legacy Requirement Profile



vCenter Single Sign-On Client Example

This chapter describes a Java example of acquiring a vCenter Single Sign-On security token.

vCenter Single Sign-On Token Request Overview

The code examples in the following sections show how to use the **`Issue`** method to acquire a holder-of-key security token.

To see an example of using the token to login to a vCenter Server, see vCenter LoginByToken Example. The code examples in this chapter are based on the following sample file located in the vCenter Single Sign-On SDK JAX-WS client `samples` directory.

```
.../JAXWS/samples/com/vmware/sso/client/samples/AcquireHoKTokenByUserCredentialSample.java
```

The **`AcquireHoKTokenByUserCredentialSample`** program creates a token request and calls the issue method to send the request to a vCenter Single Sign-On Server. The program uses a sample implementation of Web services message handlers to modify the SOAP security header for the request message.

This example uses the username-password security policy (**`STSSecPolicy_UserPwd`**). This policy requires that the SOAP security header include a timestamp, username and password, and a digital signature and certificate. The sample message handlers embed these elements in the message.

The example performs the following operations.

1. Create a security token service client object (**`STSService_Service`**). This object manages the vCenter Single Sign-On header handlers and it provides access to the vCenter Single Sign-On client API methods. This example uses the issue method.
2. Create a vCenter Single Sign-On header handler resolver object (**`HeaderHandlerResolver`**). This object acts as a container for the different handlers.
3. Add the handlers for timestamp, user credentials, certificate, and token extraction to the handler resolver.
4. Add the handler resolver to the security token service.
5. Retrieve the STS port (**`STS_Service`**) from the security token service object.
6. Create a security token request.
7. Set the request fields.
8. Set the endpoint in the request context. The endpoint identifies the vCenter Single Sign-On Server.
9. Call the issue method, passing the token request.
10. Handle the response from the vCenter Single Sign-On server.

Using Handler Methods for SOAP Headers

The VMware vCenter Single Sign-On SDK provides sample code that is an extension of the JAX-WS XML Web services message handler (**`javax.xml.ws.handler`**).

The sample code consists of a set of SOAP header handler methods and a header handler resolver, to which you add the handler methods. The handler methods insert timestamp, user credential, and message signature data into the SOAP security header for the request. A handler method extracts the SAML token from the vCenter Single Sign-On Server response.

The VMware vCenter Single Sign-On client SOAP header handler files are located in the `soaphandlers` directory.

```
SDK/sso/java/JAXWS/samples/com/vmware/sso/client/soaphandlers
```

To access the SOAP handler implementation, the example code contains the following import statements.

```
import com.vmware.sso.client.soaphandlers.HeaderHandlerResolver;

import com.vmware.sso.client.soaphandlers.SSOHeaderHandler;

import com.vmware.sso.client.soaphandlers.SamlTokenExtractionHandler

import com.vmware.sso.client.soaphandlers.TimeStampHandler;

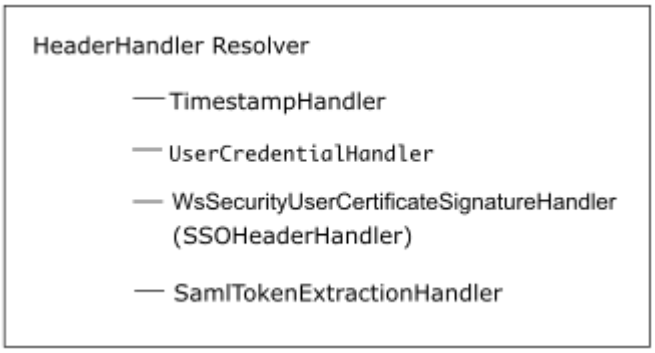
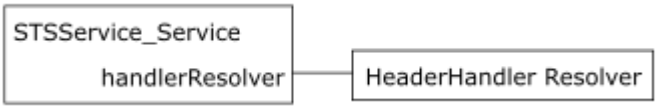
import com.vmware.sso.client.soaphandlers.UserCredentialHandler;

import com.vmware.sso.client.soaphandlers.WsSecurityUserCertificateSignatureHandler;
```

This example uses the following handler elements.

- **`HeaderHandlerResolver`**
- **`SamlTokenExtractionHandler`**
- **`TimestampHandler`**
- **`UserCredentialHandler`**
- **`WsSecurityUserCertificateSignatureHandler`** (**`SSOHeaderHandler`**)

The following sequence shows the operations and corresponding Java elements for message security.

Create an STS service object (**`STSService_Service`**). This object will bind the handlers to the request and provide access to the issue method.	STSService_Service
Create a handler resolver object (**`HeaderHandlerResolver`**). This object acts as a receptacle for the handlers.	HeaderHandlerResolver
Add the header handlers: • Timestamp – The handler will use system time to set the timestamp values. • User credential – The handler requires a username and a password; it will create a username token for the supplied values. • User certificate signature – The handler requires a private key and an x509 certificate. The handler will use the private key to sign the body of the SOAP message (the token request), and it will embed the certificate in the SOAP security header. • SAML token extraction – The handler extracts the SAML token directly from vCenter Single Sign-On Server response to avoid token modification by the JAX-WS bindings.	
Add the handler resolver to the STS service.	

The following code fragment creates a handler resolver and adds the handler methods to the handler resolver. After the handlers have been established, the client creates a token request and calls the **`Issue`** method. See Sending a Request for a Security Token.

IMPORTANT
You must perform these steps for message security before retrieving the STS service port. An example of retrieving the STS service port is shown in Sending a Request for a Security Token.

```
/*
 * Instantiate the STS Service
 */
STSService_Service stsService = new STSService_Service();

/*
 * Instantiate the HeaderHandlerResolver.
 */
HeaderHandlerResolver headerResolver = new HeaderHandlerResolver();

/*
 * Add handlers to insert a timestamp and username token into the SOAP security header
 * and sign the message.
 *
 * -- Timestamp contains the creation and expiration time for the request
 * -- UsernameToken contains the username/password
 * -- Sign the SOAP message using the combination of private key and user certificate.
 *
 * Add the TimeStampHandler
 */
headerResolver.addHandler(new TimeStampHandler());

/*
 * Add the UserCredentialHandler. arg[1] is the username; arg[2] is the password.
 */
UserCredentialHandler ucHandler = new UserCredentialHandler(args[1],args[2]);
headerResolver.addHandler(ucHandler);

/*
 * Add the message signature handler (WsSecurityUserCertificateSignatureHandler);
```

```
 * The client is responsible for supplying the private key and certificate.
 */
SSOHeaderHandler ssoHandler = new WsSecurityUserCertificateSignatureHandler(privateKey,
userCert);
headerResolver.addHandler(ssoHandler);


/*
 * Add the token extraction handler (SamlTokenExtractionHandler).
 */
SamlTokenExtractionHandler sbHandler = new SamlTokenExtractionHandler;
headerResolver.addHandler(sbHandler);


/*
 * Set the handlerResolver for the STSService to the HeaderHandlerResolver created above.
 */
stsService.setHandlerResolver(headerResolver);
```

Sending a Request for a Security Token

After setting up the SOAP header handlers, the example creates a token request and calls the **`Issue`** method.

The following sequence shows the operations and corresponding Java elements.

Retrieve the STS service port (**`STSService`**). The service port provides access to the vCenter Single Sign-On client API methods. The vCenter Single Sign-On handler resolver must be associated with the STS service before you retrieve the service port. See Using Handler Methods for SOAP Headers.	
Create a token request (**`RequestSecurityTokenType`**). Your vCenter Single Sign-On client will pass the token request to the **`Issue`** method. The **`Issue`** method will send the token request in the body of the SOAP message. This example sets the token request fields as appropriate for a holder-of-key token request.	RequestSecurityTokenType
Set the token request fields. • lifetime – Creation and expiration times. • token type – urn:oasis:names:tc:SAML:2.0:assertion • request type – http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue	RequestSecurityTokenType tokenType requestType

Table continued on next page

Continued from previous page

• key type – http://docs.oasis-open.org/ws-sx/ws-trust/200512/PublicKey (for holder-of-key token type) • signature algorithm – http://www.w3.org/2001/04/xmldsig-more#rsa-sha256 • renewable status	lifetime keyType signatureAlgorithm renewing
Set the endpoint address for the token request.	
Call the **`Issue`** method.	Issue (RequestSecurityTokenType) STSService
Handle the response from the vCenter Single Sign-On Server.	RequestSecurityTokenResponseType

The following example shows Java code that performs these operations.

Acquiring a vCenter Single Sign-On Token – Sending the Request

```
/*
 * Retrieve the STSServicePort from the STSService_Service object.
 */
STSService stsPort = stsService.getSTSServicePort();


/*
 * Create a token request object.
 */
RequestSecurityTokenType tokenType = new RequestSecurityTokenType();


/*
 *  Create a LifetimeType object.
 */
LifetimeType lifetime = new LifetimeType();


/*
 * Derive the token creation date and time.
 * Use a GregorianCalendar to establish the current time,
 * then use a DatatypeFactory to map the time data to XML.
 */
```

```
DatatypeFactory dtFactory = DatatypeFactory.newInstance();
GregorianCalendar cal = new GregorianCalendar(TimeZone.getTimeZone("GMT"));
XMLGregorianCalendar xmlCalendar = dtFactory.newXMLGregorianCalendar(cal);
AttributedDateTime created = new AttributedDateTime();
created.setValue(xmlCalendar.toXMLFormat());

/*
 * Specify a time interval for token expiration (specified in milliseconds).
 */
AttributedDateTime expires = new AttributedDateTime();
xmlCalendar.add(dtFactory.newDuration(30 * 60 * 1000));
expires.setValue(xmlCalendar.toXMLFormat());

/*
 * Set the created and expires fields in the lifetime object.
 */
lifetime.setCreated(created);
lifetime.setExpires(expires);

/*
 * Set the token request fields.
 */
tokenType.setTokenType("urn:oasis:names:tc:SAML:2.0:assertion");
tokenType.setRequestType("http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue");
tokenType.setLifetime(lifetime);
tokenType.setKeyType("http://docs.oasis-open.org/ws-sx/ws-trust/200512/PublicKey");
tokenType.setSignatureAlgorithm("http://www.w3.org/2001/04/xmldsig-more#rsa-sha256");

/*
 * Specify a token that can be renewed.
 */
RenewingType renewing = new RenewingType();
```

```
renewing.setAllow(Boolean.TRUE);
renewing.setOK(Boolean.FALSE); // WS-Trust Profile: MUST be set to false
tokenType.setRenewing(renewing);

/* Get the request context and set the endpoint address. */
Map<String, Object> reqContext = ((BindingProvider) stsPort).getRequestContext();
reqContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, args[0]);

/*
 * Use the STS port to invoke the "issue" method to acquire the token
 * from the vCenter Single Sign-On Server.
 */
RequestSecurityTokenResponseCollectionType issueResponse = stsPort.issue(tokenType);

/*
 * Handle the response - extract the SAML token from the response. The response type
 * contains the token type (SAML token type urn:oasis:names:tc:SAML:2.0:assertion).
 */
RequestSecurityTokenResponseType rstResponse =
issueResponse.getRequestSecurityTokenResponse();
RequestedSecurityTokenType requestedSecurityToken =
rstResponse.getRequestedSecurityToken();

/*
 * Extract the SAML token from the RequestedSecurityTokenType object.
 * The generic token type (Element) corresponds to the type required
 * for the SAML token handler that supports the call to LoginByToken.
 */
Element token = requestedSecurityToken.getAny();
```

vCenter LoginByToken Example

This chapter describes a Java example of using the **`LoginByToken`** method.

vCenter Server Single Sign-On Session

After you obtain a SAML token from the vCenter Single Sign-On Server, you can use the vSphere API method **`LoginByToken`** to establish a single sign-on session with a vCenter Server instance.

See vCenter Single Sign-On Client Example for an example of obtaining a vCenter Single Sign-On token.

At the beginning of a vCenter Single Sign-On session, your client is responsible for the following tasks:

- Maintain the vCenter session cookie. The vSphere architecture uses an HTTP cookie to support a persistent connection between a vSphere client and a vCenter Server instance. During the initial connection, the Server produces a session cookie. Operations during the login sequence will reset the request context so your client must save this cookie and re-introduce it at the appropriate times.
- Insert the vCenter Single Sign-On token and a timestamp into the SOAP header of the **`LoginByToken`** message.

The example program uses these general steps.

1. Call the **`RetrieveServiceContent`** method to establish an HTTP connection with the vCenter Server instance and save the HTTP session cookie. The client uses an HTTP header handler method to extract the cookie from the vCenter Server response.
2. Call the **`LoginByToken`** method to authenticate the vCenter session. To send the token to the vCenter Server instance, the client uses a handler to embed the token and a time stamp in the SOAP header for the message. To identify the session started with the **`RetrieveServiceContent`** method, the client uses a handler to embed the session cookie in the HTTP header.
3. Restore the session cookie.

HTTP and SOAP Header Handlers

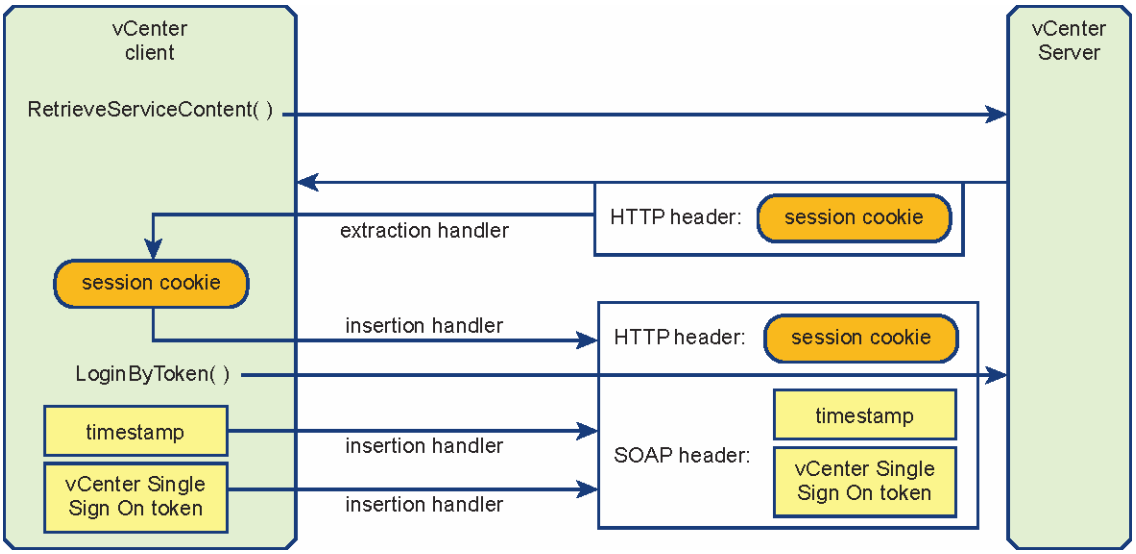
To use a vCenter Single Sign-On token to login to a vCenter Server instance, the example uses header handlers to manipulates the HTTP and SOAP header elements of the login request.

After establishing a handler, subsequent requests automatically invoke the handler.

- An extraction handler obtains the HTTP session cookie provided by the vCenter Server instance. After setting up the handler, a call to the **`RetrieveServiceContent`** method will invoke the handler to extract the cookie from the Server response.
- Insertion handlers put the vCenter Single Sign-On token and a timestamp into the SOAP header and the session cookie into the HTTP header of the login request.

The following figure shows the use of handlers to manipulate header elements when establishing a vCenter Single Sign-On session with a vCenter Server instance.

Figure 80: Starting a vCenter Session



IMPORTANT
Every call to the vCenter Server instance will invoke any message handlers that have been established. The overhead involved in using the SOAP and HTTP message handlers is not necessary after the session has been established. The example saves the default message handler before setting up the SOAP and HTTP handlers. After establishing the session, the example will reset the handler chain and restore the default handler.
The example code also uses multiple calls to the **`VimPortType.getVimPort method`** to manage the request context. The **`getVimPort`** method clears the HTTP request context. After each call to the **`getVimPort`** method, the client resets the request context endpoint address to the vCenter Server URL. After the client has obtained the session cookie, it will restore the cookie in subsequent requests.

LoginByToken Sample Code

The code examples in the following sections show how to use the **`LoginByToken`** method with a holder-of-key security token.

The code examples are based on the sample code contained in the vCenter Single Sign-On SDK. The files are located in the Java samples directory (`SDK/ssoclient/java/JAXWS/samples`).

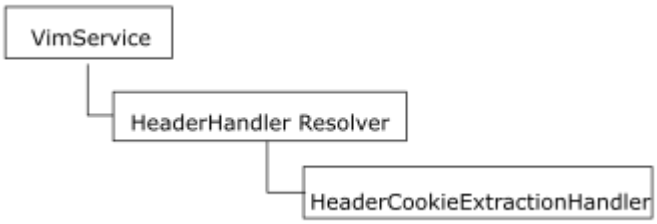
- LoginByToken sample:
  `samples/com/vmware/vsphere/samples/LoginByTokenSample.java`
- Header cookie handlers:
  `samples/com/vmware/vsphere/soaphandlers/HeaderCookieHandler.java`

  `samples/com/vmware/vsphere/soaphandlers/HeaderCookieExtractionHandler.java`
- SOAP header handlers. These are the same handlers that are used in vCenter LoginByToken Example. The SOAP handler files are located in the vCenter Single Sign-On client **`soaphandlers`** directory:
  `samples/com/vmware/sso/client/soaphandlers`

Saving the vCenter Server Session Cookie

The code fragment in this section establishes an HTTP session with the vCenter Server instance and saves the HTTP session cookie.

The following sequence describes these steps and shows the corresponding objects and methods.

Use the **`getHandlerResolver`** method to save the default message handler. To use the HTTP and SOAP message handlers, you must first save the default message handler so that you can restore it after login. The HTTP and SOAP message handlers impose overhead that is unnecessary after login.	VimService.getHandlerResolver( )
Set the cookie handler. The **`HeaderCookieExtractionHandler`** method retrieves the HTTP cookie.	
Get the VIM port. The VIM port provides access to the vSphere API methods, including the **`LoginByToken`** method.	
Set the request context endpoint address to the vCenter Server URL.	
Retrieve the **`ServiceContent`**. This method establishes the HTTP connection and sets the session cookie.	
Extract the cookie and save it for later use.	HeaderCookieExtractionHandler.getCookie ( )

The following example shows Java code that saves the session cookie.

Saving the vCenter Server Session Cookie

```
/*

 * The example uses a SAML token (obtained from a vCenter Single Sign-On Server)

 * and the vCenter Server URL.

 * The following declarations indicate the datatypes; the token datatype (Element)
corresponds

 * to the token datatype returned by the vCenter Single Sign-On Server.

 *

 * Element token;       -- from vCenter Single Sign-On Server

 * String vcServerUrl;  -- identifies vCenter Server

 *

 * First, save the default message handler.

 */
```

```
HandlerResolver defaultHandler = vimService.getHandlerResolver();

/*
 * Create a VIM service object.
 */
vimService = new VimService();

/*
 * Construct a managed object reference for the ServiceInstance.
 */
ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
SVC_INST_REF.setType("ServiceInstance");
SVC_INST_REF.setValue("ServiceInstance");

/*
 * Create a handler resolver.
 * Create a cookie extraction handler and add it to the handler resolver.
 * Set the VIM service handler resolver.
 */
HeaderCookieExtractionHandler cookieExtractor = new HeaderCookieExtractionHandler();
HeaderHandlerResolver handlerResolver = new HeaderHandlerResolver();
handlerResolver.addHandler(cookieExtractor);
vimService.setHandlerResolver(handlerResolver);

/*
 * Get the VIM port for access to vSphere API methods. This call clears the request
context.
 */
vimPort = vimService.getVimPort();

/*
 * Get the request context and set the connection endpoint.
 */
```

```
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, vcServerUrl);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

/*
 * Retrieve the ServiceContent. This call establishes the HTTP connection.
 */
serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);

/*
 * Save the HTTP cookie.
 */
String cookie = cookieExtractor.getCookie();
```

Using LoginByToken

The code fragment in this section sets up the message handlers and calls the **`LoginByToken`** method.

The following sequence describes the steps and shows the corresponding objects and methods.

Create a new **`HeaderHandlerResolver.`** Then set the message security handlers for cookie insertion and for inserting the SAML token and credentials in the SOAP header.	
Get the VIM port.	
Set the connection endpoint in the HTTP request context.	
Call the **`LoginByToken`** method. The method invocation executes the handlers to insert the elements into the message headers. The method authenticates the session referenced by the session cookie.	VimPortType.LoginByToken ( )

The following examples shows Java code that calls the **`LoginByToken`** method.

Using LoginByToken

```
/*
 * Create a handler resolver and add the handlers.
 */
HeaderHandlerResolver handlerResolver = new HeaderHandlerResolver();
handlerResolver.addHandler(new TimeStampHandler());
handlerResolver.addHandler(new SamlTokenHandler(token));
handlerResolver.addHandler(new HeaderCookieHandler(cookie));
handlerResolver.addHandler(new WsSecuritySignatureAssertionHandler(
                           userCert.getPrivateKey(),
                           userCert.getUserCert(),
                           Utils.getNodeProperty(token, "ID")));
vimService.setHandlerResolver(handlerResolver);

/*
 *  Get the Vim port; this call clears the request context.
 */
vimPort = vimService.getVimPort();

/*
 * Retrieve the request context and set the server URL.
 */
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, vcServerUrl);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

/*
 * Call LoginByToken.
 */
UserSession us = vimPort.loginByToken(serviceContent.getSessionManager(), null);
```

Restoring the vCenter Server Session Cookie

After you log in, you must restore the standard vCenter session context.

The code fragment in this section restores the default message handler and the session cookie. As the cookie handler has been replaced by the default handler, the client resets the session cookie by calling request context methods to access the context fields directly. The following sequence describes these steps and shows the corresponding objects and methods.

Restore the default message handler. The handlers used for **`LoginByToken`** are not used in subsequent calls to the vSphere API.	VimService.setHandlerResolver ( )
Get the VIM port.	
Set the connection endpoint in the HTTP request context.	
Set the HTTP request header (vCenter session cookie).	RequestContext.get () RequestContext.put ( )

The following example shows Java code that restores the vCenter session. This code requires the vCenter URL and the cookie and default handler that were retrieved before login. See LoginByToken Sample Code.

Restoring the vCenter Server Session

```
/*

 * Reset the default handler. This overwrites the existing handlers, effectively removing
them.

 */

vimService.setHandlerResolver(defaultHandler);

vimPort = vimService.getVimPort();


/*

 * Restore the connection endpoint in the request context.

 */

// Set the validated session cookie and set it in the header for once,

// JAXWS will maintain that cookie for all the subsequent requests


Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();

ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, vcServerUrl);
```

```
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);


/*
 * Reset the cookie in the request context.
 */
Map<String, List<String>> headers = (Map<String, List<String>>)
ctxt.get(MessageContext.HTTP_REQUEST_HEADERS);
if (headers == null) {
    headers = new HashMap<String, List<String>>();
}
headers.put("Cookie", Arrays.asList(cookie));
ctxt.put(MessageContext.HTTP_REQUEST_HEADERS, headers);
```

vCenter Single Sign-On Programming Guide

vCenter Single Sign-On Programming Guide describes how to use the VMware® vCenter Single Sign-On API.

VMware provides different APIs and SDKs for different applications and goals. The vCenter Single Sign-On SDK supports the development of vCenter clients that use SAML token authentication for access to vSphere environments.

Revision History

This book is revised with each release of the product or when necessary. A revised version can contain minor or major changes. The Revision History table summarizes the significant changes in each version of this book.

Table 14: Revision History

Revision Date	Description
11 OCT 2022	vSphere 8.0 release. • NET content removed.
06 OCT 2020	Update for the vSphere 7.0 Update 1 release.
02 APR 2020	Update for the vSphere 7.0 release.
17Apr2018	Updates for the vSphere 6.7 release.
15Nov2016	vSphere 6.5 release; vCenter Single Sign-On V2.0. Updated information about session creation sequence.
12Mar2015	vSphere 6.0 release; vCenter Single Sign-On V2.0. Updated Chapter 6, “LoginByToken Example (JAX-WS),” to reflect the change in session logic. Updated the STS Service specification in the SSO server URL.
19Sep2013	vSphere 5.5 release; vCenter Single Sign-On V2.0. Added documentation for C# (.NET) samples.
08Nov2012	vCenter Single Sign-On SDK V1.0 documentation update – changed SOAP envelope description to identify SSL/TLS (Transport Layer Security) correctly.
10Sep2012	vCenter Single Sign-On SDK V1.0 documentation.

Intended Audience

This book is intended for anyone who needs to develop applications using the vCenter Single Sign-On SDK. An understanding of Web Services technology and some programming background in one of the stub languages (Java or C#) is required.

Single Sign-On in the vSphere Environment

A vCenter Single Sign-On client connects to the vCenter Single Sign-On server to obtain a security token that contains authentication claims required for operations in the vSphere environment. The vCenter Single Sign-On client API supports operations to acquire, renew, and validate tokens.

vCenter Single Sign-On Overview

To support the requirements for secure software environments, software components require authorization to perform operations on behalf of a user. In a single sign-on environment, a user provides credentials once, and components in the environment perform operations based on the original authentication. vCenter Single Sign-On authentication can use the following identity store technologies:

- Windows Active Directory
- OpenLDAP (Lightweight Directory Access Protocol)

- Local user accounts (vCenter Single Sign-On server resident on the vCenter server machine)
- vCenter Single Sign-On user accounts

For information about configuring identity store support, see *vSphere Installation and Setup* and *vSphere Security* in the VMware Documentation Center.

In the context of single sign-on, the vSphere environment is a collection of services and solutions, each of which potentially requires authentication of clients that use the service or solution. Examples of solutions that might support single sign-on include vShield, SRM (Site Recovery Manager), and vCO (vCenter Orchestrator). Because a service can use another service, single sign-on provides a convenient mechanism to broker authentication during a sequence of vSphere operations.

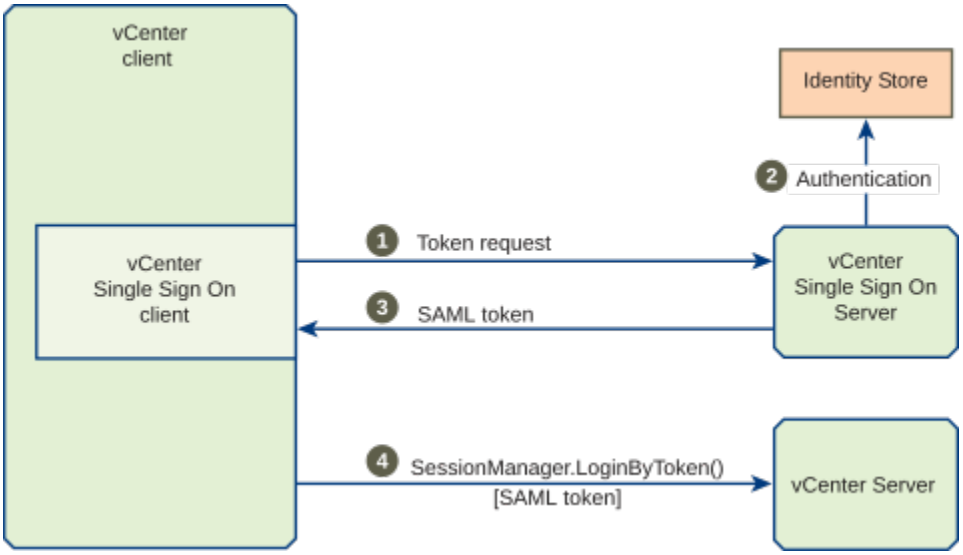
A vCenter Single Sign-On client connects to the vCenter Single Sign-On server to obtain a token that represents the client. The vCenter Single Sign-On server provides a Security Token Service (STS). A token uses the Security Assertion Markup Language (SAML), which is an XML encoding of authentication data. It contains a collection of statements or claims that support client authentication. Examples of token claims include name, key, and group.

vCenter Single Sign-On supports two types of tokens.

- Holder-of-key tokens provide authentication based on security artifacts embedded in the token. Holder-of-key tokens can be used for delegation. A client can obtain a holder-of-key token and delegate that token for use by another entity. The token contains the claims to identify the originator and the delegate. In the vSphere environment, a vCenter server obtains delegated tokens on a user's behalf and uses those tokens to perform operations.
- Bearer tokens provide authentication based only on possession of the token. Bearer tokens are intended for short-term, single-operation use. A bearer token does not verify the identity of the user (or entity) sending the request. It is possible to use bearer tokens in the vSphere environment, however there are potential limitations:
  - The vCenter Single Sign-On server may impose limitations on the token lifetime, which would require you to acquire new tokens frequently.
  - Future versions of vSphere might require the use of holder-of-key tokens.

The following figure shows a vCenter client that uses a SAML token to establish a session with a vCenter server.

Figure 81: Single Sign-On in the vSphere Environment – vCenter Server LoginByToken



The vCenter client also operates as a vCenter Single Sign-On client. The vCenter Single Sign-On client component handles communication with the vCenter Single Sign-On server.

1. The vCenter Single Sign-On client sends a token request to the vCenter Single Sign-On server. The request contains information that identifies the principal. The principal has an identity in the identity store. The principal may

be a user or it may be a software component. In this scenario, the principal is the user that controls the vCenter client.
2. The vCenter Single Sign-On server uses the identity store to authenticate the principal.
3. The vCenter Single Sign-On server sends a response to the token request. If authentication is successful, the response includes a SAML token.
4. The vCenter client connects to the vCenter server and calls the **SessionManager**.**LoginByToken** method. The login request contains the SAML token.

The previous figure shows the vCenter server, vCenter Single Sign-On server, and identity store as components running on separate machines. You can use different vCenter Single Sign-On configurations.

- A vCenter Single Sign-On server can operate as an independent component running on its own machine. The vCenter Single Sign-On server can use a remote identity store or it can manage user accounts in its own internal identity store.
- A vCenter Single Sign-On server can operate as an embedded component running on the vCenter server machine. In this configuration, the vCenter Single Sign-On server can use a remote identity store, its own internal identity store, or it can access user accounts on the vCenter server machine.

For information about installing and configuring the vCenter Single Sign-On server, see *vSphere Installation and Setup* and *vSphere Security* in the VMware Documentation Center.

vCenter Single Sign-On Client API

The vCenter Single Sign-On client API is described in the WSDL (Web Service Definition Language) file that is included in the vCenter Single Sign-On SDK. This API defines a set of request operations that correspond to the WS-Trust 1.4 bindings. The set of operations includes **Issue**, **Renew**, **Validate**, and **Challenge** requests.

- **Issue** – Obtains a token from a vCenter Single Sign-On server.
- **Renew** – Renews an existing token.
- **Validate** – Validates an existing token.
- **Challenge** – Part of a negotiation with a vCenter Single Sign-On server to obtain a token.

The vCenter Single Sign-On SDK includes Java bindings for the vCenter Single Sign-On WSDL. The SDK also contains sample code that demonstrates client-side support for the WS-SecurityPolicy standard. Security policies specify the elements that provide SOAP message security. To secure SOAP messages, a client inserts digital signatures, certificates, and SAML tokens into the SOAP headers for vCenter Single Sign-On requests.

The Java sample includes a JAX-WS implementation of SOAP header methods that support the vCenter Single Sign-On security policies.

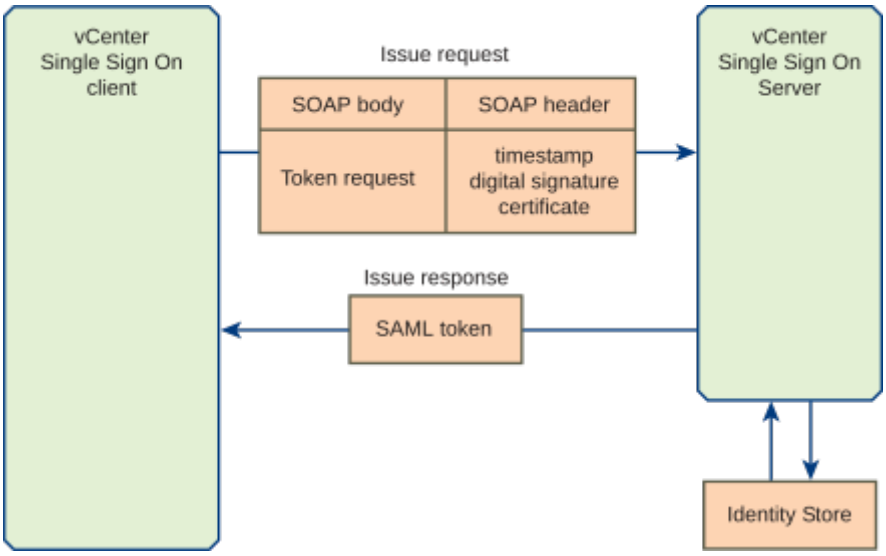
See vCenter Single Sign-On Security Policies and vCenter Single Sign-On SDK.

Acquiring a SAML Token from a vCenter Single Sign-On Server

To obtain a security token from a vCenter Single Sign-On server, the vCenter Single Sign-On client calls the **Issue** method, which sends a SOAP message that contains a token request and authentication data. This section describes a token request that uses a certificate to obtain a holder-of-key token. When the client creates the token request, it also inserts timestamp, signature, and certificate data into the SOAP security header.

The following figure represents the content of an **Issue** request and the response containing a SAML token.

Figure 82: Issue - vCenter Single Sign-On Token Request and Response



The vCenter Single Sign-On SDK provides Java packages that support SOAP header manipulation.

When the vCenter Single Sign-On server receives the issue request, it performs the following operations to generate a token:

- Uses the timestamp to validate the request.
- Validates the certificate.
- Uses the certificate to validate the digital signature.
- Uses the certificate subject to authenticate the request. Authentication is obtained from the identity store that is registered with the vCenter Single Sign-On server.
- Generates a token that specifies the principal – the vCenter Single Sign-On client – as the token subject.

vCenter Single Sign-On Security Policies

Web service security policies define the requirements for secure communication between a Web service and a client. vCenter Single Sign-On security policies are based on the WS-Policy framework and WS-SecurityPolicy specifications. A policy identifies specific elements for token requests. Based on the policy requirements, a vCenter Single Sign-On client will insert data into the SOAP security header for the token request.

vCenter Single Sign-On defines security policies for end user access, solution access, and for token exchange. The policies stipulate the following elements:

- Security certificates (x509V3, x509PKIPathV1, x509PKCS7, or WssSamlV20Token11)
- Message timestamps
- Security binding (transport)
- Encryption algorithm (Basic256Sha256)

vCenter Single Sign-On security policies specify that the body of the SOAP message for a holder-of-key token must be signed. Bearer tokens require only the username and timestamp tokens.

> **NOTE**
> The vCenter Single Sign-On server issues SAML tokens to represent client authentication. The standards documentation also uses the term “token” to refer to claims and certificate data that is inserted into SOAP security headers.

The following table shows the vCenter Single Sign-On policies and identifies the requirements for each policy. The vCenter Single Sign-On WSDL defines these policies for use with the vCenter Single Sign-On methods.

Table 15: vCenter Single Sign-On Policies

Policy	Description
STSSecPolicy	Defines the transport policy and algorithm suite for all communication with the vCenter Single Sign-On server: • Certificate-based server-side SSL authentication. • HTTPS transport binding using NIST (National Institute of Standards and Technology) Basic256Sha256 encryption algorithm. The HTTPS token is used to generate the message signature. • Request security header must contain a timestamp.
IssueRequestPolicy	Defines the security policy for Issue token requests. IssueRequestPolicy specifies either username token (signed), username token (plaintext password), X509 certificate, or holder-of-key token authentication. You specify username/password or X509 certificate credentials to obtain a vCenter Single Sign-On token. If you obtain a holder-of-key token, you can use that token for subsequent Issue requests. Username token (signed) authentication: • X509 endorsing supporting token (WssX509V3Token11, WssX509PkiPathV1Token11, or WssX509Pkcs7Token10) • WssUsernameToken11 signed supporting token Username token (plaintext password) authentication: • WssUsernameToken11 signed supporting token X509 certificate authentication: • X509 endorsing supporting token (WssX509V3Token11, WssX509PkiPathV1Token11, or WssX509Pkcs7Token10) Holder-of-Key token authentication: • WssSamlV20Token11 assertion referenced by a KeyIdentifier • Token must be used to sign the SOAP message body.
RenewRequestPolicy	Defines the security policy for Renew token requests. The request must contain one of the following endorsing supporting tokens. The SOAP message body must be included in the signature generated with the token. • WssX509V3Token11 • WssX509PkiPathV1Token11 • WssX509Pkcs7Token10

vCenter Single Sign-On SDK Support for vCenter Single Sign-On Security Policies

The vCenter Single Sign-On SDK provides Java utilities that support the vCenter Single Sign-On security policies. Your vCenter Single Sign-On client can use these utilities to create digital signatures and supporting tokens, and insert them into SOAP headers as required by the policies. The SOAP header utilities are defined in files that are located in the samples directory:

```
SDK\sso\java\JAXWS\samples\com\vmware\sso\client\soaphandlers
```

See Using Handler Methods for SOAP Headers.

Connecting to a vCenter Single Sign-On Server

When a vCenter Single Sign-On client connects to a vCenter Single Sign-On server, it must specify the server URL as the endpoint for the token request message. The endpoint specification uses the following format:

`https://hostname|IPaddress/STS/STSService`

The path suffix (`STS/STSService`) is required. See Sending a Request for a Security Token for an example of setting the endpoint for a token request.

vCenter Single Sign-On Token Delegation

Holder-of-key tokens can be delegated to services in the vSphere environment. A service that uses a delegated token performs the service on behalf of the principle that provided the token. A token request specifies a **`DelegateTo`** identity. The **`DelegateTo`** value can either be a solution token or a reference to a solution token.

Components in the vSphere environment can use delegated tokens. vSphere clients that use the **`LoginByToken`** method to connect to a vCenter server do not use delegated tokens. The vCenter server will use a vSphere client's token to obtain a delegated token. The vCenter server will use the delegated token to perform operations on behalf of the user after the user's vCenter session has ended. For example, a user may schedule operations to occur over an extended period of time. The vCenter server will use a delegated token to support these operations.

vCenter Single Sign-On Token Lifetime - Clock Tolerance

A SAML token contains information about the lifetime of a token. A SAML token uses the **`NotBefore`** and **`NotOnOrAfter`** attributes of the SAML **`Conditions`** element to define the token lifetime.

<saml2:Conditions NotBefore="2011-10-04T21:39:17.731Z" NotOnOrAfter="2011-10-04T21:39:47.731Z">

During a token's lifetime, the vCenter Single Sign-On server considers any request containing that token to be valid and the server will perform renewal and validation operations on the token. The lifetime of a token is affected by a clock tolerance value that the vCenter Single Sign-On server applies to token requests. The clock tolerance value accounts for differences between time values generated by different systems in the vSphere environment. The clock tolerance is 10 minutes.

vCenter Single Sign-On Challenge (SSPI)

The vCenter Single Sign-On server supports the use of SSPI (Security Support Provider Interface) for client authentication. SSPI authentication requires that both the client and server use security providers to perform authentication.

At the beginning of a vCenter Single Sign-On server session, the vCenter Single Sign-On client and vCenter Single Sign-On server exchange data. Each participant will use its security provider to authenticate the data it receives. The authentication exchange continues until both security providers authenticate the data.

The vCenter Single Sign-On client API provides a **`challenge`** request for client participation in SSPI authentication. The following sequence describes the challenge protocol.

1. vCenter Single Sign-On client sends an **`issue`** request to the vCenter Single Sign-On server. The request contains the client credentials.
2. vCenter Single Sign-On server uses its security provider to authenticate the client. The server returns a `RequestSecurityTokenResponseType` object in response to the **`issue`** request. The response contains a challenge.
3. vCenter Single Sign-On client uses its security provider to authenticate the vCenter Single Sign-On server response. To continue the authentication exchange, the client sends a **`challenge`** request to the vCenter Single

Sign-On server. The request contains the resolution to the server's challenge and it can also contain a challenge from the vCenter Single Sign-On client.

4. vCenter Single Sign-On server uses its security provider to authenticate the client's response. If there are still problems, the server can continue the authentication exchange by returning a response with an embedded challenge. If authentication is successful, the vCenter Single Sign-On server returns a SAML token to complete the original **`issue`** request.

To exchange challenge data, the vCenter single Sign-On client and vCenter Single Sign-On server use the following elements defined for both `RequestSecurityTokenType` and `RequestSecurityTokenResponseType` objects.

- Context attribute
- BinaryExchange element

vCenter Single Sign-On SOAP Message Structure

The requirements listed in the following table apply to the SOAP message structure in vCenter Single Sign-On message exchange.

Table 16: vCenter Single Sign-On SOAP Message Structure

Element	Message Requirements
SOAP envelope	All <wst:RequestSecurityToken>, <wst:RequestSecurityTokenResponse>, and <wst:RequestSecurityTokenResponseCollection> elements must be sent as the single direct child of the body of a SOAP 1.1 <S11:Envelope> element.
	Use HTTP POST to send all vCenter Single Sign-On SOAP messages over an SSL/TLS-protected channel. Set the SOAPAction HTTP header field to the appropriate message binding.
	The <wsse:Security> header in a vCenter Single Sign-On request must contain a <wsu:Timestamp> element.
SOAP message signature	If a signature is applied to a request then it must include: • Either the <S11:Body>, or the WS-Trust element as a direct child of the <S11:Body> • The <wsu:Timestamp>, if present, in the <S11:Header>.
	Exclusive canonicalization without comments (xml-exc-c14n) must be used prior to signature generation.
	The signature certificate must either be carried either within a <wsse:BinarySecurityToken> or a <saml:Assertion> within <wsse:Security> header of the <S11:Header>.
	The signature must contain a <wsse:SecurityTokenReference> that uses an internal direct reference to the <wsse:BinarySecurityToken>.

vCenter Single Sign-On SDK

The vCenter Single Sign-On SDK is distributed as part of the VMware vSphere Management SDK. When you extract the contents of the distribution kit, the vCenter Single Sign-On SDK is located in the **`ssoclient`** sub-directory:

```
VMware-vSphere-SDK-build-num
      eam
      sms-sdk
      ssoclient
           docs
```

```
dotnet
    cs
       samples
java
    JAXWS
       lib
       samples
wsdl
vsphere-ws
```

The following table shows the locations of the contents of the vCenter Single Sign-On SDK.

Table 17: vCenter Single Sign-On SDK Contents

vCenter Single Sign-On SDK Component	Location
C# vCenter Single Sign-On samples	ssoclient/dotnet/cs/samples
JAX-WS vCenter Single Sign-On client binding	ssoclient/java/JAXWS/lib
Java samples	ssoclient/java/JAXWS/samples/com/vmware/sso/client/samples
VMware SOAP header utilities	ssoclient/java/JAXWS/samples/com/vmware/sso/client/soaphandlers
General utilities for samples	ssoclient/java/JAXWS/samples/com/vmware/sso/client/utils
WS-Security utilities for samples	ssoclient/java/JAXWS/samples/com/vmware/sso/client/wssecurity
vCenter LoginByToken sample	ssoclient/java/JAXWS/samples/com/vmware/vsphere/samples
VMware SOAP header utilties for LoginByToken example	ssoclient/java/JAXWS/samples/com/vmware/vsphere/soaphandlers
Documentation for example code	ssoclient/docs/java/JAXWS/samples/javadoc/index.html
WSDL files	ssoclient/wsdl

vCenter Single Sign-On SDK Examples

The vCenter Single Sign-On SDK contains Java examples that show how to acquire, validate, and renew tokens. This manual describes examples that show how to obtain a holder-of-key token from a vCenter Single Sign-On server and how to use that token to login to a vCenter server.

This manual describes two of the Java examples provided by the VMware SSO Client SDK:

- vCenter Single Sign-On Client Example (JAX-WS). This example shows how to obtain a holder-of-key token from the vCenter Single Sign-On server.
- LoginByToken Example (JAX-WS). This example shows how to use the token to login to vCenter server.

The following table lists the sample files in the SDK:

Table 18: VMware SSO Client SDK Sample Files – Java

Location	Examples	Description
SDK/ssoclient/java/JAXWS/samples/com/vmware/sso/client/samples/		

Table continued on next page

Continued from previous page

Location	Examples	Description
	AcquireBearerTokenByUserCredentialSample.java	Demonstrates how to use username and password credentials to obtain a bearer token.
	AcquireHoKTokenByHoKTokenSample.java	Demonstrates how to exchange one holder-of-key token for another.
	AcquireHoKTokenBySolutionCertificateSample.java	Demonstrates how a solution uses its private key and certificate to acquire a holder-of-key token.
	AcquireHoKTokenByUserCredentialSample.java	Demonstrates how to use username, password, and certificate credentials to obtain a holder-of-key token. See vCenter Single Sign-On Client Example (JAX-WS).
	RenewTokenSample.java	Demonstrates how to renew a holder-of-key token.
	ValidateTokenSample.java	Demonstrates how to validate a token.
`SDK/ssoclient/java/JAXWS/samples/com/vmware/sso/client/soaphandlers/`		
	HeaderHandlerResolver.java	Provides methods to manage the set of header handlers.
	SamlTokenExtractionHandler.java	Extracts a SAML token from the vCenter Single Sign-On server response.
	SamlTokenHandler.java	Adds a SAML token to a SOAP security header.
	SSOHeaderhandler.java	Base class for header handler classes.
	TimeStampHandler.java	Adds a timestamp element to a SOAP security header.
	UserCredentialHandler.java	Adds a username token to a SOAP security header.
	WsSecuritySignatureAssertionHandler.java	Uses SAML token assertion ID, private key, and certificate to sign a SOAP message. For use when using an existing token to acquire a new token.
	WsSecurityUserCertificateSignatureHandler.java	Uses a private key and certificate to sign a SOAP message.
`SDK/ssoclient/java/JAXWS/samples/com/vmware/vsphere/samples/`		
	LoginByTokenSample.java	Demonstrates how to use a SAML token to login to a vCenter server. See LoginByToken Example (JAX-WS).
`SDK/ssoclient/java/JAXWS/samples/com/vmware/vsphere/soaphandlers/`		
	HeaderCookieExtractionHandler.java	Extracts the vCenter HTTP session cookie from the response to a connection request.
	HeaderCookieHandler.java	Inserts an HTTP cookie into a request.

vCenter Single Sign-On API Reference

This chapter contains descriptions of the methods and data structures defined for the vCenter Single Sign-On client API.

vCenter Single Sign-On Client API Methods

The vCenter Single Sign-On client API consists of the following methods:

- `Issue`
- `Renew`
- `Validate`
- `Challenge`

Issue

Sends a security token request to a vCenter Single Sign-On server.

Method Signature

```
Issue (requestSecurityToken : RequestSecurityTokenType)

           returns RequestSecurityTokenResponseCollectionType
```

Parameter

`requestSecurityToken` : see [R]RequestSecurityTokenType – The following `RequestSecurityTokenType` elements are required for an Issue request; the remaining elements are optional.

- `RequestType` – Must be the URL “http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue”.
- `Sig` attribute (`UseKey` element) – Specifies a security token that contains an existing certificate key for subject confirmation.
- `Context` attribute – Required if you include a `BinaryExchangeType` element for SSPI authentication.

Return Value

`RequestSecurityTokenResponseCollectionType` – Set of `RequestSecurityTokenResponseType`. A response contains a SAML token or a challenge requiring additional authentication data.

Comments

Sends a token request to a vCenter Single Sign-On server. The request message must contain security artifacts as determined by the vCenter Single Sign-On policy used for the request. The vCenter Single Sign-On server will authenticate the user credentials in the request. For information about configuring user directory support for authentication, see *`vSphere Installation and Setup`* and *`vSphere Security`* in the VMware Documentation Center. If the vCenter Single Sign-On server requires information during SSPI authentication, it will negotiate with the vCenter Single Sign-On client by embedding a challenge in the response.

Renew

Renews an existing SAML token.

Method Signature

```
Renew (token : RequestSecurityTokenType) returns RequestSecurityTokenResponseType
```

Parameter

token : `RequestSecurityTokenType` – Security token request containing a SAML token previously obtained from a vCenter Single Sign-On server. The token must be valid (not expired). The following **`RequestSecurityTokenType`** elements are required for a **`Renew`** request; the remaining elements are optional.

- **`RequestType`** – Must be the URL "`http://docs.oasis-open.org/ws-sx/ws-trust/200512/Renew`".
- **`RenewTarget`** – Identifies the SAML token to be renewed.
- **`Sig`** attribute (**`UseKey`** element) – Specifies a security token that contains an existing certificate key for subject confirmation.
- **`Context`** attribute – Required if you include a `BinaryExchangeType` element for SSPI authentication.

Return Value

`RequestSecurityTokenResponseType` – Response containing the renewed token.

Comments

You can renew holder-of-key tokens only. In addition to the the required token request elements shown above, the **`Renew`** request SOAP header must contain security elements according to the security policy.

Validate

Validates an existing SAML token.

Method Signature

```
Validate (token : RequestSecurityTokenType) returns RequestSecurityTokenResponseType
```

Parameter

token : `RequestSecurityTokenType` – Security token request containing a SAML token previously obtained from a vCenter Single Sign-On server. The following **`RequestSecurityTokenType`** elements are required for a **`Validate`** request; the remaining elements are optional.

- **`RequestType`** – Must specify the URL "`http://docs.oasis-open.org/ws-sx/ws-trust/200512/Validate`".
- **`ValidateTarget`** – Identifies the SAML token to be validated.
- **`Sig`** attribute (**`UseKey`** element) – Specifies a security token that contains an existing certificate key.
- **`Context`** attribute – Required if you include a `BinaryExchangeType` element for SSPI authentication.

Return Value

`RequestSecurityTokenResponseType` – Response containing the validated token.

Comments

Performs validation of the token and its subject. It includes but is not limited to validations of the following elements:

- Token signature
- Token lifetime
- Token subject
- Token delegates
- Group(s) to which the subject belongs

Challenge

Extends a token request to verify elements in the request.

Method Signature

```
Challenge (response : RequestSecurityTokenResponseType) returns
RequestSecurityTokenResponseType
```

Parameter

response : `RequestSecurityTokenResponseType` – Contains SSPI data in the **`BinaryExchange`** element.

Return Value

`RequestSecurityTokenResponseType` – Response containing the validated token.

Comments

Part of a negotiation with a vCenter Single Sign-On server to resolve issues related to SSPI authentication.

vCenter Single Sign-On API Data Structures

Use the following objects for the vCenter Single Sign-On methods.

RequestSecurityTokenType	ParticipantsType
RequestSecurityTokenResponseCollectionType	ParticipantType
RequestSecurityTokenResponseType	EndpointReference
LifetimeType	BinaryExchangeType
RenewingType	AdviceType
KeyTypeOpenEnum	AttributeType
UseKeyType	

RequestSecurityTokenType

Defines a set of token characteristics requested by the vCenter Single Sign-On client. The vCenter Single Sign-On client specifies this data object in a call to the `Issue`, `Renew`, and `Validate` methods. The vCenter Single Sign-On server may satisfy a request for a particular characteristic or it may use a different value in the issued token. The response to the token request contains the actual token values. See RequestSecurityTokenResponseType.

The vCenter Single Sign-On API supports a subset of the **`RequestSecurityTokenType`** elements defined in the WS-Trust specification. The following table shows the supported elements and attributes. An item in the table is defined as an element in the WSDL unless explicitly identified as an attribute.

Table 19: RequestSecurityTokenType Elements (vCenter Single Sign-On)

Element	Datatype	Description
Context	string	**`RequestSecurityToken`** attribute specifying a URI (Uniform Resource Identifier) that identifies the original request. If you include this in a request, the vCenter Single Sign-On server will include the

Table continued on next page

Continued from previous page

Element	Datatype	Description
		context identifier in the response. This attribute is required when the request includes a **`BinaryExchange`** property.
TokenType	string	Identifies the requested token type, specified as a URI (Uniform Resource Identifier). The following list shows the valid token types: • urn:oasis:names:tc:SAML:2.0:assertion – for issue and renew requests. • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Status – for validation requests.
RequestType	string	Identifies the request type, specified as a URI. The **`RequestType`** property is required. The following list shows the valid request types: • http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue • http://docs.oasis-open.org/ws-sx/ws-trust/200512/Renew • http://docs.oasis-open.org/ws-sx/ws-trust/200512/Validate
Lifetime	`LifetimeType`	Time period during which a token is valid. The vCenter Single Sign-On server can ignore the requested lifetime and assign a different lifetime to the token. The lifetime specifies creation and expiration values. This property is optional – used with `Issue` and `Renew` requests.
ValidateTarget		Specifies the token to be validated. This property can contain either a reference to the token or it can contain the token itself. The property is required for and used only with the `Validate` method.
RenewTarget		Specifies the token to be renewed. This property can contain either a reference to the token or it can contain the token itself. This property is required for and used only with the `Renew` method.
Renewing	`RenewingType`	Specifies a request for a renewable token. This property is optional. If you do not specify the **`Renewing`** property, the vCenter Single Sign-On server will issue a renewable token. This property is optional.
DelegateTo		Specifies a security token or token reference for an identity to which the requested token will be delegated. The **`DelegateTo`** value must identify a solution.
Delegatable	xs:boolean	Indicates whether the requested token can be delegated to an identity. Use this property together with the **`DelegateTo`** property. The default value for the **`Delegatable`** property is false.
UseKey	`UseKeyType`	References a token for subject confirmation. Required for `Issue`, `Renew`, and `Validate` methods.
KeyType	string	String value corresponding to a `KeyTypeOpenEnum` value. The value is a URI (Uniform Resource Identifier) that specifies the requested key cryptography type. This property is optional.
SignatureAlgorithm	string	Specifies a URI (Uniform Resource Identifier) for an algorithm that produces a digital signature for the token. The following list shows the valid values: • http://www.w3.org/2000/09/xmldsig#rsa-sha1 • http://www.w3.org/2001/04/xmldsig-more#rsa-sha256 • http://www.w3.org/2001/04/xmldsig-more#rsa-sha384 • http://www.w3.org/2001/04/xmldsig-more#rsa-sha512

Table continued on next page

Continued from previous page

Element	Datatype	Description
BinaryExchange	`BinaryExchangeType`	Contains data for challenge negotation between the vCenter Single Sign-On client and vCenter Single Sign-On server.
Participants	ParticipantsType	Specifies the identities of participants that are authorized to use the token.
AdviceSet	AdviceSetType	List of `AdviceType`.

RequestSecurityTokenResponseCollectionType

Returned by the `Issue` method. This type contains a response to the request or the requested token. .

Table 20: RequestSecurityTokenResponseCollectionType

Element	Datatype	Description
RequestSecurityTokenResponse	`RequestSecurityTokenResponseType[ ]`	List of token request response objects. The current architecture supports a single token response only

RequestSecurityTokenResponseType

Describes a single token.

Table 21: RequestSecurityTokenResponseType Properties (vCenter Single Sign-On)

Element	Datatype		Description
Context	string		
	`RequestSecurityTokenResponse` attribute specifying a URI (Uniform Resource Identifier) that identifies the original request. This attribute is included in the response if it was specified in the request.		
TokenType	string		
	Identifies the type of token in the response. TokenType is specified as a URI (Uniform Resource Identifier), one of the following: • urn:oasis:names:tc:SAML:2.0:assertion – for issue and renew operations. • http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Status – for validation operations.		
Lifetime	`LifetimeType`		
	Time period during which a token is valid. The lifetime in the token response is the actual lifetime assigned by the vCenter Single Sign-On server. The lifetime specifies creation and expiration values.		
RequestedSecurityToken	RequestedSecurityTokenType		
	SAML token.		
Renewing	`RenewingType`		

Table continued on next page

Continued from previous page

Element	Datatype		Description
	Indicates whether or not the token can be renewed. By default, the vCenter Single Sign-On server will issue a renewable token.		
BinaryExchange	`BinaryExchangeType`		
	Contains data for challenge negotiation between vCenter Single Sign-On client and vCenter Single Sign-On server.		
KeyType	string		
	Indicates whether or not key cryptography is used. The KeyType is a string value corresponding to an enumerated type value. See `KeyTypeOpenEnum`. The value is a URI (Uniform Resource Identifier) that specifies the key type.		
SignatureAlgorithm	string		
	Indicates a URI (Uniform Resource Identifier) for an algorithm that produces a digital signature for the token. The following list shows the valid values: • http://www.w3.org/2000/09/xmldsig#rsa-sha1 • http://www.w3.org/2001/04/xmldsig-more#rsa-sha256 • http://www.w3.org/2001/04/xmldsig-more#rsa-sha384 • http://www.w3.org/2001/04/xmldsig-more#rsa-sha512		
Delegatable	xs:boolean		
	Indicates whether the requested token can be delegated to an identity.		
Status	StatusType		
	Indicates the status of the request. The property specifies **`Code`** and **`Reason`** values.		

LifetimeType

Specifies the token lifetime. Used in `RequestSecurityTokenType` and `RequestSecurityTokenResponseType`.

Table 22: LifetimeType Properties

Property	Datatype	Description
created	wsu:AttributedDateTime	Creation time of the token. XML date and time, expressed as a standard time value (Gregorian calendar).
expires	wsu:AttributedDateTime	Time interval during which the token is valid, starting at the **`created`** time. The time interval is an absolute value specified in seconds.

RenewingType

Specifies token renewal.

Table 23: RenewingType Properties

Property	DataType	Description
Allow	xsd:boolean	Specifies a request for a token for which the lifetime can be extended. This property is optional. The default value is **`true`**.

Table continued on next page

Continued from previous page

Property	DataType	Description
OK	xsd:boolean	Indicates that the vCenter Single Sign-On client will accept a token that can be renewed after it has expired. This property is optional. The default value is **`false`**. If you specify this property, you must specify a value of **`false`**. A token that can be renewed after expiration does not provide adequate security.

KeyTypeOpenEnum

Specifies a set of enumerated type values that identify the supported types of key cryptography used for security tokens. The values are URIs (Universal Resource Identifiers).

Table 24: KeyType Properties

Enumerated type value	Description
http://docs.oasis-open.org/ws-sx/ws-trust/200512/PublicKey	Specifies asymmetric key cryptography using a combination of public and private keys. Use this key type for holder-of-key tokens.
http://docs.oasis-open.org/wssx/wstrust/200512/Bearer	Indicates a bearer token, which does not require a key to authenticate the token.

UseKeyType

Specifies the URI for an existing key.

Table 25: UseKeyType Properties

Property	Datatype	Description
Sig	string	URI (Universal Resource Identifer) that refers to a security token which contains an existing key. If specified, the vCenter Single Sign-On server will use the associated certificate for subject confirmation.

ParticipantsType

Identifies users and services who are allowed to use the token.

Table 26: ParticipantsType Properties

Property	Datatype	Description
Primary	`ParticipantType`	Primary user of the token.
Participant	`ParticipantType`	List of participants who are allowed to use the token.

ParticipantType

ParticipantType is an end point reference.

Table 27: ParticipantType Property

Property	Datatype	Description
	`EndpointReference`	Specifies a participant represented as a URI.

EndpointReference

Participant identification. The ReferenceParameters, **`Metadata`**, and **`any`** elements are not used.

Table 28: EndpointReference Property

Property	Datatype	Description
name	tns:AttributedURIType	URI that identifies a participant allowed to use a token.

BinaryExchangeType

Specifies a blob (binary large object) that contains data for negotation between the vCenter Single Sign-On client and server.

Table 29: BinaryExchangeType Attributes

Attribute	Datatype	Description
ValueType	xsd:anyURI	Identifies the type of negotiation.
EncodingType	xsd:anyURI	Identifies the encoding format of the blob.

AdviceType

Specifies additional informational attributes to be included in the issued token. The vCenter Single Sign-On client can ignore this data. Advice data will be copied to delegate tokens. This type is used in `RequestSecurityTokenType`.

Table 30: AdviceType Properties

Element/Attribute	Datatype	Description
Advicesource	string	AdviceType attribute specifying a URI representing the identity that provides the advice Attribute elements. This attribute is required.
Attribute	`AttributeType`	Advice data.

AttributeType

Attribute providing advice data. Used in `AdviceType`.

Table 31: AttributeType Properties

Element/Attribute	Datatype	Description
Name	string	**`AttributeType`** attribute specifying a URI that is the unique name of the attribute. This attribute is required.
FriendlyName	string	**`AttributeType`** attribute specifying a human-readable form of the name. This attribute is optional.
AttributeValue	string	List of values associated with the attribute. The **`AttributeValue`** structure depends on the following criteria: • If the attribute has one or more values, the **`AttributeType`** contains one **`AttributeValue`** for each value. Empty attribute values are represented by empty **`AttributeValue`** elements. • If the attribute does not have a value, the **`AttributeType`** does not contain an **`AttributeValue`**.

vCenter Single Sign-On Client Example (JAX-WS)

This chapter describes a Java example of acquiring a vCenter Single Sign-On security token.

vCenter Single Sign-On Token Request Overview

The code examples in the following sections show how to use the **`Issue`** method to acquire a holder-of-key security token. To see an example of using the token to login to a vCenter server, see LoginByToken Example (JAX-WS). The code examples in this chapter are based on the following sample file located in the vCenter Single Sign-On SDK JAX-WS client **`samples`** directory:

```
.../JAXWS/samples/com/vmware/sso/client/samples/AcquireHoKTokenByUserCredentialSample.java
```

The **`AcquireHoKTokenByUserCredentialSample`** program creates a token request and calls the issue method to send the request to a vCenter Single Sign-On server. The program uses a sample implementation of Web services message handlers to modify the SOAP security header for the request message.

This example uses the username-password security policy (**`STSSecPolicy_UserPwd`**). This policy requires that the SOAP security header include a timestamp, username and password, and a digital signature and certificate. The sample message handlers embed these elements in the message.

The example performs the following operations:

1. Create a security token service client object (**`STSService_Service`**). This object manages the vCenter Single Sign-On header handlers and it provides access to the vCenter Single Sign-On client API methods. This example uses the issue method.
2. Create a vCenter Single Sign-On header handler resolver object (**`HeaderHandlerResolver`**). This object acts as a container for the different handlers.
3. Add the handlers for timestamp, user credentials, certificate, and token extraction to the handler resolver.
4. Add the handler resolver to the security token service.
5. Retrieve the STS port (**`STS_Service`**) from the security token service object.
6. Create a security token request.
7. Set the request fields.
8. Set the endpoint in the request context. The endpoint identifies the vCenter Single Sign-On server.
9. Call the issue method, passing the token request.
10. Handle the response from the vCenter Single Sign-On server.

Using Handler Methods for SOAP Headers in Java

The VMware vCenter Single Sign-On SDK provides sample code that is an extension of the JAX-WS XML Web services message handler (**`javax.xml.ws.handler`**). The sample code consists of a set of SOAP header handler methods and a header handler resolver, to which you add the handler methods. The handler methods insert timestamp, user credential, and message signature data into the SOAP security header for the request. A handler method extracts the SAML token from the vCenter Single Sign-On server response.

The VMware vCenter Single Sign-On client SOAP header handler files are located in the **`soaphandlers`** directory:

```
SDK/sso/java/JAXWS/samples/com/vmware/sso/client/soaphandlers
```

To access the SOAP handler implementation, the example code contains the following import statements:

```
import com.vmware.sso.client.soaphandlers.HeaderHandlerResolver;

import com.vmware.sso.client.soaphandlers.SSOHeaderHandler;

import com.vmware.sso.client.soaphandlers.SamlTokenExtractionHandler

import com.vmware.sso.client.soaphandlers.TimeStampHandler;

import com.vmware.sso.client.soaphandlers.UserCredentialHandler;

import com.vmware.sso.client.soaphandlers.WsSecurityUserCertificateSignatureHandler;
```

This example uses the following handler elements:

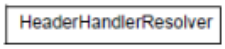
- HeaderHandlerResolver
- SamlTokenExtractionHandler
- TimestampHandler
- UserCredentialHandler
- WsSecurityUserCertificateSignatureHandler (SSOHeaderHandler)

The following sequence shows the operations and corresponding Java elements for message security.

1 Create an STS service object (`STSService_Service`). This object will bind the handlers to the request and provide access to the issue method.



2 Create a handler resolver object (`HeaderHandlerResolver`). This object acts as a receptacle for the handlers.

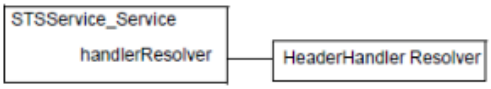


3 Add the header handlers:

- Timestamp – The handler will use system time to set the timestamp values.
- User credential – The handler requires a username and a password; it will create a username token for the supplied values.
- User certificate signature – The handler requires a private key and an x509 certificate. The handler will use the private key to sign the body of the SOAP message (the token request), and it will embed the certificate in the SOAP security header.
- SAML token extraction – The handler extracts the SAML token directly from vCenter Single Sign-On server response to avoid token modification by the JAX-WS bindings.



4 Add the handler resolver to the STS service.



The following example creates a handler resolver and adds the handler methods to the handler resolver. After the handlers have been established, the client creates a token request and calls the **`Issue`** method. See Sending a Request for a Security Token in Java.

IMPORTANT
You must perform these steps for message security before retrieving the STS service port. An example of retrieving the STS service port is shown in Sending a Request for a Security Token in Java.

Acquiring a vCenter Single Sign-On Token – Soap Handlers

```
/*
 * Instantiate the STS Service
 */
 STSService_Service stsService = new STSService_Service();


/*
 * Instantiate the HeaderHandlerResolver.
 */
 HeaderHandlerResolver headerResolver = new HeaderHandlerResolver();


/*
```

```
 * Add handlers to insert a timestamp and username token into the SOAP security header
 * and sign the message.
 *
 * -- Timestamp contains the creation and expiration time for the request
 * -- UsernameToken contains the username/password
 * -- Sign the SOAP message using the combination of private key and user certificate.
 *
 * Add the TimeStampHandler
 */
 headerResolver.addHandler(new TimeStampHandler());

/*
 * Add the UserCredentialHandler. arg[1] is the username; arg[2] is the password.
 */
 UserCredentialHandler ucHandler = new UserCredentialHandler(args[1],    args[2]);
 headerResolver.addHandler(ucHandler);

/*
 * Add the message signature handler (WsSecurityUserCertificateSignatureHandler);
 * The client is responsible for supplying the private key and certificate.
 */
 SSOHeaderHandler ssoHandler =
    new WsSecurityUserCertificateSignatureHandler(privateKey, userCert);
 headerResolver.addHandler(ssoHandler);

/*
 * Add the token extraction handler (SamlTokenExtractionHandler).
 */
 SamlTokenExtractionHandler sbHandler = new SamlTokenExtractionHandler;
 headerResolver.addHandler(sbHandler);

/*
```

```
 * Set the handlerResolver for the STSService to the HeaderHandlerResolver created above.
 */
 stsService.setHandlerResolver(headerResolver);
```

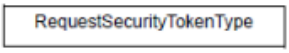
Sending a Request for a Security Token in Java

After setting up the SOAP header handlers, the example creates a token request and calls the issue method. The following sequence shows the operations and corresponding Java elements.

5 Retrieve the STS service port (`STSService`). The service port provides access to the vCenter Single Sign-On client API methods. The vCenter Single Sign-On handler resolver must be associated with the STS service before you retrieve the service port. See "Using Handler Methods for SOAP Headers" on page 34.



6 Create a token request (`RequestSecurityTokenType`). Your vCenter Single Sign-On client will pass the token request to the `Issue` method. The `Issue` method will send the token request in the body of the SOAP message. This example sets the token request fields as appropriate for a holder-of-key token request.

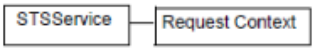


7 Set the token request fields.

- lifetime – Creation and expiration times.
- token type – urn:oasis:names:tc:SAML:2.0:assertion
- request type – http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
- key type – http://docs.oasis-open.org/ws-sx/ws-trust/200512/PublicKey (for holder-of-key token type)
- signature algorithm – http://www.w3.org/2001/04/xmldsig-more#rsa-sha256
- renewable status



8 Set the endpoint address for the token request.



9 Call the Issue method.



10 Handle the response from the vCenter Single Sign-On server.



The following code example shows Java code that performs these operations.

Acquiring a vCenter Single Sign-On Token – Sending the Request

```
/*
 * Retrieve the STSServicePort from the STSService_Service object.
 */
 STSService stsPort = stsService.getSTSServicePort();


/*
```

```
 * Create a token request object.
 */
 RequestSecurityTokenType tokenType = new RequestSecurityTokenType();

/*
 *  Create a LifetimeType object.
 */
 LifetimeType lifetime = new LifetimeType();

/*
 * Derive the token creation date and time.
 * Use a GregorianCalendar to establish the current time,
 * then use a DatatypeFactory to map the time data to XML.
 */
 DatatypeFactory dtFactory = DatatypeFactory.newInstance();
 GregorianCalendar cal = new GregorianCalendar(TimeZone.getTimeZone("GMT"));
 XMLGregorianCalendar xmlCalendar = dtFactory.newXMLGregorianCalendar(cal);
 AttributedDateTime created = new AttributedDateTime();
 created.setValue(xmlCalendar.toXMLFormat());

/*
 * Specify a time interval for token expiration (specified in milliseconds).
 */
 AttributedDateTime expires = new AttributedDateTime();
 xmlCalendar.add(dtFactory.newDuration(30 * 60 * 1000));
 expires.setValue(xmlCalendar.toXMLFormat());

/*
 * Set the created and expires fields in the lifetime object.
 */
 lifetime.setCreated(created);
 lifetime.setExpires(expires);
```

```
/*
 * Set the token request fields.
 */
 tokenType.setTokenType("urn:oasis:names:tc:SAML:2.0:assertion");
 tokenType.setRequestType("http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue");
 tokenType.setLifetime(lifetime);
 tokenType.setKeyType("http://docs.oasis-open.org/ws-sx/ws-trust/200512/PublicKey");
 tokenType.setSignatureAlgorithm("http://www.w3.org/2001/04/xmldsig-more#rsa-sha256");

/*
 * Specify a token that can be renewed.
 */
 RenewingType renewing = new RenewingType();
 renewing.setAllow(Boolean.TRUE);
 renewing.setOK(Boolean.FALSE); // WS-Trust Profile: MUST be set to false
 tokenType.setRenewing(renewing);

/* Get the request context and set the endpoint address. */
 Map<String, Object> reqContext = ((BindingProvider) stsPort).getRequestContext();
 reqContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, args[0]);

/*
 * Use the STS port to invoke the "issue" method to acquire the token
 * from the vCenter Single Sign-On server.
 */
 RequestSecurityTokenResponseCollectionType issueResponse = stsPort.issue(tokenType);

/*
 * Handle the response - extract the SAML token from the response. The response type
 * contains the token type (SAML token type urn:oasis:names:tc:SAML:2.0:assertion).
 */
 RequestSecurityTokenResponseType rstResponse =
```

```
issueResponse.getRequestSecurityTokenResponse();

 RequestedSecurityTokenType requestedSecurityToken =
rstResponse.getRequestedSecurityToken();


/*

 * Extract the SAML token from the RequestedSecurityTokenType object.

 * The generic token type (Element) corresponds to the type required

 * for the SAML token handler that supports the call to LoginByToken.

 */

 Element token = requestedSecurityToken.getAny();
```

LoginByToken Example (JAX-WS)

This chapter describes a Java example of using the **`LoginByToken`** method. A vCenter Server session begins with the call to the LoginByToken method after receiving the token from the SSO Server.

vCenter Server Single Sign-On Session

After you obtain a SAML token from the vCenter Single Sign-On server, you can use the vSphere Web Services API method **`LoginByToken`** to establish a single sign-on session with a vCenter Server. See vCenter Single Sign-On Client Example (JAX-WS) for an example of obtaining a vCenter Single Sign-On token.

At the beginning of a vCenter Single Sign-On session, your client is responsible for the following tasks:

- Insert the vCenter Single Sign-On token and a timestamp into the SOAP header of the **`LoginByToken`** message.
- Maintain the vCenter Server session cookie. During the login sequence, the server produces an HTTP session cookie to support the persistent connection. Your client must save this cookie and re-introduce it at the appropriate times.
- If at a later time your client invokes the **`LoginByToken`** method, or other login method, the Server issues a new session cookie in response. You must have a cookie handler in place to save the cookie for subsequent requests.

The example program uses these general steps:

1. Call the **`RetrieveServiceContent`** method. The method establishes the connection with the vCenter Server and provides access to the **`SessionManager`** managed object.
2. Call the **`LoginByToken`** method to authenticate the vCenter session. To send the token to the vCenter Server, the client uses a handler to embed the token and a time stamp in the SOAP header for the message. The client uses an HTTP header handler method to extract the cookie from the vCenter Server response.
3. Restore the session cookie for future requests. To identify the session started with the **`LoginByToken`** method, the client uses a handler to embed the session cookie in the HTTP header.

HTTP and SOAP Header Handlers in Java

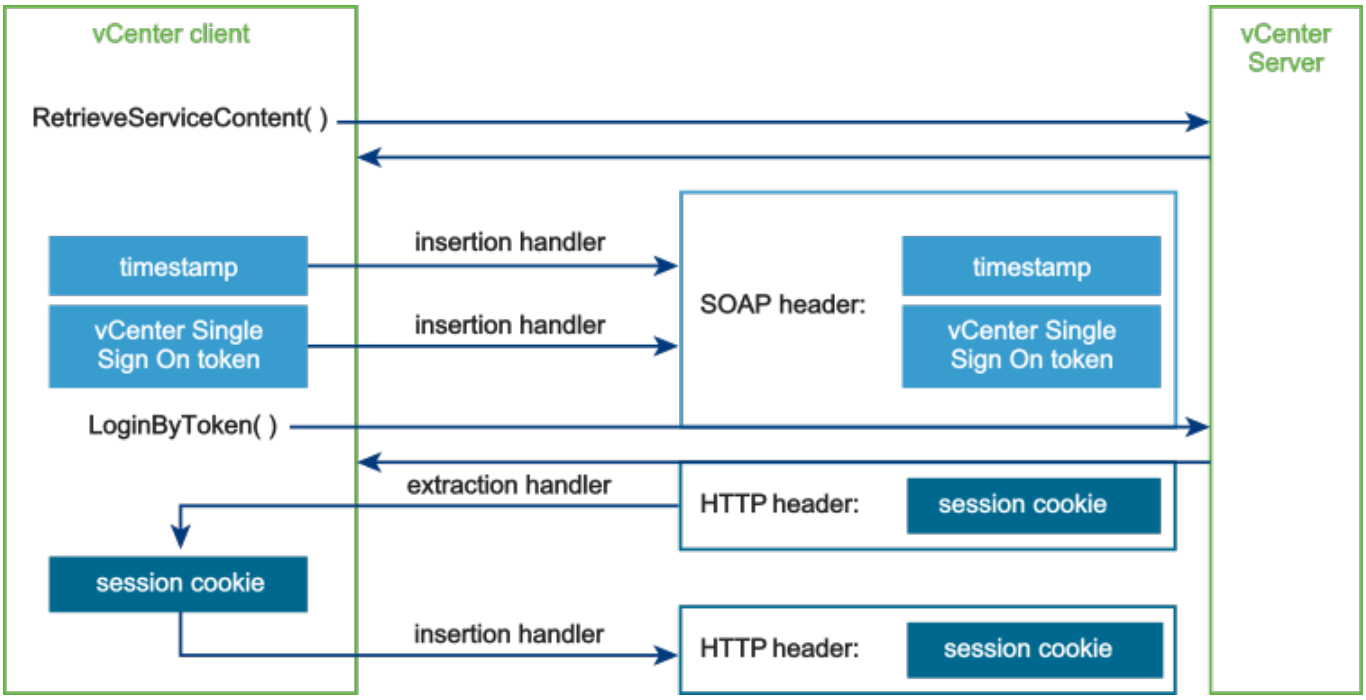
To use a vCenter Single Sign-On token to login to a vCenter server, the example uses header handlers to manipulates the HTTP and SOAP header elements of the login request. After establishing a handler, subsequent requests automatically invoke the handler.

- Insertion handlers put the vCenter Single Sign On token and a timestamp into the SOAP header into the HTTP header of the login request.

- An extraction handler obtains the HTTP session cookie provided by the vCenter Server. After setting up the handler, a call to the **`LoginByToken`** method will invoke the handler to extract the cookie from the Server response.

The following figure shows the use of handlers to manipulate header elements when establishing a vCenter Single Sign On session with a vCenter Server.

Figure 83: Starting a vCenter Session



IMPORTANT
Every call to the vCenter Server will invoke any message handlers that have been established. The overhead involved in using the SOAP and HTTP message handlers is not necessary after the session has been established. The example saves the default message handler before setting up the SOAP and HTTP handlers. After establishing the session, the example will reset the handler chain and restore the default handler.
The example code also uses multiple calls to the **`VimPortType.getVimPort method`** to manage the request context. The **`getVimPort`** method clears the HTTP request context. After each call to the **`getVimPort`** method, the client resets the request context endpoint address to the vCenter Server URL. After the client has obtained the session cookie, it will restore the cookie in subsequent requests.

LoginByToken Sample Code in Java

The code examples in the following sections show how to use the **`LoginByToken`** method with a holder-of-key security token. The code examples are based on the sample code contained in the vCenter Single Sign-On SDK. The files are located in the Java samples directory (`SDK/ssoclient/java/JAXWS/samples`):

- LoginByToken sample:
  `samples/com/vmware/vsphere/samples/LoginByTokenSample.java`
- Header cookie handlers:
  `samples/com/vmware/vsphere/soaphandlers/HeaderCookieHandler.java`

  `samples/com/vmware/vsphere/soaphandlers/HeaderCookieExtractionHandler.java`
- SOAP header handlers. These are the same handlers that are used in vCenter Single Sign-On Client Example (JAX-WS). The SOAP handler files are located in the vCenter Single Sign-On client **`soaphandlers`** directory:
  `samples/com/vmware/sso/client/soaphandlers`

Steps to Connect with a vSphere Server

For a connection with a vSphere server, use the following steps:

1. Get the `VimPort` object for access to the vSphere services.
2. Call the `RetrieveServiceContent` method.
3. Call the `LoginByToken` method; insert a SAML token into the request header..
4. Extract the session cookie from the response.
5. Use the session cookie to set the request context for future requests.

The following sections describe these steps.

Import the Necessary Java Packages for LoginByToken

The following example describes the packages that you need to import into your application to use the new LoginByToken method:

Import the Packages that Are Needed

```
package com.vmware.vsphere.samples;

import java.util.Arrays;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import javax.xml.ws.BindingProvider;

import javax.xml.ws.handler.HandlerResolver;

import javax.xml.ws.handler.MessageContext;

import org.w3c.dom.Element;

import com.vmware.sso.client.soaphandlers.HeaderHandlerResolver;

import com.vmware.sso.client.soaphandlers.SamlTokenHandler;

import com.vmware.sso.client.soaphandlers.TimeStampHandler;

import com.vmware.vim25.InvalidLocaleFaultMsg;

import com.vmware.vim25.InvalidLoginFaultMsg;

import com.vmware.vim25.ManagedObjectReference;

import com.vmware.vim25.RuntimeFaultFaultMsg;

import com.vmware.vim25.ServiceContent;

import com.vmware.vim25.VimPortType;

import com.vmware.vim25.VimService;

import com.vmware.vsphere.soaphandlers.HeaderCookieExtractionHandler;
```

Get the VimPort for LoginByToken in Java

The following example shows the code to get the VimPort:

Getting the VimPort

```
package com.vmware.vsphere.samples;


/**
* Sample program demonstrating the usage of the new loginByToken method,
* introduced to authenticate the client using the SAML token obtained from the
* SSO server.
*
* @author VMware, Inc.
*/
public class LoginByTokenExample {
/**
* This method invokes the loginByToken method for authentication. Once this
* method is called the established session is authenticated and operations
* can be performed on the connected vCenter server
*
* @param token
* {@link Element} representing the SAML token that needs to be
* used for the authentication
* @param vcServerUrl
* The vCenter server URL that needs to be connected
* @return String authenticated session cookie used by the connection
* @throws RuntimeFaultFaultMsg
* @throws InvalidLocaleFaultMsg
*/
public static String loginUsingSAMLToken(Element token, String vcServerUrl)
throws RuntimeFaultFaultMsg, InvalidLocaleFaultMsg,
InvalidLoginFaultMsg {
VimService vimService = new VimService();
```

```
HandlerResolver defaultHandler = vimService.getHandlerResolver();


// Step 1 Get the VimPort object
vimService = new VimService();
VimPortType vimPort = vimService.getVimPort();
```

Retrieve the Service Content in Java

The following example shows the call to `retrieveServiceContent`.

Call `retrieveServiceContent`

```
// Step 2 Retrieve the service content
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, vcServerUrl);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, false);
ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
SVC_INST_REF.setType("ServiceInstance");
SVC_INST_REF.setValue("ServiceInstance");
ServiceContent serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
```

Invoke the `loginByToken` method in Java

The following example shows the `loginByToken` call.

Call the `loginByToken` Method

```
// Step 3 Invoking the loginByToken method
HeaderHandlerResolver handlerResolver = new HeaderHandlerResolver();
handlerResolver.addHandler(new TimeStampHandler());
handlerResolver.addHandler(new SamlTokenHandler(token));
HeaderCookieExtractionHandler cookieExtractor = new HeaderCookieExtractionHandler();
handlerResolver.addHandler(cookieExtractor);
vimService.setHandlerResolver(handlerResolver);
vimPort = vimService.getVimPort();
Map<String, Object> ctxt2 = ((BindingProvider) vimPort).getRequestContext();
```

```
ctxt2.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, vcServerUrl);
ctxt2.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);
vimPort.loginByToken(serviceContent.getSessionManager(), null);
```

Extract the Session Cookie in Java

After calling the `loginByToken`, you extract the session cookie, as shown in the following example.

Extract the Session Cookie

```
// Step 4 Extract the session cookie that is established in the previous
// call to the server.
String cookie = cookieExtractor.getCookie();
// Clear the HandlerResolver chain setup in step 3 as that is
// required only one time for the invocation of the LoginByToken method.
// After login we do not need this handler thus reverting to the
// original (default) handler.
vimService.setHandlerResolver(defaultHandler);
vimPort = vimService.getVimPort();
```

Inject the Session Cookie Back Into the Request in Java

Before making another call to the server, inject the session cookie back into the request as shown in the following example.

Inject the Session Cookie Back Into the Request

```
// Step 5 Inject the session cookie back into the request once before
// making another call to the server. JAXWS will maintain that cookie
// for all subsequent requests.
Map<String, Object> ctxt3 = ((BindingProvider) vimPort).getRequestContext();
ctxt3.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, vcServerUrl);
ctxt3.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);
// One time setting of the cookie
@SuppressWarnings("unchecked")
Map<String, List<String>> headers =
  (Map<String, List<String>>) ctxt3.get(MessageContext.HTTP_REQUEST_HEADERS);
```

```
if (headers == null) {
   headers = new HashMap<String, List<String>>();
}
headers.put("Cookie", Arrays.asList(cookie));
ctxt3.put(MessageContext.HTTP_REQUEST_HEADERS, headers);
// Authentication complete. Proceed with rest of the API calls
// that are required for your functionality.
return cookie;
}
}
```

Additional Information

Please refer to the `LoginByTokenSample` Java sample source in the vSphere Management SDK for more information. Also refer to the VMware *vSphere API Reference Documentation* and the *vSphere Web Services SDK Developer's Setup Guide*.

Using a Proxy with vSphere Virtual Serial Ports

The VMware® *Using a Proxy with vSphere Virtual Serial Ports* manual provides information about developing a virtual serial port concentrator for use in the vSphere environment.

VMware provides different APIs and SDKs for various applications and goals. This manual describes the capabilities of a proxy implementation that supports the VMware telnet extension. It also includes information about configuring a virtual serial port for a proxy connection.

The use case for a virtual serial port proxy is to maintain specific network connections during vMotion, which is the movement of virtual machines from one ESXi host to another. Note that this method uses telnet so is best deployed in a private data center.

Revision History

The Revision History summarizes significant changes in each version of this book.

Table 32: Revision History

Revision Date	Description
26APR2016	Support for VMware vSphere 6.5
12MAR2015	Support for VMware vSphere 6.0
11Mar2014	Supported on VMware vSphere 6.0 Beta.
19Sep2013	Supported in VMware vSphere 5.5; added information about TCP keepalive configuratio (vmotionLimit).
10Sep2012	Supported in VMware vSphere 5.1; added syntax for authentication parameters used wi
24Aug2011	Supported in VMware vSphere 5.0.
13JUL2010	Initial release, supported in VMware vSphere 4.1.

Intended Audience

This book is intended for anyone developing a virtual serial port proxy for the vSphere environment. Proxy developers should have experience with the telnet protocol and network communication.

Remote Serial Port Access with a Virtual Serial Port Proxy

The VMware vSphere™ API supports the use of virtual serial ports on virtual machines that run on ESX™ hosts. A virtual serial port represents a serial port on a virtual machine. A virtual serial port can operate as a network serial port by using a network socket on the host to provide access to remote systems. A remote system is any system on the network that supports a serial port connection with a virtual machine. This feature supports remote console login for system management using an out-of-band connection that bypasses the vSphere Client.

You can also use a virtual serial port for communication through a software pipe, for output to a file, and for communication through a physical serial port on the host. For information about the vSphere API support for these applications, see the description of the **`VirtualSerialPort`** object in the *vSphere API Reference*.

In a VMware® data center, a vCenter Server uses vMotion™ to move virtual machines across hosts as part of resource management. vSphere does not support vMotion operations on a virtual machine that has a direct network serial port connection with a remote system. Network serial ports use the host IP address. If the server attempts to move a virtual machine with a network serial port connection, the host rejects the vMotion request to maintain the connection.

To support vMotion for virtual machines that use network serial ports, you can develop a proxy that operates between a virtual machine and a remote system. The vSphere API supports the use of a third-party virtual serial port concentrator in the data center environment. A virtual serial port concentrator acts as a proxy or an access server between virtual machines and remote systems. The proxy maintains communication with virtual machines that are using network serial ports. When a vMotion event occurs, the virtual machine connects to the proxy from its new host so that the connection with the remote system persists.

Virtual machines use the telnet protocol for virtual serial port proxy connections. VMware defines a telnet extension that includes commands for proxy notification of vMotion events.

See VMware Telnet Extension Commands for Proxy Communication for reference information about the VMware telnet extension commands.

Virtual Serial Port Proxy Capabilities

A virtual serial port proxy operates as an RFC2217-compatible telnet server. A proxy provides the following capabilities in the vSphere environment.

- Accepts telnet connections from virtual machines
- Supports VMware telnet extension commands and responds to VMware telnet extension messages from virtual machines
- Forwards network traffic between virtual machines and remote systems
- Handles extension commands for notification of vMotion events and redirects traffic accordingly

Proxy as a Virtual Serial Port Concentrator

Figure: Virtual Serial Port Concentrator shows a proxy that operates as a virtual serial port concentrator. The concentrator has internal connections to several virtual machines in a data center and an external connection to a remote system.

The concentrator provides a remote system with access to multiple virtual machines that act as serial port servers. The virtual machines connect to the concentrator, indicating that they will accept connections from remote systems. See Backing Options and Backing Information for Network Serial Ports for more information about serial port servers. The concentrator presents the choice of virtual machines to the remote system and maintains any subsequent connection. In Figure: Virtual Serial Port Concentrator, a solid line represents an active connection between the remote system and a virtual machine, and dashed lines represent potential connections.

Figure 84: Virtual Serial Port Concentrator

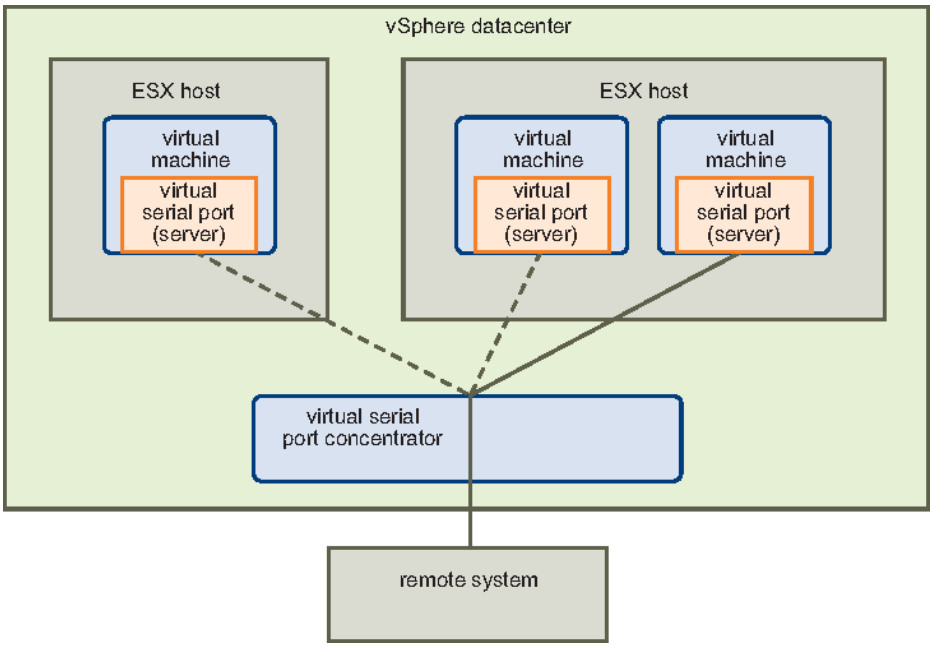


Figure: Virtual Serial Port Concentrator shows the concentrator as part of the vSphere data center. For this kind of application, you can deploy the concentrator as a virtual appliance. A virtual appliance is software on a platform of one or more virtual machines (in this case, a single virtual machine). With a virtual appliance implementation, you can take advantage of resource management that vCenter Server provides. The vCenter Server uses vMotion technology to migrate the virtual appliance proxy through the data center without interrupting remote serial port service. The virtual appliance uses virtual machine IP addresses that persist across vMotion events.

Although the primary purpose of the proxy is to preserve connections during vMotion events, it also can provide a gateway or firewall to enforce security. The proxy serves as the point of authentication for remote systems that use virtual serial ports to access virtual machines.

Network Serial Ports

A network serial port is a virtual serial port that uses a network socket on the host computer. A network socket is the endpoint of a network connection and is represented by an IP address and a port number. The vSphere API uses Uniform Resource Identifiers (URI) to identify the protocol, IP address, and port number for endpoints of network serial port connections. See RFC3986 for information about Uniform Resource Identifiers.

To use a network serial port, you must verify that the ESX host supports the feature and then create or reconfigure a virtual machine to configure the port. For information about writing an application that uses the vSphere API, see the *vSphere Web Services SDK Programming Guide*. As an alternative, you can use the vSphere Client to configure a network serial port. For more information, see the online help for the VMware vSphere Client.

Backing Options and Backing Information for Network Serial Ports

The vSphere API uses backing objects to represent the host device that is associated with a virtual device. The API offers two types of backing objects for virtual devices: option objects and information objects.

Backing option objects identify host device support for virtual machines. An ESX host defines the set of virtual device backing options that are available for virtual machines. To use a network serial port, the ESX host must provide the virtual serial port URI backing option, which is represented by the **`VirtualSerialPortURIBackingOption`** object. For information about how to find out if the host supports the URI backing option, see Determining Backing Options.

Backing information objects supply data for virtual device configuration. Use a **`VirtualSerialPortURIBackingInfo`** object to bind a virtual serial port to a network socket. You can configure a virtual serial port to use URI backing when you create or reconfigure a virtual machine. For a description of how to configure URI backing for a network serial port, see Create a Virtual Machine with a Network Serial Port.

When you specify URI backing for the network serial port connection, you define these characteristics.

- Direction – defines the role that the virtual machine plays in a connection with a remote system (client or server). When a virtual machine acts as a serial port client, it initiates the connection with the remote system. When a virtual machine acts as a serial port server, it listens for connections that are initiated by remote clients.
- Service URI – identifies the target of connections. When the virtual machine acts as a client, the service URI identifies the remote system. When the virtual machine acts as a server, the service URI identifies proxy access to the host on which the virtual machine runs. See RFC3986 for information about URI syntax.
- Proxy URI – identifies the virtual serial port proxy. The presence of the proxy URI directs the virtual machine to establish a connection with the proxy. The virtual machine always initiates the connection with the proxy. You can use either telnet or secure telnet (telnets) for the proxy connection. The proxy URI specification must use one of the following formats. See RFC3986 for information about URI syntax. See the description of the **`VirtualSerialPortURIBackingInfo`** data object in the *`vSphere API Reference`* for information about URI schemes and authentication parameters for network serial port configuration.
  - telnet://<host>:<port>[#key[=value][&key[=value] ...]]
  - telnets://<host>:<port>[#key[=value][&key[=value] ...]]
    When the virtual machine performs telnet negotiation with the proxy, it sends the direction and service URI information to the proxy. (See the description of the **`DO-PROXY`** command in Proxy Operation.)

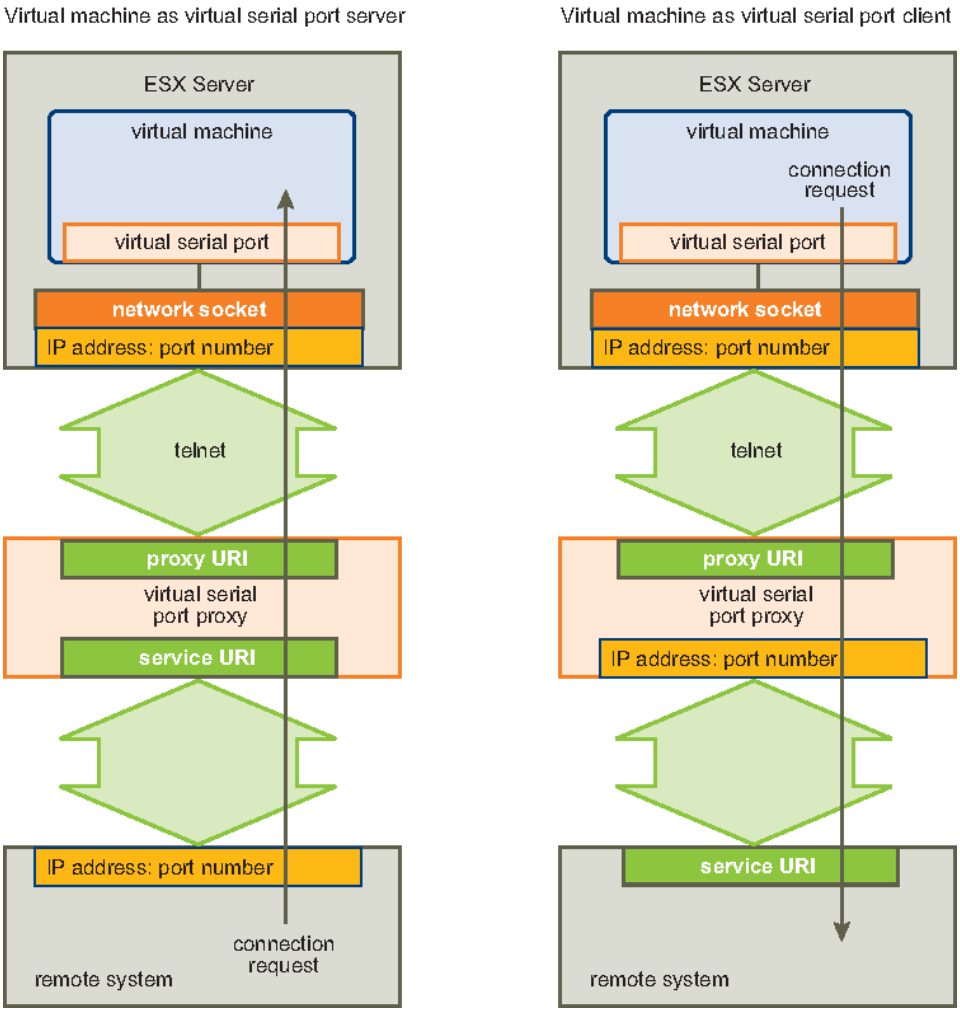
Network Connections Between a Virtual Machine, a Proxy, and a Remote System

Figure: Virtual Serial Port Connections to a Remote System Through a Proxy shows two virtual machines. Each machine uses a proxy for its connection to a remote system.

- One virtual machine acts as a virtual serial port server. This virtual machine uses the service URI to identify proxy access to the host machine. The virtual machine initiates a telnet connection with the proxy. The proxy, on behalf of the virtual machine, listens for connection requests from remote systems and forwards the requests to the virtual machine.
- One virtual machine acts as a virtual serial port client. This virtual machine uses the service URI to identify the remote system. The virtual machine initiates a telnet connection with the proxy. The proxy forwards the connection request to the remote system.

The proxy has two network sockets. It uses one socket (proxy URI) for the telnet connections with the virtual machine and the other for the connection with the remote system. The external connection between the proxy and a remote system uses the mutually agreed upon protocol. For a connection initiated by the remote system, the person operating the remote system must obtain the service URI by email or other independent communication.

Figure 85: Virtual Serial Port Connections to a Remote System Through a Proxy



Determining Backing Options

Use the **`QueryConfigOption`** method to determine which backing options are available on the host on which you will create the virtual machine. The method returns a **`VirtualMachineConfigOption`** data object. The virtual machine configuration data includes a a list of virtual device options and a list of backing options. The following pseudocode shows the path to the backing options.

```
VirtualMachineConfigOption.hardwareOptions.VirtualDeviceOption[].backingOption[]
```

The virtual device option array must include the **`VirtualSerialPortOption`** data object. For network backing, the array of serial port backing options must include the **`VirtualDeviceURIBackingOption`** data object. The `VirtualDeviceURIBackingOption.type` property specifies the name of the backing object you should create when you configure network backing.

Create a Virtual Machine with a Network Serial Port

You can use the **`CreateVM_Task`** method to create a virtual machine and configure a network serial port. The virtual machine uses the virtual serial port configuration to connect to a proxy and to accept a connection from a remote system.

1. Create a **`VirtualMachineConfigSpec`** object for the method's **`config`** parameter.

2. Create a **VirtualDeviceConfigSpec** object for the **deviceChange** array in the **VirtualMachineConfigSpec** object.
3. Create a **VirtualSerialPort** object for the **device** property in the **VirtualDeviceConfigSpec** object.
4. To control proxy connection behavior, create a **VirtualDeviceConnectInfo** data object for the **VirtualSerialPort.connectable** property.

   The following pseudocode shows the resulting path to the connection information.

   ```
   VirtualMachineConfigSpec.deviceChange[].device.connectable
   ```

5. Set the **startConnected** property in the **connectable** object to TRUE.

   The **startConnected** property determines whether the virtual machine will open a connection with the proxy when the virtual machine starts. When the virtual machine powers on, it copies the **startConnected** value to the **VirtualDeviceConnectInfo.connected** property. While the virtual machine is running, you can use the **ReconfigVM_Task** method to set the **connected** property to open or close the proxy connection. If you reconfigure a port with **connected** set to TRUE, the virtual machine closes the existing connection and opens a new one.

6. Create a **VirtualDeviceURIBackingInfo** object for the **backing** property in the **VirtualSerialPort** object.

   The following pseudocode shows the resulting path to the backing information.

   ```
   VirtualMachineConfigSpec.deviceChange[].device.backing
   ```

   Use the backing information object to supply the following information:

   - Set the **direction** property to “server” to direct the virtual machine to accept remote connections.
   - Set the **serviceURI** property to the URI for the host on which the virtual machine runs.
   - Set the **proxyURI** property to the URI for the virtual serial port concentrator. You cannot include a user/password sequence in the proxy URI.

VMware Telnet Session

To establish a VMware telnet session for remote serial port communication, a virtual machine and a proxy participate in a message exchange that follows the model of a standard telnet protocol sequence. The proxy handles messages from the virtual machine and sends the appropriate responses. Before the proxy can forward content, the virtual machine and the proxy participate in the following progression.

1. The virtual machine and the proxy negotiate the use of the VMware telnet extension option.
   Both sides of the connection must agree to use the option before continuing with any additional extension message exchange.
2. The virtual machine and the proxy identify the VMware telnet suboption commands that they support.
   The proxy must support the **KNOWN-SUBOPTIONS-2** and **UNKNOWN-SUBOPTION-RCVD-2** commands. During a session, the proxy can send the **UNKNOWN-SUBOPTION-RCVD-2** command as a valid response to virtual machine messages.
3. The proxy agrees to act as a VMware telnet extension proxy for remote serial port communication.

VMware Telnet Commands

Table lists the VMware telnet extension commands and their brief descriptions. Commands in the table are grouped by purpose: VMware telnet extension and suboptions, proxy, vMotion, and identification. The table uses command symbols. A VMware telnet extension command is a two- or three-byte code embedded in a telnet command sequence. For more

information about commands and the code equivalents, see VMware Telnet Extension Commands for Proxy Communication.

Command	Source	Description
VMWARE-TELNET-EXT	virtual machine, proxy	The VMware telnet extension option command
KNOWN-SUBOPTIONS-1	virtual machine	Suboptions supported by the virtual machine
KNOWN-SUBOPTIONS-2	proxy	Suboptions supported by the proxy
UNKNOWN-SUBOPTION-RCVD-1	virtual machine	Unknown suboption received by the virtual machine
UNKNOWN-SUBOPTION-RCVD-2	proxy	Unknown suboption received by the proxy
DO-PROXY	virtual machine	Virtual machine request for proxy support
WILL-PROXY	proxy	Positive proxy response
WON'T-PROXY	proxy	Negative proxy response
VMOTION-BEGIN	virtual machine	Notification of a pending vMotion operation for a virtual machine
VMOTION-GOAHEAD	proxy	Positive proxy response
VMOTION-NOTNOW	proxy	Negative proxy response
VMOTION-PEER	virtual machine	Virtual machine request identifying the new virtual machine instance
VMOTION-PEER-OK	proxy	Proxy response accepting the new virtual machine instance
VMOTION-COMPLETE	virtual machine	Notification that vMotion operation is complete
VMOTION-ABORT	virtual machine	Notification that vMotion operation was aborted
GET-VM-VC-UUID	proxy	Proxy request for the virtual center representation of virtual machine UUID
VM-VC-UUID	virtual machine	Virtual machine response
GET-VM-NAME	proxy	Proxy request for the virtual machine name
VM-NAME	virtual machine	Virtual machine response
GET-VM-BIOS-UUID	proxy	Proxy request for the virtual machine BIOS UUID
VM-BIOS-UUID	virtual machine	Virtual machine response
GET-VM-LOCATION-UUID	proxy	Proxy request for the virtual machine location UUID
VM-LOCATION-UUID	virtual machine	Virtual machine response

Establish a VMware Telnet Extension Session

If you specify a proxy URI in the virtual serial port configuration, the virtual machine will initiate a telnet connection with the proxy. After the connection is established, the virtual machine initiates a message exchange to establish a VMware telnet extension session. The following progression describes the model for the message exchange and uses symbols to represent the codes. For the code equivalents, see the command descriptions in VMware Telnet Extension Commands for Proxy Communication.

1. Start the VMware telnet session.
   - Virtual Machine – Sends a message that indicates it will send VMware telnet extension commands to the proxy.
     `IAC WILL VMWARE-TELNET-EXT`
   - Proxy – Responds that it will receive VMware telnet extension commands.
     `IAC DO VMWARE-TELNET-EXT`
2. Identify known suboptions.

- Virtual Machine – Sends a message that indicates the set of suboption commands that it can send and receive. The suboptions are expressed as a sequence of codes that correspond to suboption commands.
  IAC SB VMWARE-TELNET-EXT KNOWN-SUBOPTIONS-1 suboptions IAC SE
- Proxy – Responds with a message that indicates the set of suboption commands that it can send and receive.
  IAC SB VMWARE-TELNET-EXT KNOWN-SUBOPTIONS-2 suboptions IAC SE

3. Negotiate proxy support.
   - Virtual Machine – Sends a request for the proxy to act as a VMware telnet extension proxy. The proxy request includes direction and URI information from the network backing for the virtual serial port. (For information about using the API to specify this information, see Create a Virtual Machine with a Network Serial Port.)
     IAC SB VMWARE-TELNET-EXT DO-PROXY *direction uri* IAC SE
   - Proxy – Responds, indicating that it will perform proxy operations.
     IAC SB VMWARE-TELNET-EXT WILL-PROXY IAC SE

Proxy Failure Detection (TCP keepalive)

The ESX Server uses a TCP keepalive timer to determine if a serial port connection between a virtual machine and a virtual serial port proxy is still active. By default, the virtual serial port keepalive time is 300 seconds.

To configure the keepalive time for a serial port connection, set the **keepIdle** advanced option for the virtual machine. The option name uses the following format:

serial *n*.keepIdle

n is an integer in a sequence starting with zero, corresponding to the sequence of serial ports on the virtual machine. Examples of **keepIdle** options include **serial0.keepIdle**, **serial1.keepIdle**, and **serial2.keepIdle**.

The **keepIdle** value is the duration in seconds on an idle connection, after which the ESX Server sends up to five keepalive probes to the virtual serial port proxy. If there is no response after five probes, the connection is considered inactive. The Server uses a keepalive interval of 1/100th of the **keepIdle** value, and a retry value of five. Using the default **keepIdle** value (300 seconds), the Server will detect a proxy failure no later than 315 seconds after the last activity (300 + 5*300/100).

vMotion and Persistent Serial Port Connections

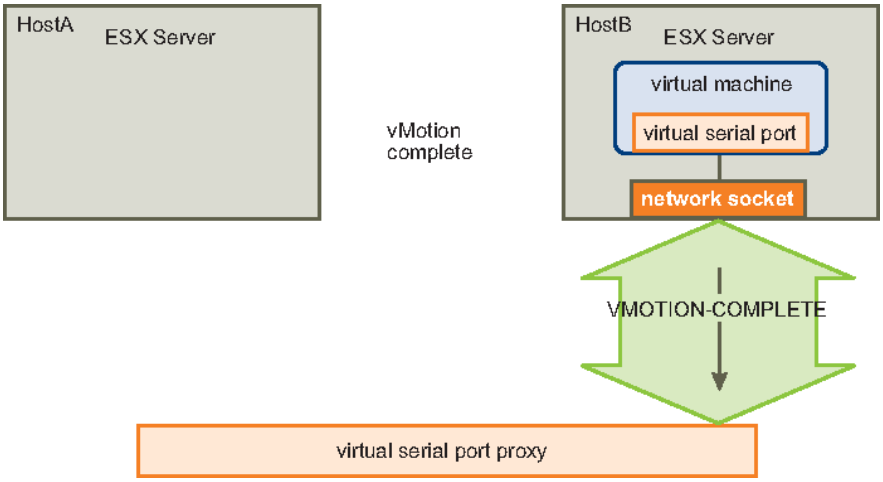
A virtual serial port proxy provides support for persistent remote serial port connections when a vCenter Server moves a virtual machine to a different host. To support the persistent connection during the vMotion operation, the proxy must support two telnet connections for the virtual machine.

A proxy buffers data while it is forwarding content from the remote system to a virtual machine. A proxy must respond to the **VMOTION-BEGIN** command, which is notification of a pending vMotion operation. ESX Server support for the VMware telnet extension allows the proxy to postpone the vMotion event until it finishes forwarding content. After sending the **VMOTION-GOAHEAD** command in response to **VMOTION-BEGIN**, the proxy must buffer any additional data it receives from the remote system. When the vMotion event is complete, the proxy continues the content transmission to the new instance of the virtual machine.

By default, the proxy must respond to a **VMOTION-BEGIN** command within 5000ms (5 seconds). If the proxy does not respond in time, the vMotion operation is terminated andthe virtual machine remains on the original server. You can specify a different time limit for the vCenter Server to wait for a **VMOTION-GOAHEAD** response from the proxy. To configure the time limit, set the **serialn.vmotionLimit** advanced option for the virtual machine. The *n* corresponds to the sequence of serial ports on the virtual machine. Specify the **vmotionLimit** value in milliseconds. Examples of **vmotionLimit** options include **serial0.vmotionLimit**, **serial1.vmotionLimit**, and **serial2.vmotionLimit**.

Figure: VMOTION-BEGIN and VMOTION-GOAHEAD shows two ESX hosts and a proxy at the start of a vMotion operation. Host A has a virtual machine that will be moved to Host B.

Figure 86: VMOTION-BEGIN and VMOTION-GOAHEAD



- Virtual machine – At the beginning of the vMotion operation, the ESX server (Host A) uses the virtual machine's serial port connection to send a **`VMOTION-BEGIN`** request to the proxy. To identify the virtual machine, the host provides an opaque **`sequence`** value for the message.

  `IAC SB VMWARE-TELNET-EXT VMOTION-BEGIN sequence IAC SE`

- Proxy – After sending pending data, the proxy replies by sending a **`VMOTION-GOAHEAD`** message to indicate that the vMotion operation can continue. The message includes the **`VMOTION-BEGINsequence`** value and an opaque **`secret`** value, which the proxy provides.

  `IAC SB VMWARE-TELNET-EXT VMOTION-GOAHEAD sequencesecret IAC SE`

After the proxy replies with a **`VMOTION-GOAHEAD`** response, the vMotion operation begins. The proxy has a single connection to the virtual machine.

The vCenter server creates a second instance of the virtual machine on Host B. When the virtual machine boots on the Host B, it configures network backing for the virtual serial port and establishes a second telnet connection with the proxy. Before continuing with the vMotion operation, the new virtual machine and the proxy renegotiate the telnet **`COM-PORT-OPTION`**. They do not renegotiate the Com Port configuration. The proxy should be prepared to support the same Com Port configuration that was established for the original telnet connection. See RFC2217 for information about the telnet Com Port control option. The proxy then maintains one telnet connection for each instance of the virtual machine.

To start the VMware telnet extension session for the new connection, the new virtual machine instance negotiates the VMware telnet extension option (**`VMWARE-TELNET-EXT`**).

Figure: VMOTION-PEER and VMOTION-PEER-OK shows the systems during the vMotion operation.

- Virtual Machine: To continue the vMotion operation, Host B sends a **`VMOTION-PEER`** message to the proxy to identify the new instance as the same virtual machine that started the vMotion operation. The message includes both the **`sequence`** and **`secret`** values to identify the virtual machine instance.

  `IAC SB VMWARE-TELNET-EXT VMOTION-PEER sequencesecret IAC SE`

- Proxy: The proxy replies with a **`VMOTION-PEER-OK`** message to indicate that it accepts the peer connection.

  `IAC SB VMWARE-TELNET-EXT VMOTION-PEER-OK sequence IAC SE`

Figure 87: VMOTION-PEER and VMOTION-PEER-OK

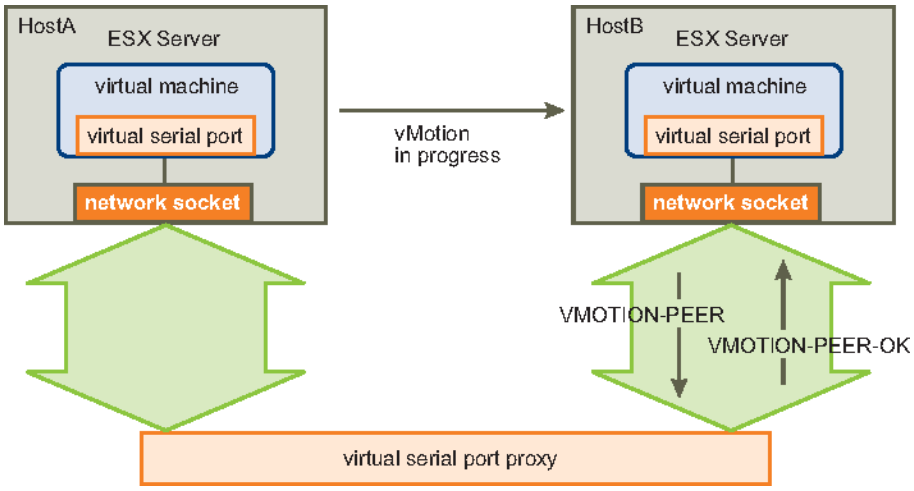
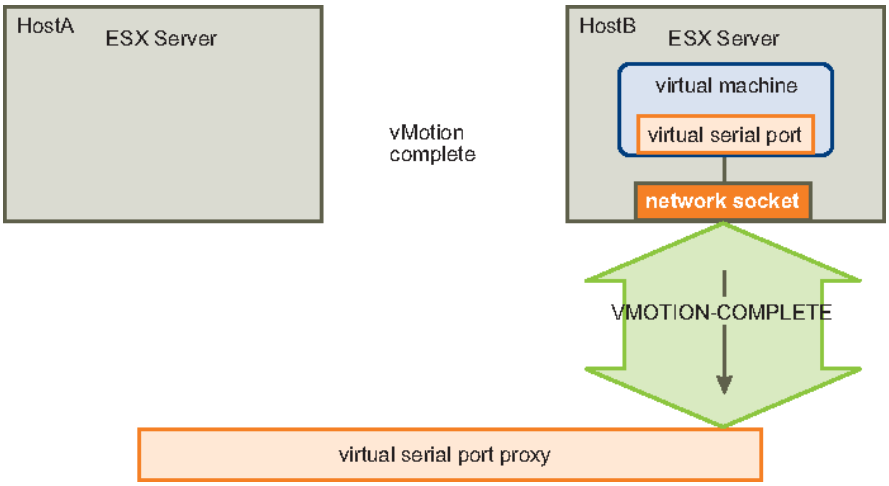


Figure: VMOTION-COMPLETE shows the completed vMotion operation. After the proxy accepts the new virtual machine instance as a peer, the ESX server on Host B sends a **`VMOTION-COMPLETE`** message to the proxy. The message indicates that the proxy should use the new connection for all traffic to the serial port. Now, the proxy can terminate the original telnet connection to Host A.

```
IAC SB VMWARE-TELNET-EXT VMOTION-COMPLETE sequence IAC SE
```

Figure 88: VMOTION-COMPLETE



VMware Telnet Extension Commands for Proxy Communication

Command Syntax

The VMware telnet extension defines a telnet proxy option for sending notification of vSphere vMotion events to a virtual serial port proxy. A VMware telnet extension command sequence uses the following general format:

```
IAC  telnet-command VMWARE-TELNET-EXT [ ext-command [ parameters] IAC SE]
```

Where

- **IAC** is the telnet escape character prefix (Interpret as Command).
- **telnet-command** is one of the following telnet commands: **WILL**, **WON'T**, **DO**, **DON'T**, and **SB**.
- **VMWARE-TELNET-EXT** is the VMware telnet extension command.
- **ext-command** is a VMware telnet extension command that indicates one of the extension suboption commands. A suboption command sequence begins with the **VMWARE-TELNET-EXT** command and is enclosed within **IAC SB ... IAC SE** telnet subnegotiation commands.
- **parameters** indicates one or more parameter values.
- **IAC SE** is the telnet sequence for completing a subnegotiation command sequence.

The following sections contain tables of VMware telnet extension commands. A table entry shows the command, the corresponding integer code, a description, and the command and code sequences. A code sequence for a VMware telnet extension command has the following characteristics.

- The code sequence begins with code 255, the telnet escape character prefix IAC (Interpret as Command).
- VMware telnet extension commands use two- or three-byte code sequences.
- The **VMWARE-TELNET-EXT** command code is 232.
- Subnegotiation command sequences include begin (**SB**) and end (**SE**) codes (250 and 240).

For example, the **DO-PROXY** command requires the following code sequence with embedded arguments for direction ("S" for server) and URI:

```
IAC SB  VMWARE-TELNET-EXT DO-PROXY direction         uri           IAC SE

255 250       232           70       "S" "telnet://example.com" 255 240

[R]VM
```

VMware Telnet Extension Option

Table : vmware-telnet-extension-option.dita#ID-3873-00000281 lists the commands that establish the VMware telnet extension communication between a virtual machine and a proxy. The virtual machine initiates the connection by sending a **WILL VMWARE-TELNET-EXT** message. If the proxy accepts by responding with a **DO VMWARE-TELNET-EXT** message, the virtual machine and the proxy can continue negotiation.

Table 33:

Command	Codes
WILL VMWARE-TELNET-EXT A virtual machine request indicating that it is willing to send VMware telnet extension messages to the proxy. **IAC WILL VMWARE-TELNET-EXT** 255 251 232	251 232
WON'T VMWARE-TELNET-EXT A virtual machine notification that it is unwilling to send VMware telnet extension messages to the proxy. The virtual machine can use this command to turn off the extension option. To resume message exchange, the virtual machine and proxy must negotiate to turn on the option again (**WILL VMWARE-TELNET-EXT** and **DO VMWARE-TELNET-EXT**).	252 232

Table continued on next page

Continued from previous page

Command	Codes
IAC WON'T VMWARE-TELNET-EXT 255 252 232	
DO VMWARE-TELNET-EXT A proxy response that indicates it is willing to receive VMware telnet extension messages from the virtual machine. **IAC DO VMWARE-TELNET-EXT** 255 253 232	253 232
DON'T VMWARE-TELNET-EXT A proxy response that indicates it is unwilling to receive VMware telnet extension messages. The proxy can send this message at any time to stop remote serial port communication. To resume message exchange, the virtual machine and proxy must negotiate to turn on the option again (**WILL VMWARE-TELNET-EXT** and **DO VMWARE-TELNET-EXT**). **IAC DON'T VMWARE-TELNET-EXT** 255 254 232	254 232

Option Subnegotiation

The **VMWARE-TELNET-EXT** telnet option requires various suboptions to support remote serial port access. The virtual machine and proxy use subnegotiation to indicate the command sets that they can support. Subnegotiation identifies all commands for negotiation as well as commands for proxy support, vMotion notification, and virtual machine identification. Table 2-2 lists the commands for option subnegotiation.

Command	Code

KNOWN-SUBOPTIONS-1 suboptions
A virtual machine notification that indicates the **VMWARE-TELNET-EXT** suboptions that it can generate and accept. *suboptions* is a sequence of one or more codes, each of which corresponds to a VMware telnet extension command.
 	
If the virtual machine sends this message more than once during a session, the list of suboptions is always the same. The presence of a command in the list does not imply that the virtual machine will use the command during the session.
 	
`IAC SB VMWARE-TELNET-EXT KNOWN-SUBOPTIONS-1 suboptions IAC SE`
 	
`255 250 232 0 suboptions 255 240`
	0

KNOWN-SUBOPTIONS-2 suboptions A proxy response that indicates the **VMWARE-TELNET-EXT** suboptions that it can generate and accept. *suboptions* is a sequence of one or more codes, each of which corresponds to a VMware telnet extension command. The proxy must support this command.	1

Table continued on next page

Continued from previous page

Command	Code

If the proxy sends this message more than once during a session, the list of suboptions must be the same each time.
 	
`IAC SB VMWARE-TELNET-EXT KNOWN-SUBOPTIONS-2 suboptions IAC SE`
 	
`255 250 232 1 suboptions 255 240`	

Unknown Command Response

Table lists the commands to indicate reception of an unknown command. The proxy must send **`UNKNOWN-SUBOPTION-RCVD-2`** any time it receives a command that it does not support.

Command	Code

`UNKNOWN-SUBOPTION-RCVD-1 code`
A virtual machine response to indicate that it received an unknown suboption command from the proxy. *`code`* identifies the unknown suboption.
 	
`IAC SB VMWARE-TELNET-EXT UNKNOWN-SUBOPTION-RCVD-1 code IAC SE`
 	
`255 250 232 2 code 255 240`
	2

`UNKNOWN-SUBOPTION-RCVD-2 code`
A proxy response to indicate that it received an unknown **`sub`**option command from the virtual machine. *`code`* identifies the unknown suboption. The proxy must support this command.
 	
`IAC SB VMWARE-TELNET-EXT UNKNOWN-SUBOPTION-RCVD-2 code IAC SE`
 	
`255 250 232 3 code 255 240`
	3

Proxy Operation

Table lists the extension commands to negotiate proxy support.

Command	Code
`DO-PROXY direction serviceUri` **`A`** virtual machine request for the proxy to do forwarding operations over the connection. The virtual machine sends a **`DO-PROXY`** request before any content is exchanged between the virtual machine and the remote system. **`direction`** and **`serviceURI`** correspond to the **`VirtualSerialPortURIBackingInfo`** properties in the virtual serial port configuration. (See the *`vSphere API Reference`* for information about the **`VirtualSerialPortURIBackingInfo`** data object.)	70

Table continued on next page

Continued from previous page

Command	Code

direction is either “C” (client) or “S” (server). If the direction indicates that the virtual machine is acting as a client, **serviceUri** identifies a remote system and the proxy initiates a connection with the remote system. If the direction indicates that the virtual machine is acting as a server, **serviceUri** identifies the network socket on the host on which the virtual machine runs. In this case, the virtual machine will accept a connection initiated by a remote system (using the proxy).
If possible, the proxy should also handle port control options, as specified by RFC 2217.
`IAC SB VMWARE-TELNET-EXT DO-PROXY direction uri IAC SE`
`255 250 232 70 direction uri 255 240`	

WILL-PROXY
A proxy response to indicate that it will do forwarding operations.
`IAC SB VMWARE-TELNET-EXT WILL-PROXY IAC SE`
`255 250 232 71 255 240`
	71

WON’T-PROXY
A proxy response to indicate that it will not do forwarding operations.
`IAC SB VMWARE-TELNET-EXT WON’T-PROXY IAC SE`
`255 250 232 73 255 240`
	73

vMotion Notification

Table lists the commands that hosts and proxy use during a vMotion operation. In the following descriptions, the *source* host contains the original virtual machine instance, and the *target* host is the host on which the new instance is created. A host uses the virtual machine serial port connection with the proxy to send vMotion messages.

Command	Code

VMOTION-BEGIN sequence
A source host request to indicate that a vMotion operation is pending. The operation requires a VMOTION-GOAHEAD response to start. The intent is to flush all pending data transmission on the connection.
The host provides an opaque **sequence** value. All subsequent vMotion messages related to the connection between the virtual machine and this proxy use this sequence value.
`IAC SB VMWARE-TELNET-EXT VMOTION-BEGIN sequence IAC SE`
`255 250 232 40 sequence 255 240`
	40

VMOTION-GOAHEAD sequence secret	41

Table continued on next page

Continued from previous page

Command	Code

A proxy response to indicate that all queued data has been transmitted, and that the vMotion operation can start. The proxy sends this message only in response to a **VMOTION-BEGIN** request.
 	
The proxy returns the sequence value that it received in the **VMOTION-BEGIN** request. The proxy also creates and returns an opaque secret value. The virtual machine uses the sequence and secret to identify itself when it establishes a proxy connection from the new host. See the description of **VMOTION-PEER**.
 	
`IAC SB VMWARE-TELNET-EXT VMOTION-GOAHEAD sequence secret IAC SE`
 	
`255 250 232 41 sequence secret 255 240`	

VMOTION-NOTNOW sequence
A proxy response to indicate that the vMotion operation should not start at this time. This message includes the sequence value that the proxy obtained in the **VMOTION-BEGIN** request. The proxy can send this message only in response to a **VMOTION-BEGIN** request.
 	
`IAC SB VMWARE-TELNET-EXT VMOTION-NOTNOW sequence IAC SE`
 	
`255 250 232 43 sequence 255 240`
	43

VMOTION-PEER sequence secret
A target host request to indicate that this connection belongs to the virtual machine associated with **sequence** and **secret**. The sequence and secret identify the duplicate connections between the proxy and virtual machine instances during a vMotion operation.
 	
The original host generates the sequence value for the **VMOTION-BEGIN** message. The proxy generates the secret value for the **VMOTION-GOAHEAD** message.
 	
`IAC SB VMWARE-TELNET-EXT VMOTION-PEER sequence secret IAC SE`
 	
`255 250 232 44 sequence secret 255 240`
	44

VMOTION-PEER-OK sequence
A proxy response to indicate that the secret in the **VMOTION-PEER** message was accepted. The proxy sends this message only in response to a **VMOTION-PEER** request.
 	
`IAC SB VMWARE-TELNET-EXT VMOTION-PEER-OK sequence IAC SE`
 	
`255 250 232 45 sequence 255 240`
	45

VMOTION-COMPLETE sequence
A target host notification that the vMotion operation is complete. The proxy must send all subsequent communication for the virtual machine on the connection associated with this **VMOTION-COMPLETE** message. The host sends this message only after sending a **VMOTION-PEER** message.
 	
`IAC SB VMWARE-TELNET-EXT VMOTION-COMPLETE sequence IAC SE`
 	
`255 250 232 46 sequence 255 240`
	46

VMOTION-ABORT	48

Table continued on next page

Continued from previous page

Command	Code

A source host notification that the vMotion operation failed. The proxy must send all communication for the virtual machine on the connection associated with this **VMOTION-ABORT** message. The host can send **VMOTION-ABORT** only if it has sent **VMOTION-BEGIN** previously on this connection.
`IAC SB VMWARE-TELNET-EXT VMOTION-ABORT IAC SE`
`255 250 232 48 255 240`	

Virtual Machine Identification

Table lists the suboption commands for virtual machine identification.

Command	Code

VM-VC-UUID vm-uuid
A virtual machine response that contains its universally unique identifier.
`IAC SB VMWARE-TELNET-EXT VM-VC-UUID vm-uuid IAC SE`
`255 250 232 80 vm-uuid 255 240`
	80

GET-VM-VC-UUID
A proxy request for the virtual machine's universally unique identifier.
`IAC SB VMWARE-TELNET-EXT GET-VM-VC-UUID IAC SE`
`255 250 232 81 255 240`
	81

VM-NAME vm-name
A virtual machine response that contains its name.
`IAC SB VMWARE-TELNET-EXT VM-NAME vm-name IAC SE`
`255 250 232 82 vm-name 255 240`
	82

GET-VM-NAME
A proxy request for the virtual machine's name.
`IAC SB VMWARE-TELNET-EXT GET-VM-NAME IAC SE`
`255 250 232 83 255 240`
	83

VM-BIOS-UUID vm-uuid
A virtual machine response that contains its universally unique identifier stored in the virtual machine BIOS.
`IAC SB VMWARE-TELNET-EXT VM-BIOS-UUID-IS vm-uuid IAC SE`
`255 250 232 84 vm-uuid 255 240`
	84

GET-VM-BIOS-UUID	85

Table continued on next page

Continued from previous page

Command	Code

A proxy request for the virtual machine's universally unique identifier stored in its BIOS.
`IAC SB VMWARE-TELNET-EXT GET-VM-BIOS-UUID IAC SE`
`255 250 232 85 255 240`	

`VM-LOCATION-UUID vm-uuid`
A virtual machine response that contains its universally unique identifier.
`IAC SB VMWARE-TELNET-EXT VM-LOCATION-UUID vm-uuid IAC SE`
`255 250 232 86 vm-uuid 255 240`
	86

`GET-VM-LOCATION-UUID`
A proxy request for the virtual machine's location universally unique identifier.
`IAC SB VMWARE-TELNET-EXT GET-VM-LOCATION-UUID IAC SE`
`255 250 232 87 255 240`
	87

Developing and Deploying vSphere Solutions, vServices, and ESX Agents

Developing and Deploying vSphere Solutions, vServices, and ESX Agents provides information about how to extend the function of vSphere by integrating solutions with `ExtensionManager`, vCenter Server Extensions, vServices, and vSphere ESX Agent Manager.

Intended audience

This information is intended for anyone who wants to deploy vSphere extensions as vSphere solutions by integrating with `ExtensionManager`, vCenter Server Extensions, vServices, and ESX Agent Manager features. This publication does not describe how to develop vSphere extensions in detail.

Related documentation

For information about how to develop vSphere Client extensions, see the *Developing Remote Plug-ins with the vSphere Client SDK* documentation.

For information about the data objects that the vSphere ESX Agent Manager provides, see the *vSphere ESX Agent Manager API Reference* documentation.

For information about developing vSphere applications, see the *vSphere Web Services SDK Programming Guide* and the *VMware vSphere API Reference* documentation.

How to develop vSphere solution

You can add functions to vSphere by developing software applications that you register as vCenter Server extensions. A vSphere solution is an extension that registers with vCenter Server and implements some or all of the extension features of the vSphere API.

Developing and Deploying vSphere Solutions, vServices, and ESX Agents uses the terms solution and extension interchangeably.

A vSphere solution is an object or program that you create by extending specific classes in the vSphere Web Services API. After you register your solution with the instance of the `ExtensionManager` managed object associated with your vCenter Server, you see your solution under **Menu › Administration › Solutions › vCenter Server Extensions** of your vSphere Client.

You can create a vService solution to provide access for a specific application to connect to a service across the network.

You can create an ESX Agent to extend the functions of an ESXi host and provide additional services that a vSphere solution requires.

The vService Manager and ESX Agent Manager are pre-built solutions that are part of the vCenter Server Extensions functionality within the vCenter Server.

The extension features in the vSphere Web Services API include functions so that you can perform the following tasks:

- Register permissions, faults, and events for an extension.
- Identify and prevent manual operations on virtual machines and vApps that extensions deploy.
- Integrate extensions with ExtensionManager.
- Store data about extensions in the vSphere database.
- Provide user interface plug-ins that extend the vSphere Client.

To use the extension functions of the vSphere API, a solution must register itself with the `ExtensionManager` that runs in a vCenter Server instance. By registering with `ExtensionManager`, a solution can access the extension features of the vSphere extension API.

You can develop solutions that add functions to the standard functions of vCenter Server. You can deploy a solution as an Open Virtualization Format (OVF) package, with optional VMware vSphere Installation Bundles (VIB). You can also install solutions by using an installer, such as Windows Installer (MSI) or RPM Package Manager. Most of the extension functions in the vSphere API are independent of the technology that you use to deploy a solution. If you deploy a solution by using OVF, you can use the vCenter Extension vService to simplify the registration of the solution with vCenter Server.

vCenter Server 8.0 provides built-in solutions.

- vSphere ESX Agent Manager
- vService Manager

vSphere ESX Agent Manager and vService Manager are part of a standard vCenter Server installation. These solutions appear in vCenter Server Extensions and ExtensionManager with any other solutions that register with ExtensionManager.

Download the vSphere ESX Agent Manager SDK

The ESX Agent Manager SDK is part of the vSphere SDK.

The ESX Agent Manager SDK includes the WSDL file definitions for the vSphere ESX Agent Manager API and the ESX Agent Manager API reference documentation.

1. Download the vSphere Management SDK 8.0 Update 1 bundle, `VMware-vSphere-SDK-8.0.1-`*`build_number`*`.zip`, from the VMware Developer site at https://developer.vmware.com/home.
2. Unzip the vSphere SDK to a convenient location in your development environment.
3. Navigate to the ESX Agent Manager SDK at the following location in the vSphere SDK.

   `VMware-vSphere-SDK-8.0.1-`*`build_number`*`\SDK\eam`

4. Copy the contents of `VMware-vSphere-SDK-8.0.1-`*`build_number`*`\SDK\eam` to a folder where you can modify the files.

   For example, copy the files to *`eam_work_folder`*.

You downloaded and located the ESX Agent Manager SDK, and made a copy of its contents to work on.

Create your solution.

What is in the vSphere Extension APIs

The vSphere SDK provides a set of APIs that you can use to register extensions with vCenter Server. To develop vSphere extensions, the most important object in the vSphere Extension API is the `ExtensionManager` managed object.

Use the `ExtensionManager` managed object in your vCenter Server instance to register a new extension. Extensions can add new objects to the vCenter inventory. Extensions define tasks, events, and faults that relate to actions that the solution performs on the objects, the events that occur in the extension, and the problems that the objects encounter. You can also use `ExtensionManager` to add user interface elements to the vSphere Client to allow users to interact with the objects that your extension provides to vCenter Server.

vCenter Server extensions can have both a client and a server component, or they can consist of just a client component or just a server component. Each instance of an extension is represented by an `Extension` data object.

You can indicate that an extension manages the virtual machines or vApps that it creates by setting the `managedBy` property in the `ManagedEntityInfo` data object for that virtual machine or vApp. When you register an extension with vCenter Server, you can define an icon that represents the virtual machines that the extension manages. Virtual machines that an extension manages display the icon that you define in the inventory of virtual machines in the vSphere Client. The

vCenter Server shows a warning if users try to perform manual operations on a virtual machine or vApp that the extension manages.

ExtensionManager adds data objects to the vSphere Extension APIs that integrate extensions as vCenter solutions that you can manage.

For information about how to develop vSphere Client extensions, see the *Developing Remote Plug-ins with the vSphere Client SDK* documentation.

For information about developing vSphere applications, see the *vSphere Web Services SDK Programming Guide* and the *VMware vSphere API Reference* documentation.

How to use the vCenter Server Extensions

The vCenter Server Extensions allows you to monitor and interact with solutions that are registered with a vCenter Server instance.

The vCenter Server Extensions shows four standard tabs for each running solution. The tabs list the virtual machines that a solution deploys and manages, show the status, name, company URL, and version of the solution.

A solution can also use the **Solutions** portlet on the **Summary** tab to add links to Web pages that provide some functionality specific to the solution. For example, the pages can be used to allow users to configure the solution, or to provide access to the functions of the solution.

Standard Tabs in the vCenter Server Extensions

vCenter Server Extensions displays standard tabs for each solution that is running on a vCenter Server instance.

Table 34: Standard tabs in the vCenter Server Extensions

Tab	Description
Summary	General information about the solution, including its name, vendor, and version.
Monitor	Filters all vSphere events and lists only the system activities that are related to the specific solution.
Configure	Displays one or more ESX Agents, the services they contain, and actions you can perform on them.
VMs	Information about the virtual machines and vApps that the solution deploys. You can perform all operations applicable for a VM and vApp.

What are the vSphere ESX agents and agencies

A vSphere ESX agent is a virtual machine and an optional vSphere Installation Bundle (VIB) that extends the functions of an ESXi host to provide additional services that a vSphere solution requires.

For example, a solution might require a particular network filter or firewall configuration to function. A solution can use an ESX agent to connect to the vSphere Hypervisor and extend the host with functions specific to that solution. For example, the ESX agent can filter network traffic, act as a firewall, or gather other information about the virtual machines on the host.

ESX agent virtual machines are similar to services in Windows or Linux. They start when the operating system starts and they stop when it shuts down. The behavior of ESX agent virtual machines is transparent to the user. A vSphere host

reaches the ready state when the ESXi operating system has started and all ESX agent virtual machines have been provisioned and powered on.

To integrate an agent with vSphere ESX Agent Manager and extend the capabilities of an ESXi server, an ESX agent must be packaged as an OVF or a VIB module.

ESX agencies act as containers for ESX agents. ESX agencies aggregate information about the agents that they manage. For example, ESX agencies provide an overview of the ESX agents that they contain by aggregating all the issues that relate to the ESX agents.

vSphere ESX Agent Manager

vSphere ESX Agent Manager automates the process of deploying and managing vSphere ESX agents.

The services that ESX Agent Manager provides include out-of-the-box integration of agents with vSphere features such as DRS, `AddHost`, High Availability, DRM, and maintenance mode. Each of these features can be difficult to integrate manually. ESX Agent Manager allows you to monitor the health of ESX agents. You can also block users from performing certain operations on ESX agents that might affect the virtual machines that use them. For example, ESX Agent Manager can prevent an ESX agent virtual machine from being powered off or moved from an ESXi host that contains other virtual machines that use that agent.

ESX Agent Manager adds an **ESX Agencies** entry to the table of contents under the **Configure** tab of vSphere ESX Agent Manager. The **ESX Agencies** tab shows information about running agencies, lists any orphaned ESX agents, and logs information about the ESX agents that ESX Agent Manager manages.

What are vServices

A vService is a service that a solution provides to specific applications that run inside virtual machines and vApps. A solution can provide several types of vServices. Virtual machines or vApps can have dependencies on several types of vServices.

A vService is similar to a virtual hardware device upon which virtual machines and vApps can depend. Instead of providing a piece of virtual hardware, vServices typically provide access to a service across a network. By providing a vService, a solution can expose application-aware services to virtual machines and vApps. For example, a vService can provide a backup service or a logging service to virtual machines and vApps.

Virtual machines that use a vService have a vService dependency. If you mark a virtual machine as having a vService dependency, the virtual machine cannot power on unless you bind it to a provider of that vService.

If you deploy virtual machines by using Open Virtualization Format (OVF), you can specify in the OVF descriptor that the virtual machine depends on a vService. If you configure a virtual machine to have dependency on a vService, the solution that provides the vService receives notifications when specific events occur on that virtual machine. The vService can modify the OVF environment and OVF descriptor of the virtual machines and vApps that depend on it. With vServices, you can encode application-specific information in the OVF package about how a virtual machine or vApp interacts with a specific solution. You can also add solution-specific information to code running inside the guest operating system.

vService Manager

vService Manager allows you to manage the set of vServices that extensions provide and to configure virtual machines and vApps to be dependent on these vServices.

You access vService Manager from vCenter Server Extensions. You can add dependencies on vServices to virtual machines and vApps directly in the vSphere Client.

vCenter Extension vService

The vCenter Extension vService is a standard vService that vSphere 8.0 provides. With the vCenter Extension vService you can register virtual machines as vCenter Server extensions with minimal user interaction.

Typically, you register an extension with vCenter Server by running an MSI installer or an installation script in which you enter the credentials of a vCenter Server administrator and the connection parameters of the vCenter Server system. You can avoid these manual steps by using the vCenter Extension vService to register extensions in vCenter Server. vCenter Extension vService performs the following functions.

- Provides the virtual machine with the connection parameters and a login token of the vCenter Server system on which you install the extension.
- Registers the vCenter extension certificate of the solution with vCenter Server.

To deploy an extension using the vCenter Extension vService, you must deliver the vCenter extension in a virtual appliance that you deploy using OVF.

How do I create vSphere solutions

The vSphere SDK provides a set of APIs that you can use to create extensions for vCenter Server. The key SDK objects for extension development are `ExtensionManager` and `Extension`.

Use the `ExtensionManager` managed object in your vCenter Server instance to register a new extension. Extensions can add new objects to the vCenter inventory. Extensions define tasks, events, and faults that relate to actions that the solution performs on the objects, the events that occur in the extension, and the problems that the objects encounter. You can also use `ExtensionManager` to add user interface elements to the vSphere Client to allow users to interact with the objects that your extension provides to vCenter Server.

ExtensionManager adds data objects to the vSphere Extension APIs that integrate extensions as vCenter solutions that you can manage.

For information about how to develop vSphere Client extensions, see the *Developing Remote Plug-ins with the vSphere Client SDK* documentation.

For information about developing vSphere applications, see the *vSphere Web Services SDK Programming Guide* and the *VMware vSphere API Reference* documentation.

Contents and structure of an extension

A vSphere solution must register with vCenter Server as an extension. If you register a solution with vCenter Server, you can mark virtual machines as belonging to that solution, and integrate it with ExtensionManager and ESX Agent Manager.

vCenter Server extensions can have both a client and a server component, or they can consist of just a client component or just a server component. Each instance of an extension is represented by an `Extension` data object.

An extension can define events, tasks, faults, and privileges for performing operations on the objects that the solution exposes to vCenter Server, or on existing vCenter Server objects.

- Events inform users about occurrences in the solution.
- Tasks are operations that the extension performs.
- Faults signal errors in the extension to the end users.
- Privileges define which users or user groups can access the objects in the extension and perform tasks on them.

Solutions typically include Web pages that appear in vSphere Client. You can use the pages to configure the solution and to manage the objects that the solution exposes.

An extension is typically a Web application that runs in an application server or a script that vCenter Server accesses through a URL. You can implement the Web server features by using any of the following programming languages.

- Java Servlets or Java Server Pages (JSP)
- Microsoft Active Server Pages (ASP.NET)
- Common Gateway Interface (CGI) scripting
- Static or dynamic HTML pages

An extension performs the following types of tasks:

- Establishes the connection to vCenter Server.
- Registers the extension with `ExtensionManager`.
- Deploys virtual machines or vApps by using Open Virtualization Format (OVF).
- Deploys VMkernel modules or ESXi applications as vSphere Installation Bundles (VIB).
- Exposes the functions of the product that the extension integrates into vCenter Server.
- Provides a user interface to access, configure, and use the extension.

To integrate an extension as a solution with ExtensionManager, the solution must be a valid extension and it must implement the `Extension.shownInSolutionsManager` property. The solution can also provide information about itself to vCenter Server in the `ExtSolutionManagerInfo` object.

The requirements for integrating a solution with ESX Agent Manager are slightly more restrictive than for integrating with ExtensionManager. For information about requirements, see How do I integrate solutions with vSphere ESX Agent Manager.

For information about how to develop vSphere Client extensions, see the *Developing Remote Plug-ins with the vSphere Client SDK* documentation.

For information about developing vSphere applications, see the *vSphere Web Services SDK Programming Guide* and the *VMware vSphere API Reference* documentation.

Key objects in the vSphere API for extension and solution development

The vSphere API contains objects that are key for developing extensions and for integrating the extensions with ExtensionManager.

Table 35: Key objects in the vSphere API for extension and solution development

Object	Description
`ExtensionManager`	You call `ExtensionManager.registerExtension()` to register an `Extension` instance with `ExtensionManager`.
`Extension`	Information about extension instances, including properties that you set in an `ExtSolutionManagerInfo` object to integrate an extension as a solution in ExtensionManager.
`ExtensionClientInfo`	Information about the client side of an extension.
`ExtensionEventTypeInfo`	Information about the types of events that occur on the objects in the extension.
`ExtensionFaultTypeInfo`	Information about the types of problems that the objects in the extension encounter.
`ExtensionHealthInfo`	Information about the health of a solution.
`ExtensionPrivilegeInfo`	Information about access privileges for the extension.
`ExtensionResourceInfo`	Information about resource files that contain localizable user interface text and messages that appear in vSphere Client in different locales.

Table continued on next page

Continued from previous page

Object	Description
`ExtensionServerInfo`	Information about the server side of an extension.
`ExtensionTaskTypeInfo`	Information about the types of tasks that the objects perform in the extension.
`ExtExtendedProductInfo`	Information about the product that the extension exposes to vCenter Server.
`ExtManagedEntityInfo`	Information about the objects that the extension manages.
`ExtSolutionManagerInfo`	Information that the solution exposes about itself.
`ExtSolutionManagerInfoTabInfo`	Information about the tabs that a solution adds to its Summary page.

How do I register an extension with vCenter Server

You must provide information about an extension when you register it with vCenter Server.

- A description of the extension
- A unique identifier for the extension
- Version information
- URLs to any server or client interfaces that the extension exposes
- An email address for the extension administrator
- Any additional metadata that the extension defines, for example, URLs to the the company and product Web sites, the managed object reference (MoRef) of the virtual machine or vApp in which the solution is running, and so on
- Definitions of the tasks, events, faults, and privileges that the extension adds to vCenter Server
- Localization data for the task, event, fault, and privilege descriptions
- A public key which vCenter Server uses to register the extension in the registry of the vCenter Server system on which it is running.

Create the program that manages the extension

An extension communicates with vCenter Server across a network, so you can use any programming language to create the program that manages an extension.

.

The product that you are exposing as an extension to vCenter Server determines the way you create the program that will manage your extension. The managing program must implement `Extension` to provide the information that vCenter Server requires to register the extension.
For example, you can create a solution that defines its server side in a `MyManager` class.
The `MyManager.java` can performs the following tasks for your solution.

- Sets up your solution by obtaining the values for the extension key, vCenter Server connection, IP addresses, and port configuration from the `mysolution.properties` file.
- Secures the connection to vCenter Server by using TLS.
- Defines methods to construct the URLs through which to access the server side of the extension.
- Implements the `Extension` object to register your solution as an extension with vCenter Server.
- Integrates your solution as a solution with ExtensionManager.
- Connects the solution to the ESX Agent Manager.
- Instantiates the class that defines the ESX agents that the solution deploys.
- Starts the solution in vCenter Server.
- Defines a task to unregister the solution from vCenter Server.

1. Create a program to manage your extension using the programming language of your choice.

   For example, you can create a `MyManager` class to manage the extension. `MyManager` can implement the Spring Framework API.

   ```
   public class MyManager implements InitializingBean {

   }
   ```

2. Create an instance of the `Extension` data object with which to register the extension with vCenter Server. The `MyManager` class can define an internal method that instantiates an `Extension`.

   ```
   public class MyManager implements InitializingBean {


     [...]

     private Extension createExtensionObject() {

       Extension extension = new Extension();

     [...]

     }

   }
   ```

Connect the extension to vCenter Server.

Connect the extension to vCenter Server

You must provide information about the vCenter Server instance to which you connect an extension. Set the details of the connection to vCenter Server in the client-side stub of the extension.

- .

  NOTE
  If you deploy your extension using the Open Virtualization Format (OVF), you can integrate it with the vCenter Extension vService. The vCenter Extension vService automates the process of registering extension with vCenter Server, so you do not need to provide any connection parameters. See How do I integrate an extension with the vCenter Extension vService.

To connect an extension to vCenter Server, provide the following information to the client-side stub of the connection.

- A username and password for a vCenter Server administrator account, if you do not use the vCenter Extension vService
- The extension key for the extension
- A reference to the `SessionManager` instance in the vCenter Server

You can define the connection to vCenter Server in the `MyVimConnection` class.
The `MyManager` class can perform the following functions.

- Implement the `MyVimConnection` class to establish the connection to vCenter Server when the My Solution starts.
- Uses the Spring framework to obtain the connection information from the `mysolution.properties` file that you configure when you set up My Solution.

- Passes the connection property values to `MyVimConnection`.

1. Create an instance of the `ManagedObjectReference` data object to define the connection to the extension.
   The `MyVimConnection` class creates a `ManagedObjectReference` object of type `ServiceInstance`, named `_siRef`.

```
public MyVimConnection(String host, int port) {
 [...]
 _siRef = new ManagedObjectReference();
 _siRef.setType("ServiceInstance");
 _siRef.setValue("ServiceInstance");
}
```

2. Define methods to get and set the vCenter Server host, ports, username, password, connection timeout, and session cookie.
   The `MyVimConnection` constructor defines methods to obtain the host, ports, username, password, connection timeout, and session cookie from the information that you set in the `mysolution.properties` file.
3. Connect to vCenter Server by obtaining the `SessionManager` managed object for the vCenter Server.
   `MyVimConnection.java` defines a method named `connect()`. The `connect()` method defines a standard connection to vCenter Server that uses WSDL. The following segment shows the calls to the `SessionManager.login()` and `SessionManager.loginExtensionByCertificate()` methods that establish the connection to vCenter Server. The `_stub` variable is an instance of `VimPortType`, `_sc` is a `ServiceContent` object, and `_siRef` is the `ManagedObjectReference` object of type `ServiceInstance`.

```
private synchronized void connect() {
  [...]
  try {
      [...]
      VimService locator = new VimService(wsdlURL, new QName("urn:vim25Service",
                                                              "VimService"));

      _stub = locator.getVimPort();
      [...]

      _sc = _stub.retrieveServiceContent(_siRef);

      ManagedObjectReference sessionManager = _sc.getSessionManager();

      if (_extensionKey == null) {
        _stub.login(sessionManager, _username, _password, null);
```

```
        }
        else {
          _stub.loginExtensionByCertificate(sessionManager, _extensionKey, null);
        }
      [...]
       _connectionStatus = ConnectionStatus.Connected;
      } catch (Exception e) {
          _logger.error(e, e);
      }
    }
```

4. Register the extension by calling the `ExtensionManager.registerExtension()` method. `MyVimConnection` defines a `registerExtension()` method that implements `ExtensionManager.registerExtension()`. `MyManager` calls `MyVimConnection.registerExtension()` after it has set the properties for the extension.

```
  public void registerExtension(Extension ex) {
    try {
         Extension findExtension = _stub.findExtension(_sc.getExtensionManager(),
                                                       ex.getKey());
         if (findExtension == null) {
             _stub.registerExtension(_sc.getExtensionManager(), ex);
         } else {
             _stub.updateExtension(_sc.getExtensionManager(), ex);
         }
         _stub.setExtensionCertificate(_sc.getExtensionManager(),
                                       ex.getKey(), null);
         } catch (Exception e) {
             _logger.error(e, e);
         }
    }
```

You registered an extension with vCenter Server.

Provide an extension key with which to register the extension with vCenter Server.

Set the extension key

Every extension that you register with vCenter Server must have a unique extension key that vCenter Server uses to identify the extension.

.

To ensure uniqueness, you can use the Java package-naming convention for the key value, for example `com.yourcompany.yourextension`. Your solution can set the extension key by defining the `extensionKey=com.mycompany.mysolution` property in the `mysolution.properties` file. The `MyManager.java` class picks up this property by calling the `Extension.setKey()` method.
If you change the extension key in the `extensionKey` property in `mysolution.properties`, you can see the changed name in the vSphere Client.
You can see the extensions that you register with `ExtensionManager` in the vCenter Managed Object Browser at `https://<vcenter_server_ip_address>/mob/?moid=ExtensionManager`

1. Call the `Extension.setKey()` method to set the extension key.
   `MyManager.java` sets the extension key to the `extensionKey` variable that the `mysolution.properties` file defines.

```
public class MyManager implements InitializingBean {

  [...]

  public final String EXTENSION_KEY;

  [...]

     EXTENSION_KEY = extensionKey;

  [...]

}

  [...]

  private Extension createExtensionObject() {

    Extension extension = new Extension();

    extension.setKey(EXTENSION_KEY);

  [...]

  }
```

You set the extension key for an extension.

Provide information about the extension to vCenter Server.

Set the extension product information

You provide product information about an extension by setting properties when you instantiate the `Extension` object in the program that manages the extension. The product information that you set appears in the vSphere Client.

.

You can instantiate the `Extension` data object in the `MyManager.java` class. Your solution can set the `Extension` product information properties directly in `MyManager.java`, but you can set the property values in configuration files that the program that manages the extension accesses.

1. Provide a description for the extension by creating an instance of the `Description` data object and passing it to `Extension`.
```
  private Extension createExtensionObject() {

    Extension extension = new Extension();

    [...]

    Description description = new Description();

    description.setLabel("My Solution");

    description.setSummary("This extension represents my solution.");

    extension.setDescription(description);

    [...]

  }
```
2. Provide a version number for the extension by calling the `Extension.setVersion()` method.
```
private Extension createExtensionObject() {

    Extension extension = new Extension();

    [...]

    extension.setVersion("0.1");

    [...]

  }
```
3. Change the value of the version property in `MyManager.java`.
4. Provide information about the vendor of the extension by calling the `Extension.setCompany()` method. You can set the `vendor` property by adding the following line of code to `MyManager.java`.
```
private Extension createExtensionObject() {

    Extension extension = new Extension();

    [...]

    extension.setVersion("0.1");

    extension.setCompany("My Company");

    [...]

  }
```
5. Provide URLs to Web pages for the product and for the vendor of the extension by creating an instance of the `ExtExtendedProductInfo` data object and passing it to `Extension`. You can set the `companyUrl` property for `ExtExtendedProductInfo` by calling the `ExtExtendedProductInfo.setCompanyUrl()` method in `MyManager.java`:
```
private Extension createExtensionObject() {

    Extension extension = new Extension();
```

```
        [...]
        ExtExtendedProductInfo extExtendedProductInfo = new
        ExtExtendedProductInfo();
        extExtendedProductInfo.setCompanyUrl("www.mycompany.com");
        extExtendedProductInfo.setProductUrl("www.mycompany.com/myproduct");
        extension.setExtendedProductInfo(extExtendedProductInfo);
        [...]
    }
```

6. Save, build and deploy your solution.
7. View your changes in the vSphere Client.

   If you edit the values of the `version`, `company`, `companyUrl`, and `productUrl` properties, your changes appear in the **Summary** tab of your solution.

You provided product information about an extension by setting properties in the `Extension` and `ExtExtendedProductInfo` data objects.

Set the extension name and localization information.

Set the extension name and localization resources

When you develop extensions, the vSphere Client can appear in different languages in different locales. You can set the information that appears in the vSphere Client, for example, the extension name, as resources that can be translated.

.

You provide the information that requires translation in an `ExtensionResourceInfo` data object. You can add an `ExtensionResourceInfo` object for every locale that your extension supports. You set an array of `ExtensionResourceInfo` objects in the `resourceList` property of the `Extension` instance that defines your extension.
You provide onscreen messages and labels to `ExtensionResourceInfo` in a key and value pairing that you add to a `KeyValue` array in the `ExtensionResourceInfodata` property. You can set the values for the `KeyValue` pair directly in the `ExtensionResourceInfo` object, or you can refer to entries in resource files that contain the message text in different languages, according to the locale in which vSphere is running.
You provide a two-character ISO-639 language ID for the `KeyValuelocale` property, and set the `module` property to the type of resource to which this locale applies. For example, you can set the module value to `task`, `event`, `auth`, or `extension`, depending on whether the messages that the resource contains relate to tasks, events, privileges, or extensions.

1. Create an instance of the `ExtensionResourceInfo` data object.
   For example, in `MyManager.java` you can instantiate `ExtensionResourceInfo` in the implementation of `Extension`.

```
private Extension createExtensionObject() {
    Extension extension = new Extension();
    [...]
    ExtensionResourceInfo extensionResourceInfo = new ExtensionResourceInfo();
```

```
    [...]
}
```

2. Set the locale and module properties for the `ExtensionResourceInfo` object.
   For example, in `MyManager.java` you can set the default locale to `en` and apply this locale to the `Extension` instance, `extension`.

```
private Extension createExtensionObject() {
    Extension extension = new Extension();
    [...]
    ExtensionResourceInfo extensionResourceInfo = new ExtensionResourceInfo();
    extensionResourceInfo.setLocale("en");
    extensionResourceInfo.setModule("extension");
    [...]
}
```

3. Provide the data to the `ExtensionResourceInfo` in the form of a `KeyValue` array.

   The `label` property is a property of the `Description` object, that `Extension` implements, and defines the name of the extension as it appears in the vSphere Client.

   For example, in `MyManager.java` you can add the text `My Solution` as the value of the `label` property.

```
private Extension createExtensionObject() {
    Extension extension = new Extension();
    [...]
    ExtensionResourceInfo extensionResourceInfo = new ExtensionResourceInfo();
    extensionResourceInfo.setLocale("en");
    extensionResourceInfo.setModule("extension");
    KeyValue keyValue = new KeyValue();
    keyValue.setKey(EXTENSION_KEY + ".label");
    keyValue.setValue("My Solution");
    [...]
}
```

4. Call the `ExtensionResourceInfo.getData()` method to add the `KeyValue` array that contains the localization data to the `data` property of the `ExtensionResourceInfo` object.

```
private Extension createExtensionObject() {
    Extension extension = new Extension();
    [...]
    ExtensionResourceInfo extensionResourceInfo = new ExtensionResourceInfo();
    extensionResourceInfo.setLocale("en");
```

```
    extensionResourceInfo.setModule("extension");

    KeyValue keyValue = new KeyValue();

    keyValue.setKey(EXTENSION_KEY + ".label");

    keyValue.setValue("My Solution");

    extensionResourceInfo.getData().add(keyValue);

    [...]

}
```

5. Add another `KeyValue` object to the `ExtensionResourceInfodata` property that adds a description of the extension for a given locale.
   For example, you can add the following description to `MyManager.java` in a `KeyValue` object named `keyValue_summary`.

```
private Extension createExtensionObject() {

    Extension extension = new Extension();

    [...]

    ExtensionResourceInfo extensionResourceInfo = new ExtensionResourceInfo();

    extensionResourceInfo.setLocale("en");

    extensionResourceInfo.setModule("extension");

    KeyValue keyValue = new KeyValue();

    keyValue.setKey(EXTENSION_KEY + ".label");

    keyValue.setValue("My Solution");


    KeyValue keyValue_summary = new KeyValue();

    keyValue_summary.setKey(EXTENSION_KEY + ".summary");

    keyValue_summary.setValue("This is a brief description of My Solution.");


    extensionResourceInfo.getData().add(keyValue);

    extensionResourceInfo.getData().add(keyValue_summary);

    [...]

}
```

6. Call the `Extension.getResourceList()` method to pass the `ExtensionResourceInfo` object to the `Extension` instance.

```
private Extension createExtensionObject() {

    Extension extension = new Extension();

    [...]
```

```
    ExtensionResourceInfo extensionResourceInfo = new ExtensionResourceInfo();
    extensionResourceInfo.setLocale("en");
    extensionResourceInfo.setModule("extension");
    KeyValue keyValue = new KeyValue();
    keyValue.setKey(EXTENSION_KEY + ".label");
    keyValue.setValue("My Solution");

    KeyValue keyValue_summary = new KeyValue();
    keyValue_summary.setKey(EXTENSION_KEY + ".summary");
    keyValue_summary.setValue("This is a brief description of My Solution.");

    extensionResourceInfo.getData().add(keyValue);
    extensionResourceInfo.getData().add(keyValue_summary);

    extension.getResourceList().add(extensionResourceInfo);
    [...]
}
```

7. Add more `ExtensionResourceInfo` instances to provide localized text that displays when the extension runs in different locales.
   For example, you can add an `ExtensionResourceInfo` instance to `MyManager.java` to provide a French translation of the extension name.

```
private Extension createExtensionObject() {
    Extension extension = new Extension();
    [...]
    ExtensionResourceInfo extensionResourceInfo = new ExtensionResourceInfo();
    extensionResourceInfo.setLocale("en");
    extensionResourceInfo.setModule("extension");
    KeyValue keyValue = new KeyValue();
    keyValue.setKey(EXTENSION_KEY + ".label");
    keyValue.setValue("My Solution");

    KeyValue keyValue_summary = new KeyValue();
    keyValue_summary.setKey(EXTENSION_KEY + ".summary");
    keyValue_summary.setValue("This is a brief description of My Solution.");
```

```
        extensionResourceInfo.getData().add(keyValue);
        extensionResourceInfo.getData().add(keyValue_summary);

        ExtensionResourceInfo extensionResourceInfo_FR = new ExtensionResourceInfo();
        extensionResourceInfo_FR.setLocale("fr");
        extensionResourceInfo_FR.setModule("extension");
        KeyValue keyValue_FR = new KeyValue();
        keyValue_FR.setKey(EXTENSION_KEY + ".label");
        keyValue_FR.setValue("Ceci est une brève description de Ma solution.");

        extensionResourceInfo_FR.getData().add(keyValue_FR);

        extension.getResourceList().add(extensionResourceInfo);
        extension.getResourceList().add(extensionResourceInfo_FR);
        [...]
    }
```

8. Save your changes, build, and deploy your solution.
   If you edit the `label` value, the extension appears in vCenter Server Extensions with the new name. If you added an `ExtensionResourceInfo` object for a different locale, the localized text that you added appears in vCenter Server Extensions when you connect your solution to a vCenter Server instance that runs in that locale.

You added localizable message resources to the extension, so that onscreen messages and labels that your extension provides can appear in different languages in different locales.

Identify the virtual machines or vApps that an extension manages

If an extension manages virtual machines or vApps, you can identify those virtual machines or vApps as being managed by that extension. You can add icons to the objects that the extension manages that appear in the vCenter Server inventory, and warn users if they try to perform actions on those objects.

You have a vCenter Server extension that manages virtual machines or vApps.

For example, your solution can create ESX agent virtual machines. The ESX agent virtual machines that it manages appear in the vCenter Server inventory with an icon that identifies them as ESX agents. A panel appears in the **Summary** tab for those virtual machines that identifies ESX Agent Manager as the solution that manages them. If you attempt to perform an operation directly on an ESX agent virtual machine, you see a warning that instructs you to perform the operation by using ESX Agent Manager, rather than by performing it manually on the virtual machine. For the list of operations that trigger warnings when you try to perform them on a virtual machine or vApp that an extension manages, see What are the operations that trigger warnings from extensions.
You identify a virtual machine or vApp as belonging to an extension by setting the `managedBy` property in the `VirtualMachineConfigSpec` or `VAppConfigSpec` implementations that define the virtual machines or vApps that the extension manages. You set the `managedBy` property to a `ManagedByInfo` object.

You identify the extension that manages a virtual machine or vApp by setting the `extensionKey` property in the `ManagedByInfo` object. You specify different types of virtual machine or vApp that an extension manages by setting the `type` property in `ManagedByInfo`.
If you set the `ManagedByInfotype` property in the virtual machine or vApp definition, you can pass this value to the `ExtManagedEntityInfo` implementation in the extension definition. `ExtManagedEntityInfo` applies descriptions and icons to all the virtual machines or vApps of this type that the extension manages.
For information about configuring and deploying virtual machines programmatically, see the *vSphere Web Services SDK Programming Guide* and the *VMware vSphere API Reference*.

1. In the program that defines the virtual machines or vApps that an extension deploys, create an instance of `VirtualMachineConfigSpec` or `VAppConfigSpec`.
   For example, you can instantiate `VirtualMachineConfigSpec`.
   ```
   VirtualMachineConfigSpec configSpec = new VirtualMachineConfigSpec();
   ```
2. Create an instance of `ManagedByInfo`.
   ```
   ManagedByInfo managedByInfo = new ManagedByInfo();
   ```
3. Set the `ManagedByInfoextensionKey` property to the extension key of the extension that deploys the virtual machines or vApps.

   Use the extension key that you define in the implementation of `Extension` in your extension.

   ```
   managedByInfo.setExtensionKey("com.mycompany.myextension");
   ```
4. Set the `ManagedByInfotype` property to identify the virtual machine or vApp as being of a certain type.

   Set the `type` property to a descriptive name for this type of virtual machine. In the implementation of `Extension` that manages the extension, you can apply icons and descriptions to all virtual machines or vApps of this type that the extension deploys.

   ```
   managedByInfo.setType("my VM type");
   ```
5. Pass the `ManagedByInfo` instance to the `managedBy` property of the `VirtualMachineConfigSpec` or `VAppConfigSpec` implementation.
   ```
   configSpec.setManagedBy(managedByInfo);
   ```

You set the `managedBy` properties in a virtual machine or vApp definition, to identify the virtual machines or vApps as being of a certain type and as belonging to an extension.

Set the types of virtual machines or vApps that an extension manages by implementing `ExtManagedEntityInfo`.

What are the operations that trigger warnings from extensions

If an extension deploys virtual machines or vApps, you can configure them as being virtual machines or vApps that the extension manages. If users attempt to perform operations on these virtual machines or vApps, they see a warning.

You configure a virtual machine or vApp as belonging to an extension by implementing `ManagedByInfo` in the configuration of that virtual machine or vApp.

The following operations result in a warning if a user attempts to run them from the vSphere Client on a virtual machine or vApp that an extension manages:

- `PowerOn`
- `PowerOff`
- `Suspend`

- Reset
- RebootGuest
- ShutdownGuest
- StandbyGuest
- Edit
- Clone
- Templatize
- ConvertToVirtualMachine
- Deploy
- CloneTemplateToTemplate
- CloneVmToTemplate
- TakeSnapshot
- GotoSnapshot
- StartRecording
- Remove
- CreateSecondaryVm
- DisableFaultTolerance
- TurnFaultToleranceOff
- MigrateHost
- Migrate
- RelocateDatastore
- Rename

Set the types of the virtual machines or vApps that the extension manages

You can set properties in the `ExtManagedEntityInfo` data object to define how the vSphere Client displays the different types of virtual machines and vApps that an extension deploys.

Verify that you have set the `managedBy` property in the configuration of the virtual machines or vApps that an extension deploys. See Identify the virtual machines or vApps that an extension manages.

If you set the `ManagedByInfotype` property in a virtual machine or vApp configuration, you can configure an extension to apply certain properties to all the virtual machines or vApps of that type that it deploys. You can implement the `ExtManagedEntityInfo` data object to apply a common description and an icon that appears in the vCenter inventory for all virtual machines of that type that the extension deploys.
You add an array of `ExtManagedEntityInfo` objects to an extension by setting the `ExtensionmanagedEntityInfo` property. If your extension deploys different types of virtual machine or vApp, you can add one `ExtManagedEntityInfo` object to the array for each type of virtual machine or vApp. By creating several instances of `ExtManagedEntityInfo` with different properties, you can differentiate the different types of virtual machines or vApps that the extension deploys in the vCenter inventory.

1. Create an instance of `ExtManagedEntityInfo` in the implementation of `Extension` in the program that manages the extension.
   ```
   Extension extension = new Extension();

   ExtManagedEntityInfo extManagedEntityInfo = new ExtManagedEntityInfo();
   ```
2. Set the `ExtManagedEntityInfotype` property to the `type` value that you set in the `ManagedByInfo` property in the program that defines the virtual machine or vApp to deploy.
   For example, set the `type` property to the same value as the `type` property of the `ManagedByInfo` object.
   ```
   extManagedEntityInfo.setType("my VM type");
   ```

3. Add the `ExtManagedEntityInfo` instance to the array of `ExtManagedEntityInfo` instances in the `ExtensionmanagedEntityInfo` property.
```
extension.getManagedEntityInfo().add(extManagedEntityInfo);
```

You set the types of virtual machine or vApp that the extension manages. When you set different types of virtual machine or vApp in an extension, you can change how those types of object appear in the vSphere inventory.

Set a description for the types of object that the extension manages.

Set the description for a type of virtual machine or vApp that a solution manages

If a solution deploys different types of virtual machine or vApp, you can provide a description for each type by setting the `ExtManagedEntityInfodescription` property.

- Verify that you have set the `managedBy` property in the configuration of the virtual machines or vApps that an extension deploys. See Identify the virtual machines or vApps that an extension manages.
- Verify that you have created an instance of `ExtManagedEntityInfo` in the program that defines an extension.

The text that you set in the `ExtManagedEntityInfodescription` property allows you to add a description to the types of virtual machines or vApps that your solution deploys. If a solution deploys different types of virtual machine, you can create several `ExtManagedEntityInfo` instances, each with a different description.

1. Call the `ExtManagedEntityInfo.setDescription()` method to set the `ExtManagedEntityInfodescription` property for a type of virtual machine or vApp that the solution deploys.
```
Extension extension = new Extension();

ExtManagedEntityInfo extManagedEntityInfo = new ExtManagedEntityInfo();

extManagedEntityInfo.setType("my VM type");

extManagedEntityInfo.setDescription("Description of this type of virtual machine or vApp.");

extension.getManagedEntityInfo().add(extManagedEntityInfo);
```

You added a description to all virtual machines or vApps of a certain type that your solution deploys. The description appears in the vSphere Client when the solution registers with vCenter Server.

Unregister the extension from vCenter Server

When you develop an extension, the best practice is to provide a means of unregistering the extension from vCenter Server.

.

You unregister an extension from vCenter Server by calling the `ExtensionManager.unregisterExtension()` method. To unregister an extension, you pass to `unregisterExtension()` the `ManagedObjectReference` instance that identifies the `ExtensionManager` for the vCenter Server and the extension key for the extension.

1. Define a method that unregisters the extension by calling `ExtensionManager.unregisterExtension()`. For example, in `MyManager.java` you can define a method named `cleanup()` that unregisters your solution from vCenter Server. In the `MyVimConnection.java` class you can define the connection to vCenter Server. `MyManager.java` creates an instance of `MyVimConnection`, `_myvimConnection`, with which it connects to

vCenter Server. The `MyVimConnection.getStub()` method obtains the port on which the vCenter Server is running.

```
public void cleanup()
  throws NotFoundFaultMsg, RuntimeFaultFaultMsg {
    [...]
    _myvimConnection.getStub().unregisterExtension(
                                    _myvimConnection.getExtensionManager(),
                                    EXTENSION_KEY);
}
```

You defined a method to unregister an extension from vCenter Server.

How do I integrate solutions with vSphere ESX Agent Manager

vSphere ESX Agent Manager is a standard vCenter Server solution that allows other solutions to deploy, monitor, and manage ESX agents on ESXi hosts.

ESX Agent Manager performs the following functions:

- Provisions ESX agent virtual machines for solutions.
- Monitors changes to the ESX agent virtual machines and their scope in vCenter Server.
- Reports configuration issues in the ESX agents to the solution.
- Integrates agent virtual machines with vSphere features such as Distributed Resource Scheduler (DRS), Distributed Power Management (DPM), vSphere High Availability (HA), fault tolerance, maintenance mode, and operations such as adding and removing hosts to and from clusters.

A solution can add functions to an ESXi host by deploying an ESX agent virtual machine and an optional vSphere Installation Bundle (VIB) that provide this function.

Every vCenter Server instance contains a running ESX Agent Manager.

Benefits of integrating solutions with ESX Agent Manager

Integrating a solution with ESX Agent Manager has several benefits and eliminates potential problems that can occur when you deploy ESX agent virtual machines.

Monitoring solutions

When you integrate a solution with ESX Agent Manager, users can monitor the health of agent virtual machines in the vSphere Client. The solution can monitor ESX agents by using the ESX Agent Manager API. Integrating a solution with ESX Agent Manager also helps you to add functions to ESXi hosts.

User privileges

By integrating a solution with ESX Agent Manager, you can limit certain operations on ESX agent virtual machines to certain types of user. For example, you can block all users from powering off an ESX agent unless they have the **EAM.Modify** privilege.

Integration with vSphere features

When you develop solutions that deploy virtual machines that extend the function of ESXi hosts, integrate the solution with ESX Agent Manager. If you do not integrate the solution and ESX Agent Manager, vSphere does not detect that virtual machines are ESX agent virtual machines, Virtual machines that ESX Agent Manager deploys integrate with vSphere features in ways that other virtual machines do not. For example, if an ESXi host implements vSphere High Availability (VMware HA) and that host stops running unexpectedly, normally all the virtual machines running on that host migrate to another ESXi host. An ESX agent virtual machine should not migrate to another host. Features such as vSphere Distributed Resource Scheduler (DRS), the Add Host to Cluster operation, vSphere Distributed Power Management (DPM), and maintenance mode can also create problems if they modify or migrate ESX agents. Similarly, if one of these features migrates or modifies an ESX agent that a solution requires, that solution can stop running.

ESX Agent Manager is compatible with the VMware HA, DRS, `AddHost`, DPM, and maintenance mode features. By integrating a solution with ESX Agent Manager, you can deploy ESX agent virtual machines that modify the function of ESXi hosts and implement VMware HA, DRS, `AddHost`, DPM, and maintenance mode. ESX Agent Manager ensures that solutions that deploy ESX agents integrate correctly with these vSphere features.

ESX Agent virtual machines and High Availability

If you have a configuration with a very large virtual appliance per host, a best practice is to use an ESX Agent VM as a container for the appliance, because vCenter Server High Availability (HA) feature gives special priority to ESX Agent VMs.

The HA feature will power on an ESX Agent VM before any other VMs.

If a failover host policy is in use, the HA feature allows Agent VMs to be powered on with the failover hosts.

The HA feature does not include ESX Agent VMs in admission control calculations. Therefore, if you want to use a VM that is not included in the slot size for the 'host failures to tolerate' admission control policy, you can use an Agent VM.

To take advantage of these special priorities, you must include an ESX Agent VM on each host in a cluster.

Table 36: Interaction between vSphere features and ESX Agent Manager

vSphere Feature	Interaction with ESX Agent Manager
VMware HA	• If a host stops unexpectedly, VMware HA does not start an ESX agent virtual machine on another host, unless a failover policy is in use. • When a host restarts, VMware HA restarts the ESX agent virtual machines before it restarts other virtual machines. • ESX agent virtual machines are not included in slot size for admission control calculations.
Distributed Resource Scheduler (DRS)	• ESX Agent Manager pins ESX agent virtual machines to the hosts on which they are running. DRS does not move ESX agent virtual machines between hosts in a cluster. • ESX Agent Manager blocks DRS from moving virtual machines that are not ESX agents to hosts in clusters where ESX agents that the virtual machines require are not available.
Add Host to Cluster operation	When a user adds a host to a cluster of hosts that require an ESX agent, ESX Agent Manager deploys the required ESX agents on the new host.

Table continued on next page

Continued from previous page

vSphere Feature	Interaction with ESX Agent Manager
vSphere Distributed Power Management (DPM)	DPM can put a host into standby mode even when ESX agent virtual machines are present. DPM powers off ESX agent virtual machines only after it has moved all the other virtual machines to another host and put the host into standby mode.
Maintenance mode	When a host enters maintenance mode, ESX Agent Manager powers off any ESX agent virtual machines and restarts the ESX agents when the host exits maintenance mode.

vSphere ESX Agent Manager architecture

ESX Agent Manager is an intermediary between vCenter Server and a solution. ESX Agent Manager provides a WSDL based SOAP API that you can use to create solutions that deploy and monitor ESX agent virtual machines on ESXi hosts, and integrate the ESX agents with Distributed Resource Scheduler (DRS), Distributed Power Management (DPM), High Availability (HA), and other vSphere features.

For an overview of the architecture of ESX Agent Manager and how it integrates with vCenter Server, see the introduction to the *`ESX Agent Manager API Reference`*.

Configuration components of ESX agencies

For a solution to integrate with ESX Agent Manager, it must register with vCenter Server as an extension, and create an ESX agency that defines the configuration of the solution and the ESX agents that the solution deploys.

ESX agencies encapsulate the ESX agents that they deploy from a solution. When you create an ESX agency, you must provide configuration information that the solution applies to the individual ESX agents that it deploys.

You must set the scope of the ESX agents that the agency defines and the ESX agency goal state. You provide a URL to an Open Virtualization Format (OVF) file that defines the ESX agent virtual machines. You also provide a URL to an optional vSphere installation bundle (VIB) that provides a function that extends an ESXi host, for example, a VMkernel module or a custom ESXi server application.

ESX agency scope

When you create an ESX agency, you must define the scope of the agency. You define the scope of an ESX agent in terms of compute resources, namely standalone hosts or clusters.

For example, ESX agents can run on standalone ESXi hosts or on clusters of ESXi hosts.

A solution must obtain from vCenter Server the set of compute resource instances on which to deploy ESX agents. For example, you can implement a query in your solution that obtains a list of all the hosts that are running in a vCenter Server instance. Users can then select the hosts on which to deploy ESX agents.

You set the initial ESX agency scope in the ESX agency configuration. As the solution runs, the scope of the ESX agency can change. For example, your solution runs on compute resources that users select from your solution Configuration page. As users select different compute resources on which to run the solution, the scope of the ESX agency changes as it adds the MoRefs of the selected compute resources to the scope.

ESX agency goal state

When you create an ESX agency, you define a goal state for that ESX agency. The goal state of an ESX agency defines the state in which all the ESX agents in that ESX agency should be when the solution is running correctly.

If the ESX agency is not in its goal state, ESX Agent Manager attempts to remediate the issues that it discovers, so that the agency can achieve or return to its goal state.

Table 37: ESX agent goal states

Goal State	Description
Enabled	Solution has deployed all ESX agents in the ESX agency successfully, and they are running with a heartbeat. For ESX agent virtual machines without VMware Tools installed, which do not return a heartbeat, ESX Agent Manager considers them to be enabled when they are powered on.
Disabled	Solution has deployed all ESX agents in the ESX agency successfully, but they are not running.
Undeployed	Solution has removed all ESX agents in the ESX agency from all the compute resources in the ESX agent scope.

ESX agency status

When you create an agency, you define the goal state of the ESX agents that the agency contains. The status of an ESX agency reflects the situation of the ESX agency in relation to its goal state.

ESX Agent Manager notifies a solution of the status of its ESX agencies. ESX Agent Manager defines three possible statuses for ESX agencies.

Table 38: ESX agent status

Status	Description
Green	The ESX agency has achieved its goal state.
Yellow	ESX Agent Manager is working to move the ESX agency into its goal state.
Red	The ESX agency cannot achieve its goal state. The solution or the vSphere administrator must actively resolve one or more issues. ESX Agent Manager reports the issues to the solution in the agency runtime. ESX Agent Manager can resolve certain issues if you select the agency, click the vertical ellipsis and click **Resolve All Issues**, for example if an ESX agent is powered off.

ESX Agent Manager sets the ESX agency status to yellow until the ESX agency reaches its goal state. This can either mean that the deployment of the ESX agents is still ongoing, or that ESX Agent Manager is attempting to remediate issues. You define the remediation of issues in the implementation of the solution, or you can leave the remediation of issues to the vSphere administrator, who can resolve the issue by using the ESX Agent Manager user interface.

The criteria on which ESX Agent Manager determines that the status of an ESX agency is green depends on the goal state of that ESX agency.

Table 39: Criteria for meeting the goal state of an ESX agency

ESX Agency Goal State	Criteria for Green Status
Enabled	ESX Agent Manager has downloaded the ESX agent binaries from the OVF and VIB URLs, deployed the ESX agent virtual machines, installed the VIB, started the ESX agent virtual machines, and received heartbeats from all of them.
Disabled	ESX Agent Manager has downloaded the ESX agent binaries from the OVF and VIB URLs, deployed the ESX agent virtual machines, and installed the VIB.
Undeployed	ESX Agent Manager has powered off and deleted all the ESX agent virtual machines and uninstalled the VIB.

If the ESX agency cannot reach its goal state, ESX Agent Manager reports the issues to the solution and sets the status to red.

NOTE
If the scope of an ESX agency is empty, there are no compute resources onto which to deploy ESX agents, so no ESX agents are deployed. In this case, ESX Agent Manager determines that the ESX agency has performed correctly, and sets the status to green.

Status of ESX agency scope changes

When a solution updates its ESX agency scope, ESX Agent Manager provisions new ESX agents on new hosts and removes ESX agents from hosts that are no longer in the scope.

When the scope of an ESX agency changes, vSphere ESX Agent Manager detects the change and displays the status of the ESX agents in the scope and operations that are in progress. You can add functions in the solution definition to remediate the issues, or the solution can report the issues so that administrators can fix them manually.

When a solution adds or removes compute resources from the ESX agency scope, the status of the ESX agency changes as the solution works to achieve the goal state of the ESX agency.

Table 40: Status of ESX agency scope changes

Scope Change	Current ESX Agency State	ESX Agency Status
Add compute resource.	Undeployed	Unchanged. New ESX agents always start in the undeployed state.
Remove compute resource.	Undeployed	Unchanged. Old ESX agents that the solution has not removed yet are pending undeployment.
Add compute resource.	Disabled	Yellow, until the solution deploys the ESX agents on the new compute resource.
Remove compute resource.	Disabled	Yellow, until the solution undeploys the ESX agents from the compute resource.
Add compute resource.	Enabled	Yellow, until the solution deploys the ESX agents on the new compute

Table continued on next page

Continued from previous page

Scope Change	Current ESX Agency State	ESX Agency Status
		resource, starts the ESX agents, and receives heartbeats.
Remove compute resource.	Enabled	Yellow, until the solution stops and undeploys the ESX agents from the compute resource.

Requirements for itegrating a solution with ESX Agent Manager

To integrate with ESX Agent Manager, a solution must meet certain requirements.

To integrate a solution with ESX Agent Manager, you must develop the solution according to the following requirements:

- The solution must be a vCenter Server extension that implements the `Extension` data object and registers with `ExtensionManager`.
- Use Open Virtualization Format (OVF) to package ESX agent virtual machines or vApps. ESX Agent Manager only supports the deployment of virtual machines using OVF.
- Use HTTP or HTTPS to publish OVF files to ESX Agent Manager.
- Use vSphere installation bundles (VIB) to add functions to ESXi hosts, for example to add VMkernel modules or custom ESX Server applications to ESXi hosts.
- Use HTTP or HTTPS to publish VIB files to ESX Agent Manager.
- Use vCenter Server Compute Resources to define the ESX agent scope.

How do I authenticate against ESX Agent Manager

vCenter Server handles the authentication of ESX Agent Manager clients, so a solution must first log in to vCenter Server before it can call the methods of the ESX Agent Manager API.

ESX Agent Manager registers itself with the vCenter Server reverse proxy in the `eam` namespace. You find the ESX Agent Manager API under `eam/sdk`. You must direct all ESX Agent Manager API calls in your solution to *`vcenter_server_ip_address`*`/eam/sdk`.

ESX Agent Manager grants access to clients that are also clients of the vCenter Server instance in which ESX Agent Manager is an extension. All vCenter Server extensions that have an active vCenter Server session have access to ESX Agent Manager. These extensions can create ESX agencies and monitor existing ESX agencies in ESX Agent Manager. The user name that the extension provides is the extension key that the extension sets in the vCenter Server`ExtensionManager` instance.

Users can access the ESX Agent Manager user interface only if they have the **Eam.View** and **Eam.Modify** privileges. Solutions by definition have all vCenter Server privileges, but the **Eam.View** and **Eam.Modify** privileges limit what users can do in the vSphere Client.

Table 41: ESX Agent Manager privileges

Privilege	Permitted Actions
`Eam.View`	Users can monitor all running ESX agencies in ESX Agent Manager.
`Eam.Modify`	Users can modify the ESX agencies, for example by powering off ESX agent virtual machines.

When you make an HTTP request to call an object in the ESX Agent Manager API for the first time, you must set the VMware SOAP session cookie. You set it to the value of the VMware SOAP session cookie of the vCenter Server HTTP connection. You must add a function in your solution to obtain the SOAP session cookie from vCenter Server when the solution establishes the connection to vCenter Serverr. The cookie remains set for the duration of the ESX Agent Manager session.

How do I configure the availability of ESX agent virtual machines

You can configure ESX agencies so that they only mark ESX agent virtual machines as available after the client of the agency has performed additional configuration after provisioning or powering on the ESX agent virtual machine.

You configure an ESX agency in the `Agency.ConfigInfo` object. You can set two options in `Agency.ConfigInfo` that set when ESX agents can be marked as available.

Table 42: ESX agent availability options

Option	Description
Boolean `manuallyMarkAgentVmAvailableAfterProvisioning`	If set to true, the client of this ESX agency must manually mark the agent as ready after the ESX agent virtual machine has been provisioned. This is useful if the client of this solution performs some extra reconfiguration of the ESX agent virtual machine before it is powered on.
Boolean `manuallyMarkAgentVmAvailableAfterPowerOn`	If set to true, the client of this ESX agency must manually mark the agent as ready after the ESX agent virtual machine has been powered on. In this case, DRS will not regard the ESX agent virtual machine as ready until the client has marked the agent as ready.

Monitoring and resolving ESX agent issues

With the ESX Agent Manager API, you can obtain status and state data about all the ESX agencies that a solution creates. The status of an ESX agency reflects the status of all the ESX agents that are running in the ESX agency.

Solutions can only access data about ESX agencies that they create. A solution cannot access the data of ESX agencies that other solutions create on the same host.

You can use ESX Agent Manager to monitor and track many of the issues that can occur when you deploy ESX agent virtual machines. ESX Agent Manager handles updates to the deployment specifications of ESX agent virtual machines. ESX Agent Manager responds to changes in the vCenter Server instance in which it runs. If ESX Agent Manager encounters a conflict and cannot satisfy the requirements of an update to a deployment specification, it reports the conflict to the solution as an issue and shows the issue in the ESX Agent Manager user interface.

The issues that the ESX Agent Manager monitors relate to the goal state and status of ESX agencies and agents. If ESX Agent Manager encounters an issue, it sets the status of the ESX agency or agent to red and alerts the solution about the issue. ESX Agent Manager can resolve certain issues if you click the vertical ellipsis icon and then select **Resolve Issues** in the ESX Agent Manager user interface.

How does the ESX Agent Manager remediate issues

ESX Agent Manager reports to the solution or to the vSphere administrator issues that can occur during the deployment of ESX agent virtual machines. ESX Agent Manager can resolve certain issues, if the solution or the administrator requests that it does so.

ESX Agent Manager defines standard issues that can occur during the deployment of ESX agent virtual machines. For example, deployment of ESX agents can fail because the network, datastore, or resource pool settings are incorrect.

Certain errors occur only in the certain goal states. Other errors can occur in any ESX agency goal state.

Table 43: Remediation of issues by ESX Agent Manager (partial list)

Issue	Goal State	Resolved by ESX Agent Manager?
`AgentVmPoweredOffIssue`	Enabled	Yes
`AgentVmWrongDatastoreIssue`	Any	No
`AgentVmWrongNetworkIssue`	Any	No
`AgentVmWrongResourcePoolIssue`	Any	Yes
`AgentVmWrongFolderIssue`	Any	Yes
`InsufficientSpaceOnDatastoreIssue`	Enabled	No

How do I integrate a solution with ESX Agent Manager

You can integrate your solution with ESX Agent Manager.

For more information about the objects that your solution can implement, see the *`vSphere ESX Agent Manager API Reference`*.

Connect a solution to ESX Agent Manager

To connect a solution to ESX Agent Manager, you must obtain connection information from the vCenter Server in which you run the solution. You pass a reference to the `EsxAgentManager` instance that is running in vCenter Server to the solution.

- .
- Obtain the following information from the vCenter Server instance on which you run a solution:
  - The vCenter Server host ID.
  - The HTTPS proxy port for vCenter Server.
  - The session cookie for the current user session in which the solution is logged in as an extension.
  - The managed object reference (MoRef) of the ESX Agent Manager instance that is running in the vCenter Server instance.

You pass the connection information to a solution by calling the appropriate methods from the vSphere API.

> **NOTE**
> If you integrate your solution with the vCenter Extension vService, the vService obtains the vCenter Server connection information for you. See How do I integrate an extension with the vCenter Extension vService.

You can define the connection to vCenter Server in the `MyVimConnection.java` class and the connection to ESX Agent Manager in the `MyEamConnection.java` class.
The `MyVimConnection` class defines methods to obtain the vCenter Server host ID, HTTPS proxy, and the session cookie. The `MyManager.java` class implements `MyVimConnection` to connect to the vCenter Server on which your solution is running. See Connect the extension to .

1. Establish a connection to the vCenter Server instance on which the solution runs.
   Your solution can establish the connection to vCenter Server in the `MyVimConnection.java` class.
2. Create an instance of the `ManagedObjectReference` object of type `EsxAgentManager`.
   The `MyEamConnection.java` class can define a constructor that creates a MoRef named `_myeamRef` and a method named `getEsxAgentManager()` that returns the MoRef to the solution.

```
public MyEamConnection(String vcHost, int vcHttpsProxyPort, String sessionCookie) {
 [...]

 _myeamRef = new ManagedObjectReference();
 _myeamRef.setType("EsxAgentManager");
 _myeamRef.setValue("EsxAgentManager");
}

 [...]

 public ManagedObjectReference getEsxAgentManager() {
   return _myeamRef;
}
```

3. Connect to the ESX Agent Manager with the URL to the ESX Agent Manager service and the session cookie for the current session in which the solution is logged as an extension.
   The `MyEamConnection` class can define a `connect()` method that connects to the ESX Agent Manager service.

```
public void connect() {
  if (_isConnected) {
    return;
  }

  try {
    [...]

    String myeamUrl = "https://" + _vcHost + ":" + _vcHttpsProxyPort + "/eam/sdk/";
    [...]
    Map<String, List<String>> map = new HashMap<String, List<String>>();
    map.put("Cookie", Collections.singletonList(_sessionCookie));
    ((BindingProvider) _stub).getRequestContext()
            .put(MessageContext.HTTP_REQUEST_HEADERS, map);
    _isConnected = true;
  }
}
```

You obtained the connection details for vCenter Server and connected a solution to ESX Agent Manager.

Configure ESX agencies and agents to deploy from the solution.

Configure an ESX agency and ESX agents

You must define the configuration of the ESX agencies in the implementation of the solution. You set the configuration for ESX agencies and ESX agents in the `AgencyConfigInfo` and `AgentConfigInfo` data objects.

.

When you create ESX agencies, you provide the `AgencyConfigInfo` object with an array of `AgentConfigInfo` objects for each version of ESXi on which the agency deploys ESX agents. You also define the name of the ESX agency and of the ESX agents and the scope of the ESX agency in the `AgencyConfigInfo` object.
You define the deployment of the ESX agent virtual machines in the `AgentConfigInfo` object. You set the following information in the `AgentConfigInfo` object.

- A URL to the Open Virtualization Format (OVF) file from which to deploy the ESX agent.
- A URL to an optional vSphere Installation Bundle (VIB) that adds function to ESXi, for example a VMkernel module or a custom ESXi application that you developed.

The URL to the ESX agent virtual machine OVF and the URL to an optional VIB must lead to a server that ESX Agent Manager can access. ESX Agent Manager downloads the ESX agent virtual machine from the URLs that you provide and deploys the virtual machines on the ESXi hosts. ESX Agent Manager installs one ESX agent instance per agency per host.

NOTE
To install VIBs, all ESXi hosts must have configured the firewall so that they can access the HTTP port on the vCenter Server instance.

Setting the `ovfEnvironment` property allows a solution to provide OVF properties specific to the ESX agent virtual machine. ESX Agent Manager sets the OVF properties when it deploys an ESX agent. A typical use of the `ovfEnvironment` field is to specify the IP address and credentials of the solution so that ESX agents can connect back to the solution when they are running.

1. Create a program to configure and create ESX agencies and agents.
   Your solution can define the configuration and creation of ESX agencies and agents in the `MyAgentHandler.java` class.

```
public MyAgentHandler(String selfUrl,
                      String selfIp,
                      String ovfUrl4x,
                      String ovfUrl50,
                      String vibUrl4x,
                      String vibUrl50,
                      boolean deployVibs,
                      Map<String, String> ovfEnvironment,
                      VcUtils vcUtils) {
   _vcUtils = vcUtils;
```

```
   _agentConfigInfo4x = new AgentConfigInfo();
   _agentConfigInfo50 = new AgentConfigInfo();

   [...]
}
```

2. Create `AgentConfigInfo` instances for each type of ESX agent that the ESX agency deploys.
   Your solution can define ESX agents for ESXi 6.7 and for ESXi 7.0.

```
public AgentHandler([...]) {
   [...]
   _agentConfigInfo67 = new AgentConfigInfo();
   _agentConfigInfo70 = new AgentConfigInfo();
   [...]
}
```

3. Set the URLs to the OVF files from which the solution deploys ESX agent virtual machines by calling the `AgentConfigInfo.setOvfPackageUrl()` method.
   Your solution can construct the URLs to OVF files from information that you set in the `mysolution.properties` file.

```
public MyAgentHandler([...]) {
   [...]

   _agentConfigInfo67.setOvfPackageUrl(urlPrefix + ovfUrl4x);
   _agentConfigInfo70.setOvfPackageUrl(urlPrefix + ovfUrl50);

   [...]
   }
```

4. Set the URLs to the optional VIB files from which the solution adds functions to ESXi by calling the `AgentConfigInfo.setVibUrl()` method.
   Your solution can construct the URLs to VIB files from information that you set in the `mysolution.properties` file.

```
public MyAgentHandler([...]) {
   [...]

   _agentConfigInfo4x.setOvfPackageUrl(urlPrefix + ovfUrl4x);
   _agentConfigInfo50.setOvfPackageUrl(urlPrefix + ovfUrl50);

   if (deployVibs) {
```

```
      _agentConfigInfo67.setVibUrl(urlPrefix + vibUrl4x);
      _agentConfigInfo70.setVibUrl(urlPrefix + vibUrl50);

      [...]
   }
}
```

5. Set any OVF environment properties that the solution requires by creating an instance of the `AgentOvfEnvironmentInfo` object and passing it to the `AgentConfigInfo` object for each ESX agent.
   Your solution can set some dummy properties in the `eamri-webapp.xml` file.

```
public MyAgentHandler([...]) {
   [...]
   AgentOvfEnvironmentInfo ovfEnv = new AgentOvfEnvironmentInfo();
   for (final Map.Entry<String, String> entry : ovfEnvironment.entrySet()) {
      ovfEnv.getOvfProperty().add(new AgentOvfEnvironmentInfoOvfProperty() {
         {
            setKey(entry.getKey());
            setValue(entry.getValue());
         }
      });
   }
   _agentConfigInfo67.setOvfEnvironment(ovfEnv);
   _agentConfigInfo70.setOvfEnvironment(ovfEnv);
   [...]
}
```

6. Create an instance of `AgencyConfigInfo` to define the ESX agency that the solution deploys.

```
public MyAgentHandler([...]) {
   [...]

   _agencyConfigInfo = new AgencyConfigInfo();
   [...]
}
```

7. Provide names for the ESX agency and the ESX agents by calling the `AgencyConfigInfo.setAgencyName()` and `setAgentName()` methods.
   Your solution can name the ESX agency and the ESX agents My Service.

```
public MyAgentHandler([...]) {

   [...]


   _agencyConfigInfo = new AgencyConfigInfo();

   _agencyConfigInfo.setAgencyName("My Service");

   _agencyConfigInfo.setAgentName("My Service");

   [...]

}
```

8. Add an array of ESX agent configurations to the ESX agency configuration by calling the `AgencyConfigInfo.getAgentConfig()` method.

```
public MyAgentHandler([...]) {

   [...]


   _agencyConfigInfo = new AgencyConfigInfo();

   _agencyConfigInfo.setAgencyName("My Service");

   _agencyConfigInfo.setAgentName("My Service");

   _agencyConfigInfo.getAgentConfig().add(_agentConfigInfo4x);

   _agencyConfigInfo.getAgentConfig().add(_agentConfigInfo50);

   [...]

}
```

9. Set the scope of the ESX Agency by calling the `AgencyConfigInfo.setScope()` method.
   Users of your solution can set the scope of the ESX agency by selecting ESXi hosts from your solution Configuration page. Consequently, the scope is empty until your solution updates it according to the user interaction.

```
public MyAgentHandler([...]) {

   [...]

   _agencyConfigInfo.setScope(null);

   [...]

}
```

You set the configuration properties for an ESX agency and the ESX agents that it contains.

Call the `createAgency()` method to create the ESX agency.

Create an ESX agency

You create an ESX agency by calling the `EsxAgentManager.createAgency()` method. You must specify ESX agent configurations for each version of ESXi on which you deploy ESX agents.

.

When you call the `EsxAgentManager.createAgency()` method you must pass it an `AgencyConfigInfo` object and a string to define the initial goal state of the ESX agents that the agency deploys. The `initialGoalState` property informs ESX Agent Manager of the state in which to deploy ESX agent virtual machines when the solution first runs. You can define a function that creates an ESX agency in a `MyAgentHandler.java` class.

1. Establish a connection to the ESX Agent Manager running in vCenter Server.
   The `MyAgentHandler.java` can implement the `MyEamConnection.java` class to connect to vCenter Server and ESX Agent Manager.
   ```
   public void setup(MyEamConnection myeamConnection) {

    assert myeamConnection != null;

    _myeamConnection = myeamConnection;


     ManagedObjectReference eamRef = _myeamConnection.getEsxAgentManager();

     EamPortType stub = _myeamConnection.getStub();


     [...]

   }
   ```
2. Create an ESX agency by calling the `EsxAgentManager.createAgency()` method.
   The `MyAgentHandler.java` class checks whether any ESX agencies are already running, and if not calls the `EsxAgentManager.createAgency()` method. `MyAgentHandler.java` passes to the `EsxAgentManager.createAgency()` method the `ManagedObjectReference` object for the ESX Agent Manager instance running in vCenter Server, `eamRef`. `MyAgentHandler.java` also passes to `createAgency()` the `AgencyConfigInfo` object that defines the configuration of the ESX agency. The `MyAgentHandler.java` class sets the initial goal state of the ESX agency to `ENABLED`.
   ```
   public void setup(MyEamConnection myeamConnection) {

    assert myeamConnection != null;

    _myeamConnection = myeamConnection;


     ManagedObjectReference eamRef = _myeamConnection.getEsxAgentManager();

     EamPortType stub = _myeamConnection.getStub();

     try {

       List<ManagedObjectReference> agencyRefs = stub.queryAgency(eamRef);

       if (agencyRefs != null && agencyRefs.size() > 0) {

            _agency = agencyRefs.get(0);

       } else {

         _agency = stub.createAgency(eamRef,

                                     _agencyConfigInfo,

                                     EamObjectRuntimeInfoGoalState.ENABLED.toString()
   ```

```
                                        .toLowerCase());
        }
    [...]
    }
```

3. Call the `Agency.queryConfig()` method to verify the configuration of the ESX agency and report any issues with the configuration.

```
public void setup(MyEamConnection myeamConnection) {
    [...]
    _agencyConfigInfo = stub.queryConfig(_agency);
    _isSetup = true;
  } catch (RuntimeFaultFaultMsg e) {
     _log.error(e, e);
  } catch (InvalidAgencyScopeFaultMsg e) {
     _log.error(e, e);
  } catch (InvalidAgentConfigurationFaultMsg e) {
     _log.error(e, e);
  } catch (InvalidUrlFaultMsg e) {
     _log.error(e, e);
  }
}
```

4. Call the `Agency.agencyQueryRuntime()` method to return the status of the ESX agency.

   The `Agency.agencyQueryRuntime()` method returns an `EamObjectRuntimeInfo` object that contains the goal state of the agency, its current status, and a list of any problems that the agency has encountered.

```
public EamObjectRuntimeInfo getRuntime() throws RuntimeFaultFaultMsg {
    waitForSetup();
    return _eamConnection.getStub().agencyQueryRuntime(_agency);
}
```

You created an ESX agency that a solution can deploy on ESXi hosts.

Update the Agency scope of a solution

You define the ESX agency scope of a solution by passing the managed object references (MoRefs) of the vSphere compute resources to the solution.

- .
- Verify that you have set up and started your solution in an application server.

You set the initial ESX agency scope in the `scope` property of the `AgencyConfigInfo` object. You can change the scope when a solution runs by calling the `Agency.update()` method. For example, in your solution, a user can select the ESXi hosts on which to run the solution from a list on the solution Configuration page. The solution can update the scope of the sample ESX agency according to the hosts that the user selects.
Your solution can define a function to update the scope of the ESX agency in the `MyAgentHandler.java` class.

1. Write a function that implements the vSphere Web Services API to detect compute resources on which to run the solution.
   Your solution can provide a helper class, `MyVcUtils.java`, that defines functions to obtain the compute resources on which to run the solution. `MyAgentHandler.java` calls the `MyVcUtils.getComputeResources()` method to obtain a list of `ManagedObjectReference` objects for the ESXi hosts running in vCenter Server.

```
public void updateConfig(String[] updates) throws RuntimeFaultFaultMsg {

   waitForSetup();


   boolean changed = false;

   Map<String, ManagedObjectReference> crs = _myvcUtils.getComputeResources();
```

2. Add the `ManagedObjectReference` objects for the compute resources to a `HashSet` that defines the ESX agency scope.
   The `MyAgentHandler.java` class adds the list of `ManagedObjectReference` objects that the `MyVcUtils.getComputeResources()` method returns to the existing scope and updates the list if additional compute resources are present.

```
   Set<ManagedObjectReference> newScope = new HashSet<ManagedObjectReference>();


   for (String update : updates) {
       String[] kv = update.split("=", 2);
       if (kv[0].equals("scope")) {
         try {
            ManagedObjectReference cr = crs.get(kv[1]);
            if (cr == null) {
              continue;
            }
            ManagedObjectReference moRef = new ManagedObjectReference();
            moRef.setType(cr.getType());
            moRef.setValue(cr.getValue());
            newScope.add(moRef);
         }
          catch (NullPointerException e) {
              // ignore
```

```
            }
          }
      }
```

3. Create an `AgencyComputeResourceScope` instance to contain the scope `HashSet`.

```
   AgencyComputeResourceScope scopeDO = (AgencyComputeResourceScope)
_agencyConfigInfo.getScope();

   Set<ManagedObjectReference> oldScope = new
HashSet<ManagedObjectReference>(scopeDO.getComputeResource());
```

4. Compare the old scope to the new scope to establish whether any compute resources have been added or removed.
   The `MyAgentHandler.java` class compares the size of the new scope to the initial scope and adds any new compute resources to the `HashSet` of `ManagedObjectReference` objects.

```
   if (!oldScope.containsAll(newScope) || oldScope.size() != newScope.size()) {

      AgencyComputeResourceScope scope = new AgencyComputeResourceScope();

      scope.getComputeResource().addAll(newScope);

      agencyConfigInfo.setScope(scope);

      changed = true;

   }
```

5. If the new scope differs from the old scope, call `Agency.update()` to add the new scope to the ESX agency.

```
   if (changed) {

     assert _agency != null;

     try {

        _eamConnection.getStub().update(_agency, agencyConfigInfo);

     } catch (Exception e) {

      _log.error("Failed to update agency. Reason: " + e.getMessage());

     }

     updateConfiguration();

   }

}
```

You defined a function in a solution to detect changes of scope and to update an ESX agency.

Change the goal state of an agency

A solution can change the goal state of its ESX agencies while the solution is running. For example, when a solution starts, the goal state of its ESX agencies can be `ENABLED`. If the solution includes a function to remove ESX agencies, when this function runs, the goal state of the ESX agency changes to `UNDEPLOYED`.

- .

- Verify that you have set up and started your solution in an application server.

You call the `Agency.enable()`, `Agency.disable()`, and `Agency.uninstall()` methods to change the goal state of an ESX agency. Calling these methods changes the status of the ESX agency to yellow until the agency reaches the desired state, in which case the status changes to green. If the ESX agency cannot achieve the goal state, the status changes to red.
Your solution can define a function to change the goal state of its ESX agencies in the `MyAgentHandler.java` class. Your solution can change the goal state of its ESX agency when users select ESXi hosts on which to run the solution, and when they uninstall the solution.

1. Get the current goal state of the ESX agency by calling the `EamObjectRuntimeInfo.getGoalState()` method.
   Your solution can define a function in the `MyAgentHandler.java` class to obtain the current goal state from the solution.

```
public void updateGoalState(String params) throws RuntimeFaultFaultMsg,

 NotFoundFaultMsg {

 String[] kv = params.split("=", 2);

 assert kv[0].equals("goalState");


 String goalState = kv[1];

 String currentGoalState = getRuntime().getGoalState().toString();
```

2. Call the `Agency.enable()`, `Agency.disable()`, and `Agency.uninstall()` methods to set the ESX agency in the new goal state.
   Your solution can call the appropriate methods to set the ESX agencies to the goal state. If the goal state is `UNINSTALLED`, the solution calls the`cleanup()` method that `MyManager.java` defines to remove the ESX agencies and uninstall the solution.

```
 if (goalState.equals(currentGoalState)) {

     return;

 }


 if (goalState.equals(EamObjectRuntimeInfoGoalState.ENABLED.toString()

                                                          .toLowerCase())) {

     enable();

 } else if (goalState.equals(EamObjectRuntimeInfoGoalState.DISABLED.toString()

                                                                  .toLowerCase())) {

     disable();

 } else {

     assert goalState.equals(EamObjectRuntimeInfoGoalState.UNINSTALLED.toString()


 .toLowerCase());

     _unregistered = true;
```

```
        MyManager.getInstance().cleanup();
    }
}
```

You changed the goal state of an ESX agency while the solution is running.

Delete an ESX agency

A solution can delete an ESX agency by calling the `Agency.destroyAgency()` method on the `Agency` object.

.

Typically, before deleting an ESX agency, a solution firsts call the `EsxAgentManager.uninstall()` method to put the agency in the uninstalled state. The solution tracks the progress of `EsxAgentManager.uninstall()` and only calls `destroyAgency()` to remove the ESX agency when the status of the ESX agency is green.
If your solution does not need to track the removal of the ESX agency and its ESX agents, you can call `destroyAgency()` without first calling `uninstall()`. ESX Agent Manager removes the ESX agency and all of the ESX agents without tracking the status of the uninstallation process.
Alternatively, disconnecting the solution from vCenter Server by calling the `ExtensionManager.unregisterExtension()` method removes all ESX agencies and ESX agents.

1. Call the `Agency.disable()` method to disable the ESX agency.

   Calling the `Agency.disable()` method powers off the ESX agent virtual machines, but does not undeploy them.

   Your solution can call the `Agency.disable()` method in the `MyAgentHandler.java` class.

```
public void disable() throws RuntimeFaultFaultMsg {
   waitForSetup();
      _myeamConnection.getStub().disable(_myagency);
}
```

2. Call the `Agency.uninstall()` method to put the ESX agency in the uninstalled state.

   Calling the `Agency.uninstall()` method uninstalls all the ESX agents in the ESX agency.

   Your solution can call the `Agency.uninstall()` method in the `MyAgentHandler.java` class.

```
public void uninstall() throws RuntimeFaultFaultMsg {
   waitForSetup();
      _myeamConnection.getStub().uninstall(_myagency);
}
```

3. Delete the ESX agency and ESX agents.

Option	Description
Call the `Agency.destroyAgency()` method on the `Agency` object.	Deletes the agency and its ESX agents, but the solution remains registered with `ExtensionManager`.

Table continued on next page

Continued from previous page

Option	Description
Call the `ExtensionManager.unregisterExtension()` method on the `Extension` object.	Unregisters the solution from vCenter Server, which uninstalls the solution and deletes the ESX agency and its ESX agents.

Your solution can delete the ESX agency by calling the `ExtensionManager.unregisterExtension()` method in the `MyManager.java` class.

```
public void cleanup() throws NotFoundFaultMsg, RuntimeFaultFaultMsg {

   if (_myeamConnection != null) {

      _myeamConnection.disconnect();

   }



 _myvimConnection.getStub().unregisterExtension(_myvimConnection.getExtensionManager(
);
                                                    EXTENSION_KEY);

}
```

You defined a function to delete an ESX agency from a solution.

Resolve ESX agent issues

ESX Agent Manager can detect issues in the ESX agents that solutions deploy. Solutions can try to resolve issues when the status of an ESX agency or an ESX agent is set to red.

- .
- Verify that you have set up and started your solution in an application server.

You obtain the issues that affect an ESX agency or agent by using the `EventManager` to listen for issues.
A solution can try to resolve issues by calling the `EamObject.resolve(Issue[])` method on an individual ESX agent, in which case ESX Agent Manager tries to resolve the issues. The solution can also call `EamObject.resolveAll()` on an ESX agency, in which case ESX Agent Manager attempts to resolve all the issues on all the ESX agents that the ESX agency deploys.

1. Obtain the unique identifiers of issues by calling the `Issue.getKey()` method.
   Your solution can define a method that calls the `Issue.getKey()` method in the `MyAgentHandler.java` class.
   ```
   public String getIssueId(Issue issue) {

      return Integer.toString(issue.getKey());

   }
   ```
2. Call the `Agent.getRuntime()` method to obtain an `EamObjectRuntimeInfo` object for a running ESX agent.
   Your solution can implement `Agency.getRuntime()` in a method that checks for a running ESX agency and obtains the runtime information for that ESX agency.
   ```
   public Issue getIssue(String issueId) throws RuntimeFaultFaultMsg {

      waitForSetup();
   ```

```
   EamObjectRuntimeInfo runtime = getRuntime();

   assert runtime != null;

   [...]

}
```

3. Call the `EamObject.queryIssue()` method to obtain the list of issues affecting an ESX agency from the runtime of that ESX agency.
   Your solution can add any issues that it discovers for the ESX agency to a `List` object, and returns the issues with their issue identifiers.

```
public Issue getIssue(String issueId) throws RuntimeFaultFaultMsg {

   waitForSetup();

   EamObjectRuntimeInfo runtime = getRuntime();

   assert runtime != null;

   List<Issue> issues = _myeamConnection.getStub().queryIssue(_myagency, null);

   if (issues == null) {

      return null;

   }

   for (Issue issue : issues) {

      if (getIssueId(issue).equals(issueId)) {

         return issue;

      }

   }

   return null;

}
```

4. Call the `EamObject.resolveAll()` method to resolve all issues with ESX agents running in an ESX agency.
   Your solution can call the `resolveAll()` method on the `Agency` object that the `MyAgentHandler.java` class creates.

```
public void resolveAll() throws RuntimeFaultFaultMsg {

   waitForSetup();

   _myeamConnection.getStub().resolveAll(_myagency);

}
```

5. Call the `EamObject.resolve()` method to resolve a specific issue with an ESX agent.
   Your solution can call the `resolve()` method on the issue identifier that the `getIssue()` method returns, and generates a list of unknown issues that the solution cannot resolve.

```
public void resolve(String issueId) throws NumberFormatException,

   RuntimeFaultFaultMsg {

   waitForSetup();
```

```
    List<Integer> unknownIssueIds = _myeamConnection.getStub()
                                                    .resolve(_myagency,

  Collections.singletonList(Integer.parseInt(issueId)));
    if (unknownIssueIds != null) {
       _log.error("Failed to resolve issue:" + issueId);
    }
}
```

You called the methods from the ESX Agent Manager API to obtain issues and attempt to resolve them.

How do I integrate an extension with the vCenter Extension vService

The vCenter Extension vService simplifies the installation and deployment of extensions. By integrating an extension with the vCenter Extension vService you can deploy extensions from the vSphere Client without having to enter the connection parameters of the vCenter Server on which you install the extension, or provide the login credentials for that vCenter Server instance.

The vCenter Extension vService performs the following functions for solutions that you integrate with it.

- Provides the extension with the connection parameters of the vCenter Server instance.
- Registers the extension certificate with vCenter Server.

For example, to deploy your solution you can provide the IP address and login credentials in the `mysolution.properties` file, which requires you to update the solution for every vCenter Server instance on which you install it. By integrating an extension with the vCenter Extension vService, you avoid this manual step.

Integrate a virtual machine with the vCenter Extension vService

To integrate a virtual machine with the vCenter Extension vService, you must deliver the extension in an Open Virtualization Format (OVF) package, add sections to the OVF descriptor file, and provide in the guest operating system a script that connects to the vCenter Extension vService.

Configure the OVF descriptor file

To integrate an extension with the vCenter Extension vService, you must add a section to the Open Virtualization Format (OVF) descriptor file that contains the virtual machine.

You have developed an extension running in a virtual machine that you deliver using OVF.

Including a `<vServiceDependencySection>` element in the OVF descriptor file of the virtual machine informs vCenter Server that this virtual machine depends on the vCenter Extension vService.
When you deploy a virtual machine by using OVF, vCenter Server generates an OVF environment XML document for the virtual machine. The OVF environment is a secure one-way communication channel between vCenter Server and the guest OS of the virtual machine. To use this feature you must enable OVF environment transport in the OVF descriptor of your virtual machine. The vCenter Extension vService makes the OVF environment available to the virtual machine in an ISO image that it locates in the first CD-ROM drive of the virtual machine.
The vCenter Extension vService uses the OVF environment to pass connection parameters to the guest operating system in the virtual machine. The vCenter Extension vService adds a `<vServiceEnvironmentSection>` element to the OVF environment of the virtual machine. The `<vServiceEnvironmentSection>` element contains the following connection parameters that a script running within the guest operating system requires to register with vCenter Server as an extension.

- Communication parameters that allow the virtual machine to make a secure connection to the vCenter Extension vService Guest API:
  - HTTPS URL to the vCenter Extension vService Guest API.
  - Authentication token that authenticates the virtual machine with the vCenter Extension vService.
  - SSL thumbprint of an X509 certificate that the vCenter Extension vService uses to establish a secure HTTPS connection with the virtual machine.
- Communication parameters that allow the guest operating system to make a secure connection to vCenter Server:
  - IP address of the vCenter Server instance
  - SSL thumbprint of the X509 certificate that vCenter Server uses to make a secure HTTPS connection to the virtual machine
  - HTTP port on which the vCenter Server is reachable
  - HTTPS port on which the vCenter Server is reachable
  - Managed object reference of the virtual machine in vCenter Server

  IMPORTANT
  The `<vServiceEnvironmentSection>` element contains sensitive data. Take special care within the guest operating system to prevent other users from accessing it. When the connection to vCenter Server is established, the guest operating system no longer needs the OVF environment so you should eject the CD-ROM. Ejecting the CD-ROM deletes the OVF environment from the datastore.

1. Add a `<vServiceDependencySection>` element to the OVF descriptor file for the virtual machine.
   You nest the `<vServiceDependencySection>` element in the `<VirtualSystem>` element.

```
<Envelope>
 [...]
 <VirtualSystem>
  [...]
  <vmw:vServiceDependencySection>
  </vmw:vServiceDependencySection>
  [...]
 </VirtualSystem>
 [...]
</Envelope>
```

2. Add the URLs to the standard OVF and VMware OVF schemas.

```
<vmw:vServiceDependencySection xmlns:ovf="http://schemas.dmtf.org/ovf/envelope/1"
                                        xmlns:vmw="http://www.vmware.com/schema/ovf">
</vmw:vServiceDependencySection>
```

3. Set the `ovf:required` attribute.

Option	Description
`ovf:required="true"`	Users can only deploy the OVF on vCenter Server 5.x. Users cannot power on the virtual machine if it is not bound to the vCenter Extension vService.
`ovf:required="false"`	Users can deploy the OVF vCenter Server 4.x and 5.x, but the virtual machine only integrates with the vCenter Extension vService in vCenter Server 5.x.

```
<vmw:vServiceDependencySection xmlns:ovf="http://schemas.dmtf.org/ovf/envelope/1"
                               xmlns:vmw="http://www.vmware.com/schema/ovf"
                               ovf:required="true"
                               vmw:id="installation" >
</vmw:vServiceDependencySection>
```

4. Set the type of the dependency to `com.vmware.vservice.extension`, to bind the extension to the vCenter Extension vService.

```
<vmw:vServiceDependencySection xmlns:ovf="http://schemas.dmtf.org/ovf/envelope/1"
                               xmlns:vmw="http://www.vmware.com/schema/ovf"
                               ovf:required="true"
                               vmw:id="installation" >
   <Info>A vService dependency</ovf:Info>
   <vmw:Type>com.vmware.vservice.extension</vmw:Type>
</vmw:vServiceDependencySection>
```

5. Provide a name and a description for the vService dependency, that appears in vService Manager.

```
<vmw:vServiceDependencySection xmlns:ovf="http://schemas.dmtf.org/ovf/envelope/1"
                               xmlns:vmw="http://www.vmware.com/schema/ovf"
                               ovf:required="true"
                               vmw:id="installation" >
   <ovf:Info>A vService dependency</ovf:Info>
   <vmw:Type>com.vmware.vservice.extension</vmw:Type>
   <vmw:Name>dependency_name</vmw:Name>
   <vmw:Description>dependency_description</vmw:Description>
   <vmw:Configuration />
</vmw:vServiceDependencySection>
```

6. Enable OVF environment transport by setting the `ovf:transport` attribute of the `<VirtualHardwareSection>` element to `iso`.

```
<Envelope>
 [...]
```

```
    <VirtualSystem>
     [...]
     <VirtualHardwareSection ovf:transport="iso">
       [...]
     </VirtualHardwareSection>
     [...]
    </VirtualSystem>
    [...]
   </Envelope>
```

You configured the OVF file for a extension to bind the extension to the vCenter Extension vService and to make the connection information for the vCenter Server instance available over OVF transport.

vService dependency section in the OVF descriptor file

The following code extract shows an example of a `<vServiceDependencySection>` element in an OVF descriptor file.

```
<vmw:vServiceDependencySection xmlns:ovf="http://schemas.dmtf.org/ovf/envelope/1"
                                        xmlns:vmw="http://www.vmware.com/schema/ovf"
                                        ovf:required="true"
                                        vmw:id="installation" >
   <Info>A vService dependency</Info>
   <vmw:Type>com.vmware.vservice.extension</vmw:Type>
   <vmw:Name>vCenter Extension Installation</vmw:Name>
   <vmw:Description>
        This appliance requires a binding to the vCenter Extension vService,
        which allows it to register automatically as a vCenter Extension at runtime.
   </vmw:Description>
   <vmw:Configuration />
</vmw:vServiceDependencySection>
```

Provide a script in the Extension virtual machine to register as a vCenter Extension

You must write a script that you run in the virtual machine that contains the extension.

- You have developed an extension running in a virtual machine that you deliver using OVF.
- You have configured the OVF descriptor file of the extension to depend on the vCenter Extension vService and enabled OVF transport.

The script reads the URL of the vCenter Extension vService Guest API, SSL thumbprint, and authentication token in the `<vServiceEnvironmentSection>` element of the Open Virtualization Format (OVF) environment XML file. The script authenticates itself with the vCenter Extension vService using the authentication token from the vService environment, generates a self-signed certificate, and sends the extension key and certificate to the `RegisterExtension` function from the vCenter Extension vService Guest API.
The script that you run in the guest operating system of the virtual machine must perform the following functions.

1. Parse the `<vServiceEnvironmentSection>` section in the OVF environment file.
2. Create a self-signed X509 certificate.

   For example, create the certificate by using OpenSSL.

3. Send an **`HTTP POST`** request to the vCenter Extension vService.

   The script obtains the URL to the vCenter Extension vService from the `<vServiceEnvironmentSection>`.

4. Include in the `HTTP` header the authentication token value from the `<vServiceEnvironmentSection>` element of the OVF environment file.
   `"evs-token: `*`Token value`*`"`
5. Verify that the SSL thumbprint of the server-side certificate of the HTTPS connection is the same as that of the `X509Thumbprint` value from the `<vServiceEnvironmentSection>` element of the OVF environment file.
6. In the body of the **`HTTP POST`** request, pass the extension key of the extension and the PEM encoding of the certificate to the `RegisterExtension` function from the vCenter Extension vService Guest API.

```
<RegisterExtension xmlns="http://www.vmware.com/schema/vservice/ExtensionVService">
   <Key>com.mycompany.extensionkey</Key>
   <Certificate>
   -----BEGIN CERTIFICATE-----
   MIICRTCCAa4CCQDC/hX5KA9rSzANBgkqhkiG9w0BAQUFADBnMQswCQYDVQQGEwIu
   LjEKMAgGA1UECBMBLjEKMAgGA1UEBxMBLjEPMA0GA1UEChMGVk13YXJlMQowCAYD
   VQQLEwEuMREwDwYDVQQDEwhFVlMgZGVtbzEQMA4GCSqGSIb3DQEJARYBLjAeFw0x
   MDA3MDYwNzUwNTVaFw0xMDA4MDUwNzUwNTVaMGcxCzAJBgNVBAYTAi4uMQowCAYD
   VQQIEwEuMQowCAYDVQQHEwEuMQ8wDQYDVQQKEwZWTXdhcmUxCjAIBgNVBAsTAS4x
   ETAPBgNVBAMTCEVWUyBkZW1vMRAwDgYJKoZIhvcNAQkBFgEuMIGfMA0GCSqGSIb3
   DQEBAQUAA4GNADCBiQKBgQC4/XVcMhvNixk35iWl3nn1KHVSgUE18TuQBj7spNUc
   y506RmV8BR847jg9fHl7aErShOQ8RT/EuEEUGey4U1dB2pSocoYldtp2r4g/Lcew
   ZuuyQh2+MC0YzeFe+nyxBDHa0BGUId0dQH9nrjyboW/kNIrWfDkXnxxtq6pQAmFw
   +QIDAQABMA0GCSqGSIb3DQEBBQUAA4GBAElC68z59fuicYUa4fGWBuXNxzb+uqWF
   +cnf78lctBY1pr1DcEedhyww2SYbaGh/xGCc1zqO5kqYhIexQbN/2Vxaol9lJc/n
   vRfQRCp+HaIFTJMu4mVZ2GsYSp/tZSGgiBBQAUXqCLxFQr0eQ29b9rj4Q3/1N+7i
   hbVOln67TOBZ
   -----END CERTIFICATE-----
   </Certificate>
```

```
   </RegisterExtension>
```

The vCenter Extension vService calls the `RegisterExtension` function and registers the extension with vCenter Server.

`vServiceEnvironmentSection` in the OVF environment XML file

If you add a vService dependency in an OVF descriptor from which you deploy an extension, the vCenter Extension vService generates and adds a `<vServiceEnvironmentSection>` element to the OVF environment of the virtual machine that it deploys.

```
<ve:vServiceEnvironmentSection xmlns:ve="http://www.vmware.com/schema/ovfenv"
                                          xmlns:evs="http://www.vmware.com/schema/vservice/
ExtensionVService"
                                          ve:bound="true" ve:id="installation"
                                          ve:type="com.vmware.vservice.extension">
   <evs:GuestApi>
         <evs:URL>https://192.168.1.42/vsm/extensionService</evs:URL>
         <evs:Token>1efc34a14232f81a245b9e8172f7a383fdeab312</evs:Token>
         <evs:X509Thumbprint>2b:04:e6:7d:8c:7b:73:70:d4:29:32:ed:96:11:2b:ae:b4:a0:28:78</
evs:X509Thumbprint>
   </evs:GuestApi>
   <evs:VCenterApi>
         <evs:IP>192.168.1.42</evs:IP>
         <evs:X509Thumbprint>2b:04:e6:7d:8c:7b:73:70:d4:29:32:ed:96:11:2b:ae:b4:a0:28:78</
evs:X509Thumbprint>
         <evs:HttpPort>80</evs:HttpPort>
         <evs:HttpsPort>443</evs:HttpsPort>
         <evs:SelfMoRef>VirtualMachine:vm-246</evs:SelfMoRef>
   </evs:VCenterApi>
</ve:vServiceEnvironmentSection>
```

Deploy an extension in the vSphere Client by using the vCenter Extension vService

If you configure the virtual machine that runs an extension to use the vCenter Extension vService, you can deploy the extension directly in the vSphere Client.

- You have developed an extension running in a virtual machine that you deliver using OVF.
- You have configured the OVF descriptor file of the extension to depend on the vCenter Extension vService and enabled OVF transport.

- You have provided a script in the virtual machine that runs the extension that reads the OVF environment file to obtain the connection parameters from the vCenter Extension vService and uses them to register as an extension with vCenter Server.

1. Log in to the vSphere Client.
2. Select **Menu › Hosts and Clusters**.
3. Select the host.
4. From the **Actions** pull-down menu, select **Deploy OVF Template**.
5. Type the URL to the OVF file from which to deploy the extension.
6. Select a host, datastore, disk format, and network on which to run the extension.

You deployed an extension from OVF by using the vCenter Extension vService.

vCenter Extension vService XML schema

The vCenter Extension vService XML Schema defines the `<vServiceEnvironmentSection>` in the OVF environment of the virtual machine that contains the extension. It also defines the `RegisterExtension` function that registers the extension with vCenter Server.

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:evs="http://www.vmware.com/schema/vservice/ExtensionVService"
        targetNamespace="http://www.vmware.com/schema/vservice/ExtensionVService"
        elementFormDefault="qualified" attributeFormDefault="qualified">

   <!-- RegisterExtension_Type:
        The command type used in the guest API when registering as a
        vCenter extension.
   -->
   <complexType name="RegisterExtension_Type">
      <sequence>
         <!-- The key of the extension to register -->
         <element name="Key" type="string" />
         <!-- The PEM-encoded certificate of the extension to register -->
         <element name="Certificate" type="string" />
         <any processContents="lax" minOccurs="0" maxOccurs="unbounded" />
      </sequence>
      <anyAttribute namespace="##any" processContents="lax"/>
```

```
</complexType>

<element name="RegisterExtension" type="evs:RegisterExtension_Type" />

<!-- GuestApi: Contains information for the guest software about how to connect to the
     Extension vService.
-->
<element name="GuestApi">
  <complexType>
   <sequence>
      <!-- The URL of the Extension vService guest API -->
      <element name="URL" type="anyURI" />
      <!-- Authentication token for the Extension vService guest API -->
      <element name="Token" type="string" />
      <!-- The X509 thumbprint of the Extension vService server
        certificate.
      -->
      <element name="X509Thumbprint" type="string" />
      <any processContents="lax" minOccurs="0" maxOccurs="unbounded" />
   </sequence>
   <anyAttribute namespace="##any" processContents="lax"/>
  </complexType>
</element>

<!-- VCenterApi:
     Contains information for the guest software about how to connect
     to vCenter Server.
-->

<element name="VCenterApi">
  <complexType>
   <sequence>
```

```
         <!-- The IP address of the vCenter server -->
         <element name="IP" type="string" />
         <!-- The X509 thumbprint of the vCenter server certificate -->
         <element name="X509Thumbprint" type="string" />
         <!-- The HTTP port of the vCenter server -->
         <element name="HttpPort" type="int" />
         <!-- The HTTPS port of the vCenter server -->
         <element name="HttpsPort" type="int" />
         <!--
             The managed object reference of this virtual machine
             in the vCenter server
         -->
         <element name="SelfMoRef" type="string" />
         <any processContents="lax" minOccurs="0" maxOccurs="unbounded" />
      </sequence>
      <anyAttribute namespace="##any" processContents="lax"/>
     </complexType>
   </element>
</schema>
```

VMware vSphere Automation REST API Programming Guide

VMware vSphere Automation REST API Programming Guide provides information about how to use the VMware vSphere®Automation™ REST services to automate different vSphere management tasks.

At VMware, we value inclusion. To foster this principle within our customer, partner, and internal community, we have updated this guide to remove instances of non-inclusive language.

Intended Audience

This manual is intended for anyone who wants to develop applications for accessing and using vSphere features such as virtual machine management, tagging, content libraries, managing the life cycle of clusters with the vSphere Lifecycle Manager, managing internal and external certificates, vSphere Supervisor, and so on. The information is written for developers who have understanding of the targeted vSphere features and some experience with the Representational State Transfer (REST) architectural style.

Introduction to the vSphere Automation REST API

Use the vSphere Automation REST API to manage your virtual infrastructure by using HTTP.

With the vSphere Automation REST API, you can manage vSphere components connected to authentication, tagging, content libraries, clusters, internal and external certificates, namespaces, vSphere Trust Authority, vSphere Supervisor and more.

This programming guide introduces the most important features and use cases of the vSphere Automation REST API. You can use this guide in combination with the *vSphere Automation API Reference* that features all available API services, as well as many code samples.

Understanding the vSphere Automation API

You can use the vSphere Automation REST services to automate many aspects of your vSphere environment such as configuration, authentication, certificate management, tagging, managing the life cycle of clusters, content libraries, Kubernetes cluster management, and many more.

- How Does the vSphere Automation API Work
- The API in Context
- Which REST Clients to Use with the API
- Why Use the vSphere Automation API Through REST

How Does the vSphere Automation API Work

The vSphere Automation API follows a resource-based REST architecture with JavaScript Object Notation (JSON) requests and responses. REST, an acronym for Representational State Transfer, is a software architectural style in which programs use the Hypertext Transfer Protocol (HTTP) to exchange serialized representations of objects between a client and a server. The client sends an HTTP request and receives an HTTP response from the server. Request and response bodies are encoded using JSON, and must conform to the JSON schema associated with each operation.

The vSphere Automation API uses the main CRUD (Create, Retrieve, Update, Delete) functions over HTTP to manage representations of the objects that the API defines. In the vSphere Automation API, these objects are represented by a collection of JSON schemas.

You use:
- HTTP GET to retrieve a representation of an object
- HTTP POST to create an object
- HTTP PATCH or PUT to modify an object

- HTTP DELETE to delete an object

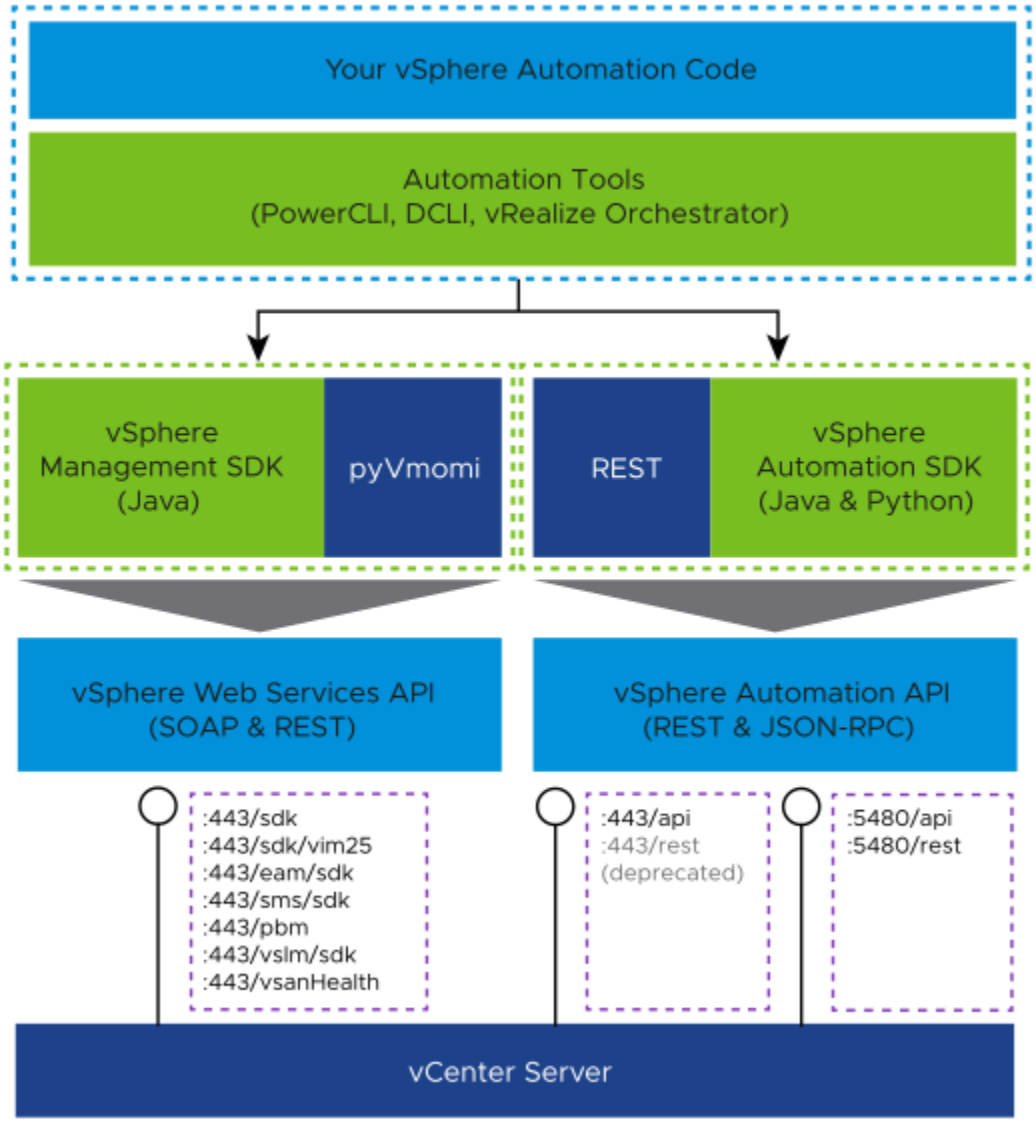
In addition, the HTTP POST operation is used for some custom actions outside of the CRUD framework such as registering or relocating virtual machines.

The HTTP verbs are also called methods. Each API operation is identified by a method and a URL. Operations can also contain a body that is composed of one or more data structures. The operation body contains parameters that are structured as key-value pairs. For more information, see API Requests.

The vSphere Automation API in Context

To take full advantage of the vSphere Automation API, you must understand the broader context in which it functions.

Figure 89: vCenter Server Automation Architecture



vSphere Automation API: the REST API for vSphere Management

The underlying platform of the vSphere Automation API is vSphere, which consists of vCenter Server and ESXi. The vSphere Automation API is available to all licensed vSphere users.

Starting with vSphere 6.0, VMware introduced the vSphere Automation API to supplement the SOAP-based vSphere Web Services API. Since then, VMware engineers have been designing new vSphere features with a RESTful approach and these new services have been integrated into the vSphere Automation API. As a result, you can now use the vSphere Automation API to manage and automate many aspects of your vSphere environment such as vCenter Server configuration, authentication, tagging, content libraries, managing the life cycle of clusters, managing internal and external certificates, vSphere Trust Authority, vSphere Supervisor, and so on.

However, you cannot manage all aspects of your vSphere environment with the vSphere Automation API as it does not cover all vSphere features. To expose the full set of vSphere functionalities you must use both the vSphere Automation API and the vSphere Web Services API. For both APIs, VMware has compiled SDKs that help programmers automate tasks.

vSphere Web Services API: the SOAP API for vSphere Management

The vSphere Web Services API, which is based on the SOAP protocol, provides basic vSphere functionalities such as host management, virtual machine provisioning, monitoring, vSAN and many more. It uses port 443 and the `/sdk` and `/sdk/vim25` base paths. The vSphere Web Services API can be consumed through:

- the vSphere Management SDK for Java
- the Python SDK for managing vSphere (pyVmomi)
- the Virtual Infrastructure JSON API

  Starting in vSphere version 8.0.1, you can communicate with the vSphere Web Services API by using HTTP and JSON. For more information, see the *vSphere Web Services SDK Programming Guide*.
- customer generated bindings from the WSDL files included in the vSphere Management SDK

vSphere Automation API Ports

The vSphere Automation API uses the standard HTTPS network port 443. A subset of the vSphere Automation API related to appliance configuration and life cycle is available also on port 5480. You can use the services on port 5480 immediately after installing vCenter Server and during life cycle and configuration operations when the services on port 443 are unavailable.

vSphere Automation API Base Paths

All existing and non-deprecated HTTP operations of the vSphere Automation API are available on port 443 and the `/api` base path. The APIs released up to vSphere version 7.0.2 are also available on the deprecated `/rest` base path. The `/api` base path will remain the only active base path when the `/rest` base path is removed in a future vSphere release.

vSphere Automation API Security

The vSphere Automation REST APIs are designed with security best practices in mind. The HTTP link between the client application and the vSphere Automation API is established through the Transport Layer Security (TLS) cryptographic protocol. User identity security is guaranteed by token-based authentication and by the OAuth 2.0 workflows for federated authentication.

SDKs for the vSphere Automation API

VMware supports open-source SDKs that empower developers to consume the vSphere Automation API through a programming language. The vSphere Automation SDK for Java and the vSphere Automation SDK for Python contain tools, libraries, documentation, and sample code for vSphere automation. Both SDKs are based on the JSON-RPC protocol. You can download or contribute to the vSphere Automation SDKs on GitHub.

vSphere Automation Tools

Automation tools such as PowerCLI, Data Center CLI (DCLI), and VMware Aria Automation Orchestrator provide another option for consuming the vSphere APIs. For example, you can use PowerCLI and the vSphere Automation SDK for PowerShell to communicate directly with the vSphere Automation API by using PowerShell scripts. For more information, see the *PowerCLI User's Guide*.

User Interface Tools

User interface tools such as the vSphere Client and the vCenter Server Management Interface consume the vSphere Automation API and the vSphere Web Services API in the back end.

Which REST Clients to Use with the vSphere Automation API

Client Applications for REST

You can use any REST client to communicate with the vSphere Automation API.

Some popular applications are:

- cURL - http://curl.haxx.se
- Postman - https://www.getpostman.com/
- jq parser - https://stedolan.github.io/jq/

API Explorer in the vSphere Client

Use the API Explorer, a built-in tool in the vSphere Client, to explore and try out vSphere APIs.

Why Use the vSphere Automation API Through REST

You can choose whether to access the vSphere Automation API directly through REST or through the various SDKs and automation tools binded to the API.

Possible use cases for using the vSphere Automation API through REST:

- There is no SDK available for your programming language. For example, you use PHP, JavaScript, Swift, Rust, and so on.
- Your integration footprint is small and you can't justify the overhead of installing and maintaining the vSphere Automation SDKs for Java or Python. For example, if you are designing a client application that fetches a few parameters from your vSphere environment, you might decide that installing and maintaining the Java or Python SDKs is unnecessary.

vSphere Automation API Base Path

You connect to the vSphere Automation REST API services through the `/api` base path.

The vSphere Automation REST API services append to the `/api` base path, following the host and port segments of the URL.

For example, you can reach the `Session` service through the following URL.

`https://<vcenter_server_ip_address_or_fqdn>/api/session`

For the full list of vSphere Automation API services, see the *vSphere Automation API Reference*.

> **CAUTION**
> Starting with vSphere 7.0 Update 2, you should use the `/api` base path to connect to the vSphere Automation API. The old `/rest` base path has been deprecated and will be removed in a future release. All existing and non-deprecated HTTP operations that use the `/rest` base path are available on the `/api` base path, which will remain the only active base path after the `/rest` base path is removed.

vSphere Automation API Requests

To retrieve or manage resources, clients make HTTP requests to a vSphere Automation API server.

An API request through HTTP must include a method (or verb), a URL, and headers. Depending on the type of operation, the request might or might not include a body.

HTTP Method

The methods of the vSphere Automation API follow the logic of the CRUD (Create, Retrieve, Update, Delete) functions used to interact with database applications.

Table 44: vSphere Automation API Methods

HTTP Method/Verb	Operation Type	Operation Summary
`POST`	Create	Creates a resource. Also used for some custom actions such as registering or relocating virtual machines.
`GET`	Retrieve	Retrieves the representation of an existing resource in its current state.
`PUT`	Update	Updates or replaces an entire resource.
`PATCH`	Update	Updates or replaces a part of a resource.
`DELETE`	Delete	Deletes a resource.

URL

You can reach each vSphere Automation API operation through a specific URL.

The URL is typically composed of:

- the vCenter Server IP address or FQDN.
- the `/api` base path
- the package name
- the service (also called interface) name
- the resource identifier (if applicable)

```
https://<vcenter_server_ip_address_or_fqdn>/api/<package_name>/<service_name>/<resource_identifier>
```

To determine what you want to do with the resource, you associate an HTTP verb (POST, GET, PUT, PATCH, or DELETE) to the URL.

For example, to retrieve information about a virtual machine, you contact the `VM` service of the `vcenter` package with the GET method. You pass the virtual machine identifier as a path parameter.

```
GET https://<vcenter_server_ip_address_or_fqdn>/api/vcenter/vm/<vm_identifier>
```

HTTP Headers

The HTTP headers transfer metadata associated with the API requests and responses. They are an integral part of HTTP requests and responses as they convey information about authorization, request and response content type, response caching, and so on.

You must typically set the following HTTP headers in your requests.

`Authorization`

All requests to create a vSphere Automation API session must include an `Authorization` header in the form prescribed by your identity provider. For vCenter Single Sign-On basic authentication, you must use the base-64

encoded value of your user name and password separated by a colon (`username:password`). For more information, see Authentication Mechanisms.

`vmware-api-session-id`

This is a custom header that you use to authenticate your subsequent calls after you obtain a session identifier. Paste the session identifier as the header value.

`Content-Type`

This header is used to indicate the data format of the request body.

Use *`application/x-www-form-urlencoded`* for requests to the `Authentication Token` service.

Use *`application/json`* for all other requests.

Request Body

The HTTP body must be in the JSON format.

Some vSphere Automation API requests contain a body and some do not. For example, retrieve (GET) operations do not contain a body in the request.

To explore the request and response schemas, visit the *`vSphere Automation API Reference`*.

vSphere Automation API Responses

The vSphere Automation API responds to every HTTP request with either a success or an error status code.

The vSphere Automation API uses a specific set of HTTP success and error codes for the responses. The response structure depends on the request type: some responses include a document body, some include a string, and some are empty.

Table 45: HTTP Success Codes

HTTP Code	Message Type	Message Description
`200`	OK	The request is valid and was completed. The response includes a document body.
`201`	`Created`	The request is valid. The requested object was created and its identifier is returned with the response.
`202`	`Accepted`	The request is valid and a task was created to handle it. The response usually contains a task identifier.
`204`	`No Content`	The request is valid and was completed. The response does not include a body.

Table 46: HTTP Error Codes

HTTP Code	Error Type	Error Description
`400`	`ALREADY_EXISTS`	An attempt was made to create an entity but an entity with the same name or identifier already exists in that context.
`400`	`ALREADY_IN_DESIRED_STATE`	Indicates that an attempt to change the state of a resource or service had no effect because the resource or service is already in the desired state.
`400`	`CANCELED`	Indicates that an operation was canceled due to an explicit request to do so.
`400`	`FEATURE_IN_USE`	Indicates that an action cannot be completed because a feature is in use. For example, trying to deactivate snapshots on a virtual machine that has a snapshot.

Table continued on next page

Continued from previous page

HTTP Code	Error Type	Error Description
400	INVALID_ARGUMENT	Indicates that the values accepted for one or more parameters are not acceptable. Examples: • A parameter value that is not of the expected type. • A parameter value that is not in the required range. • A parameter value that is different from the specifically allowed strings.
400	INVALID_ELEMENT_CONFIGURATION	Indicates that an attempt to modify the configuration of an element or a group containing the element failed due to the configuraton of the element. Examples: • An attempt to move a host with a fault-tolerant virtual machine out of a cluster. • An attempt to remove a host from a DRS cluster without putting the host into maintenance mode.
400	INVALID_ELEMENT_TYPE	Indicates that the server was unable to fulfill the request because an element of a specific type does not fit into a particular container type. Examples: • An attempt to put a virtual machine into a folder that can only contain hosts. • An attempt to attach an SCSI virtual disk to an IDE port.
400	INVALID_REQUEST	Indicates that the request is formed in such a way that the server is unable to process it. For example, sending an invalid JSON structure.
400	NOT_ALLOWED_IN_CURRENT_STATE	Indicates that the requested operation is not allowed with a resource or service in its current state. Examples: • Trying to upgrade the virtual hardware version of a suspended virtual machine. • Trying to power off, reset, or suspend a virtual machine that is not powered on.
400	RESOURCE_IN_USE	Indicates that the operation could not be completed because a resource is in use. Examples: • Trying to remove a datastore when there is a virtual machine registered on a host attached to the datastore. • Trying to add a virtual switch if the physical network adapter being bridged is already in use.
400	UNEXPECTED_INPUT	Indicates that the request body contains a parameter or a field whose name is unknown to the server.
400	UNSUPPORTED	Indicates that the operation is not supported by the underlying platform. Examples: • Trying to hot-plug a CPU when the current virtual machine configuration does not support CPU hot-plugging . • Trying to change the memory size to a value beyond the acceptable guest memory limits supported by the virtual machine's host.
400	UNVERIFIED_PEER	Indicates that an attempt to connect to an unknown or untrusted endpoint failed because the system was unable to verify the identity of the endpoint. Typically, the error data field of this error contains information about the endpoint. If you decide to trust the endpoint, the request can be resubmitted with an indication that the endpoint should be trusted. Examples:

Table continued on next page

Continued from previous page

HTTP Code	Error Type	Error Description
		• The client provides an IP address or an URL of an endpoint the system should communicate with using a TLS connection, but the endpoint's TLS certificate is self-signed, expired, or otherwise untrustworthy. • The client provides an IP address of a host, with which the system must communicate through SSH, but SSH does not recognize the host's public key.
401	UNAUTHENTICATED	Indicates that the operation requires authentication and the user is not authenticated. For example, if the session identifier in the request header is missing or if it identifies a session that has expired.
403	UNAUTHORIZED	Indicates that the user is not authorized to perform the operation. For example, some operations might require that the user has one or more privileges over the operation or over a resource identifier but the user identified by the operation does not have the required privileges.
404	NOT_FOUND	Indicates that a specified resource could not be found. Examples: • Trying to retrieve information about a virtual machine by passing an ID that does not match an existing virtual machine. • Trying to remove a vSwitch by passing an ID that does not match an existing vSwitch.
404	OPERATION_NOT_FOUND	Indicates that the API infrastructure is unable to find the requested service or operation. For example, providing invalid REST API service name or method.
409	CONCURRENT_CHANGE	Indicates that a data structure, entity, or resource has been modified in comparison to some earlier point in time. For example, when the client is doing the write portion of a read-modify-write sequence and indicates that it wants the server to notify it if the data in the server has changed after the read was done, so that overwriting a change can be avoided.
500	ERROR	Indicates that the operation resulted in some error that does not fit into the standard error types.
500	INTERNAL_SERVER_ERROR	Indicates that the server encountered an unexpected condition that prevented it from fulfilling the request. Examples: • The operation returns a value type that does not match the operation schema. • The operation returns an error that is not included in the list of vSphere Automation API errors.
500	RESOURCE_BUSY	Indicates that the operation could not be completed because a resource is busy. For example, trying to power off a virtual machine that is in the process of being powered on.
500	RESOURCE_INACCESSIBLE	Indicates that the operation could not be completed because an entity is not accessible. For example, attempting to invoke some operation on a virtual machine when the virtual machine's configuration file is not accessible.
500	TIMED_OUT	Indicates that the operation did not complete within the allowed amount of time. The operation might or might not complete after the TIMED_OUT error is reported.
500	UNABLE_TO_ALLOCATE_RESOURCE	Indicates that the operation failed because it was unable to allocate or acquire a required resource. Examples: • Trying to power on a virtual machine when there are not enough licenses to do so. • Trying to power on a virtual machine that would violate a resource usage policy.

Table continued on next page

Continued from previous page

HTTP Code	Error Type	Error Description
503	SERVICE_UNAVAILABLE	Indicates that the service you are trying to reach is unavailable. Examples: • Trying to reach the server when it is too busy. • Trying to reach the server while it's undergoing maintenance.

Supplementing the vSphere Automation APIs

Some features are not accessible by API. These features require direct shell access or specific user interfaces.

vSphere Client

The vSphere Client is a user interface for general management tasks. All administrative functions are available through the vSphere Client.

ThevSphere Client is a cross-platform application that can connect only tovCenter Server. It has a full range of administrative functionality and an extensible plug-in-based architecture. Typical users are virtual infrastructure administrators, help desk, network operations center operators, and virtual machine owners.

vCenter Server Management Interface

The vCenter Server Management Interface is an interface for configuring, monitoring, and patching vCenter Server.

The vCenter Server Management Interface runs in a browser that connects to port 5480 of vCenter Server. The vCenter Server Management Interface provides access to all the service APIs of vCenter Server.

Direct Console User Interface

The Direct Console User Interface (DCUI) provides access to basic operations for vCenter Server management and set up.

The DCUI provides access to a subset of management functions. It provides direct access to vCenter Server if the vSphere Client and the vCenter Server Management Interface become unavailable.

After the vCenter Server startup is complete, the DCUI displays basic CPU, memory, and network information on the operator console. The root user can use the DCUI screen to configure network interfaces, DNS, and super administrator password.

vCenter Server Appliance Bash Shell

You can use the vCenter Server appliance Bash shell to access all vCenter Server commands and plug-ins that you use for monitoring, troubleshooting, and configuring the vCenter Server instance through the API. For more information about the appliance Bash shell, see *vCenter Server Configuration*.

Data Center CLI

The Data Center CLI (DCLI) is a CLI client of the vSphere Automation APIs. Almost all methods that are available in the vSphere Automation APIs are available as DCLI commands. For more information about DCLI, see *DCLI User's Guide*.

PowerCLI

PowerCLI contains modules of cmdlets based on Microsoft PowerShell. The main PowerCLI module provides cmdlets for automated administration of the vSphere environment.

PowerCLI, the most popular vSphere automation tool, provides alternative, and often more user-friendly, methods to consume the vSphere APIs. The vSphere Automation SDK for PowerShell that was introduced with VMware PowerCLI

12.4, enables you to prepare data structures and call the vSphere REST APIs directly from PowerShell. It uses the `Initialize-` and `Invoke-` cmdlet prefixes to prepare data structures and call API endpoints. This approach keeps the REST workflow intact but transfers it to a PowerShell interface. For more information, see the *Managing the vSphere Automation API with VMware PowerCLI* chapter in the *PowerCLI User's Guide*.

vSphere Web Services API

The vSphere API is exposed as a Web service, running on VMware vSphere server systems. The API provides access to the vSphere management components - the managed objects that you can use to manage, monitor, and control life-cycle operations of virtual machines and other virtual infrastructure components, such as data centers, datastores, networks, and so on. For more information about the vSphere Web Services API, see the *vSphere Web Services API Reference*.

Authentication Mechanisms

To perform operations in the vSphere environment, principals must authenticate to the vSphere Automation services. You can use the default vCenter Single Sign-On identity source or federate your vCenter Server to an external identity provider such as Microsoft Active Directory Federation Services, Okta, or Azure AD.

As a result from authenticating to the vSphere Automation services, client applications and users receive a session identifier. You can choose from several supported authentication mechanisms to obtain a session identifier that grants you access to the vSphere Automation services in the virtual environment.

Choosing Your Identity Source

First, you must decide whether to authenticate to vCenter Server by using the default identity provider, vCenter Single Sign-On, or leverage your enterprise configuration to an external identity provider.

Default or External Identity Provider

vCenter Server acts as the default identity provider, by using its built-in vCenter Single Sign-On service to authenticate users and service accounts.

In vSphere 7.0 and later, you can delegate vSphere identity management to an external identity provider and take advantage of increased security and multifactor authentication. In this scenario, vCenter Single Sign-On is replaced by an external identity provider as the entity that performs principals' authentication.

Identity provider federation enhances the security and compliance of your applications and provides access to flexible benefits such as multifactor authentication (MFA) and automatic account synchronization.

You can federate vCenter Server authentication to:

- AD FS (starting in vSphere 7.0)
- Okta (starting in vSphere 8.0 Update 1)
- Azure AD (starting in vSphere 8.0 Update 2)

For more information, see Identity Provider Federation for vCenter Server.

Basic or Token-Based Authentication

You can use basic and token-based authentication to authenticate principals to the vSphere Automation services. VMware encourages you to use token-based authentication as it provides more security and flexibility for your applications and users.

Basic Authentication

Basic authentication passes a user name and password to vCenter Server.

By default, principals can use the vCenter Single Sign-On user name and password to connect to the vSphere Automation endpoint. The vSphere Automation endpoint checks whether the user name and password are present in the VMware Directory Service. On success, the system returns a session identifier for the vSphere Automation endpoint.

If your vCenter Server is federated to an external identity provider, you can use basic authentication through the OAuth 2.0 Password grant type.

> **IMPORTANT**
> VMware encourages you to refrain from using basic authentication and adopt token-based authentication, as it is more secure and provides more flexibility.

Token-Based Authentication

Token-based authentication involves the use of an encrypted token that provides authentication and authorization data to the vSphere Automation endpoint. The vCenter Server token complies with the Security Assertion Markup Language (SAML) specification, an XML-based schema for communicating authentication data.

To acquire a SAML token, client applications issue a token request to vCenter Single Sign-On. Client applications then send the SAML token to the vSphere Automation API endpoint in exchange for a session identifier.

Starting with vSphere 7.0, you can use token-based authentication for your federated vCenter Server through a 3-step process:

1. Principals use the various OAuth 2.0 grant types with the external identity provider to obtain identity and access tokens in JSON Web Token (JWT) format.
2. Principals exchange the JWT tokens for a vCenter Single Sign-On SAML token.
3. Principals use the SAML token to obtain a session identifier from the vSphere Automation endpoint.

Authentication Terminolgy

To use the vSphere programming features effectively, you must understand a set of specific authentication terms and concepts.

Table 47: vCenter Single Sign-On Glossary

Term	Definition
Principal	An entity that can be authenticated, such as a user.
Identity Provider	A service that manages identity sources and authenticates principals. Examples: Microsoft Active Directory Federation Services (AD FS) and vCenter Single Sign-On.
Identity Source (Directory Service)	Stores and manages principals. Principals consist of a collection of attributes about a user or a service account such as name, address, email, and group membership. Examples: Microsoft Active Directory and VMware Directory Service (vmdir).
Authentication	The means of determining whether someone or something is, in fact, who or what it declares itself to be. For example, users are authenticated when they provide their credentials, such as smart cards, user name and correct password, and so on.
Authorization	The process of verifying what objects principals have access to.

Table continued on next page

Continued from previous page

Term	Definition
Token	A signed collection of data comprising the identity information for a given principal. A token might include not only basic information about the principal such as email address and full name, but also, depending on the token type, the principal's groups and roles.
vmdir	VMware Directory Service. The internal (local) LDAP repository in vCenter Server that contains user identities, groups, and configuration data.
OAuth 2.0	An open authorization standard that enables the exchange of information among principals and web services without exposing principals' credentials.
OpenID Connect (OIDC)	Authentication protocol based on OAuth 2.0 that augments OAuth with user-identifying information. It is represented by the ID token that the authorization server returns together with the access token during OAuth authentication. vCenter Server uses OIDC capabilities when interacting with Active Directory Federation Services (AD FS), Okta, Microsoft Entra ID, and PingFederate.
System for Cross-domain Identity Management (SCIM)	The standard for automating the exchange of user identity information between identity domains or IT systems.
VMware Identity Services	Starting in version 8.0 Update 1, VMware Identity Services is a built-in container within vCenter Server that you can use for identity federation to external identity providers. It serves as an independent identity broker within vCenter Server and comes with its own set of APIs. Currently, VMware Identity Services support Okta, Microsoft Entra ID, and PingFederate as external identity providers.
Tenant	A VMware Identity Services concept. A tenant provides a logical separation of data from other tenants' data in one and the same virtual environment.
JSON Web Token (JWT)	A token format defined by the OAuth 2.0 specification. A JWT token carries authentication and authorization information about a principal.
Relying party	A relying party "relies" on the authorization server, VMware Identity Services or AD FS, for identity management. For example, through federation, vCenter Server establishes relying party trust to VMware Identity Services or AD FS.
Security Assertion Markup Language (SAML)	An XML-based open standard for exchanging authentication and authorization data between parties that is used by vCenter Server. Principals obtain a SAML token from vCenter Single Sign-On and then send it to the vSphere Automation API endpoint for a session identifier.

Authenticate to vCenter Server with vCenter Single Sign-On Credentials

You can use your vCenter Single Sign-On credentials with basic authentication to establish a session with a vCenter Server system.

To perform this task, you must have:

- The IP address or FQDN of your vCenter Server system

- Valid vCenter Single Sign-On user name and password

The single-factor basic authentication is the simplest method to authenticate to vCenter Server. You connect to the vSphere Automation API endpoint with your vCenter Single Sign-On user name and password. The vSphere Automation endpoint checks whether the user credentials are present in vmdir, a vCenter Single Sign-On component. On success, the system returns a session identifier that you can use for subsequent API calls.

1. Call the Session interface with the base-64 encoded value of your vCenter Single Sign-On user name and password separated by a colon (`username:password`) in the authorization header.

   ```
   POST https://<vcenter_server_ip_address_or_fqdn>/api/session

   Authorization: Basic <username:password>
   ```

   On success (status code 201), the system returns a session identifier. For example, `"a5adf880fc8e1e2944a541a4c7d74465"`.
2. To authenticate subsequent API calls, pass the session identifier as a request header.

   ```
   vmware-api-session-id: <session_ID>
   ```

   IMPORTANT
   VMware encourages you to refrain from using basic authentication and adopt token-based authentication, as it is more secure and provides more flexibility.

Identity Provider Federation for vCenter Server

Starting with vSphere 7.0, you can replace through federation your built-in vCenter Server identity source with an external identity source. Depending on your vCenter Server version, you can use external identity providers such as AD FS, Azure AD, or Okta.

With identity federation, you can use the same corporate identity source for your vCenter Server that you use for your other federated desktop and cloud applications.

vCenter Server Identity Provider Federation Basics

In vSphere 7.0 and later, vCenter Server supports federated authentication. In this scenario, when a user logs in to vCenter Server, vCenter Server redirects the user login to the external identity provider. The user credentials are no longer provided to vCenter Server directly. Instead, the user provides credentials to the external identity provider. vCenter Server trusts the external identity provider to perform the authentication. In the federation model, users never provide credentials directly to any service or application but only to the identity provider. As a result, you "federate" your applications and services, such as vCenter Server, with your identity provider.

Why Is Identity Provider Federation Useful

Federating vCenter Server to an enterprise identity provider alleviates the burden of identity management and provides flexible options such as multifactor authentication (MFA), automatic registration and termination of users across services, and many more. Identity provider federation uses token-based authentication and minimizes the risk of bad actors acquiring protected credentials such as user names and passwords. Identity provider federation also helps your organization with compliance as various international standards require MFA to guarantee data security. In addition, with identity provider federation, you can automate vCenter Server user management because you utilize the users and groups from your enterprise identity source.

vCenter Server External Identity Provider Support

vCenter Server supports the following external identity providers:

- AD FS (vSphere 7.0 and later)

- Okta (vSphere 8.0 Update 1 and later)
- Microsoft Entra ID, formerly called Azure AD (vSphere 8.0 Update 2 and later)
- PingFederate (starting in vSphere 8.0 Update 3)

Federation to Microsoft Active Directory Federation Services (AD FS)

In vSphere 7.0 and later, you can activate identity federation to Microsoft Active Directory Federation Services (AD FS). In this scenario, vCenter Server federates directly to the enterprise identity provider, AD FS, without the use of an authentication intermediary service.

You can configure federation to AD FS with the help of the vSphere Automation API. For more information, see Federate vCenter Server to Microsoft Active Directory Federation Services (AD FS).

Federation to Okta and Azure AD Through VMware Identity Broker - vCenter Server

Starting from vSphere 8.0 Update 1, you can activate federation to Okta as the identity provider. Starting from vSphere 8.0 Update 2, you can activate federation to Azure AD as the identity provider. Both configurations use VMware Identity Broker - vCenter Server, an authentication intermediary that functions as a built-in container within vCenter Server. With VMware Identity Broker - vCenter Server, you can configure principals to authenticate to vCenter Server by using an external identity provider.

vCenter Server can support the default `vsphere.local` identity source and only one configured external identity provider. You cannot use multiple external identity providers.

Once you set up your federated authentication, you can use OAuth for login to vCenter Server.

> **NOTE**
> You can federate vCenter Server to Okta or Azure AD by using the vSphere Client. For more information, see Configure vCenter Server Identity Provider Federation for Okta and Configure vCenter Server Identity Provider Federation for Microsoft Entra ID in the *vSphere Authentication Guide*.

What is OAuth 2.0

OAuth 2.0 is an open authorization framework that allows information exchanges among web services without exposing sensitive data such as user names and passwords. OAuth 2.0 boosts security and prevents malicious actors from stealing users' credentials over the Internet. OAuth 2.0 is an open framework and can be customized according to the organizational requirements.

You can use different OAuth 2.0 grant types to authenticate your applications to vCenter Server. The OAuth grant types represent different methods by which a principal can obtain an access token from an authorization server. In the vSphere implementation, the access token is used to obtain a SAML token and authenticate principals to vCenter Server.

OAuth 2.0 Grant Types

Password

In the Password grant type, the client application exchanges the user's credentials, user name and password, for an access token, and optionally an ID token (JWT tokens), from the authorization server. OAuth 2.0 has been designed to prevent the exchange of user credentials, so the Password grant type must be avoided whenever possible, especially for third-party application authentication. The latest *OAuth 2.0 Security Best Current Practice* recommends avoiding the Password grant type altogether. As a single-factor method, the Password grant type is incompatible with the requirements of multifactor authentication (MFA). For more information, see Authenticate to an AD FS-Federated vCenter Server by Using the Password Grant Type.

> **IMPORTANT**
> The Password grant type is disallowed by the latest *OAuth 2.0 Security Best Current Practice*. The Password grant is excluded entirely from OAuth 2.1.

Authorization Code

In the Authorization Code grant type, the client application uses an authorization code to obtain JWT tokens from the authorization server. This grant type requires a user to verify an authentication request through a web browser. The Authorization Code grant type is compatible with multi-factor authentication and guarantees a substantial degree of security. This grant type is appropriate for regular web applications running on a server. For more information, see Authenticate to an AD FS - Federated vCenter Server by Using the Authorization Code Grant Type.

Client Credentials

The Client Credentials grant type permits web services to authenticate to other web services without the involvement of a user. This grant type is useful for applications that are resource owners and can take advantage of headless, or backend automation, when a physical user is not present. For example, a client application might use this grant type to authenticate and fetch details about a virtual environment.

Refresh Token

The Refresh Token grant type is used by clients to exchange an expiring access token for a refresh token that can be used to receive a new access token. With this grant type, a client application can fetch a valid access token without asking the user to log in again. It allows client applications to provide a seamless user experience without further interaction with the user after timeouts.

What Is OpenID Connect (OIDC)

The vSphere implementation of OAuth 2.0 uses the OpenID Connect (OIDC) authentication protocol. With OIDC, the authorization server returns an ID token containing data about who the users are and what permissions they have on the resource server. As a result, vCenter Server can identify users and grant them the respective permissions that are defined on the authorization server.

AD FS Federation Workflows

You can federate your vCenter Server to Microsoft Active Directory Federation Services (AD FS) and use the OAuth 2.0 grant types to authenticate.

Federate vCenter Server to Microsoft Active Directory Federation Services (AD FS)

You can federate vCenter Server to Microsoft Active Directory Federation Services (AD FS) as an external identity provider by using the vCenter Server `Identity Providers` service.

> **NOTE**
> This process of configuring an AD FS identity provider requires that you have administrative access to both your vCenter Server and your AD FS server. During the configuration process, you enter information first in your vCenter Server, then in your AD FS server, then in your vCenter Server.

Active Directory Federation Services requirements:

- AD FS for Windows Server 2016 or later must already be deployed.
- AD FS must be connected to Active Directory.
- An Application Group for vCenter Server must be created in AD FS as part of the configuration process. See the VMware knowledge base article at https://kb.vmware.com/s/article/78029.
- An AD FS server certificate (or a CA or intermediate certificate that signed the AD FS server certificate) that you add to the Trusted Root Certificates Store.
- You have created a vCenter Server administrators group in AD FS that contains the users you want to grant vCenter Server administrator privileges to.

For more information about configuring AD FS, see the Microsoft documentation.

vCenter Server and other requirements:

- vSphere 7.0 or later

- vCenter Server must be able to connect to the AD FS discovery endpoint, and the authorization, token, logout, JWKS, and any other endpoints advertised in the discovery endpoint metadata.
- You need the **VcIdentityProviders › Manage** privilege to create, update, or delete a vCenter Server Identity Provider that is required for federated authentication. To limit a user to view the Identity Provider configuration information only, assign the **VcIdentityProviders › Read** privilege.

1. Authenticate to the vSphere Automation API endpoint and establish a session.
2. Add your AD FS root CA certificate to the Trusted Root Certificates Store.

   See Managing Certificates.

3. Add the identity provider by using the service.
4. Configure vCenter Server permissions for Active Directory users or groups in your AD FS environment.

   You can do this in two ways:

   a)
   - Add a user from your AD FS environment to a group in vCenter Server.
   - Configure Global Permissions for an AD FS user.

   NOTE
   In vSphere 8.0 and later, you cannot configure permissions through the vSphere Automation API. Instead, you use either the vSphere Client or the vSphere Web Services API. For more information, see the *vSphere Authentication Guide* or the *vSphere Web Services SDK Programming Guide*.

5. Copy the two redirect URIs from the Identity Provider Configuration page in the vSphere Client and add them to your AD FS Application Group.

   NOTE
   You must do this step to enable logging in to vCenter Server through AD FS by using the vSphere Client.

You configured vCenter Server to use AD FS as the identity provider.

Authenticate to an AD FS-Federated vCenter Server by Using the Password Grant Type

Authenticate by Using the Password Grant Type
If your vCenter Server is federated to AD FS, you can authenticate with the OAuth 2.0 Password grant type.

- Verify that your vCenter Server is federated to AD FS. For more information, see *Federate vCenter Server to Active Directory Federation Services (AD FS)*.
- You must have an AD FS account and user credentials with the necessary permissions to view and manage vCenter Server.
- You must register an OAuth client for your application on the authorization server (AD FS).

You can use the Password grant type to exchange user credentials for an access token and an ID token from the authorization server. From the user's perspective, the password grant type functions exactly as basic authentication with the only difference that you use your AD FS, and not your local vCenter Single Sign-On credentials.
The Password grant type is possible among native apps but is not recommended for authentication to third-party apps. Modern security best practices require that primary user credentials such as passwords do not leave the native API environment. Therefore, it is more secure to use other OAuth 2.0 grant types for authentication to third-party apps.
To use the Password grant type with AD FS, you provide your AD FS user name and password to the `Common Infrastructure Services (CIS) Session` endpoint.

In the background, the client application calls the authorization server (AD FS) and obtains an access token and an ID token in JWT format. vCenter Single Sign-On converts the JWT tokens into a SAML token which is used to obtain a session identifier and authenticate your application to vCenter Server.

IMPORTANT
The Password grant type is disallowed by the latest *OAuth 2.0 Security Best Current Practice*. The Password grant is excluded entirely from OAuth 2.1.

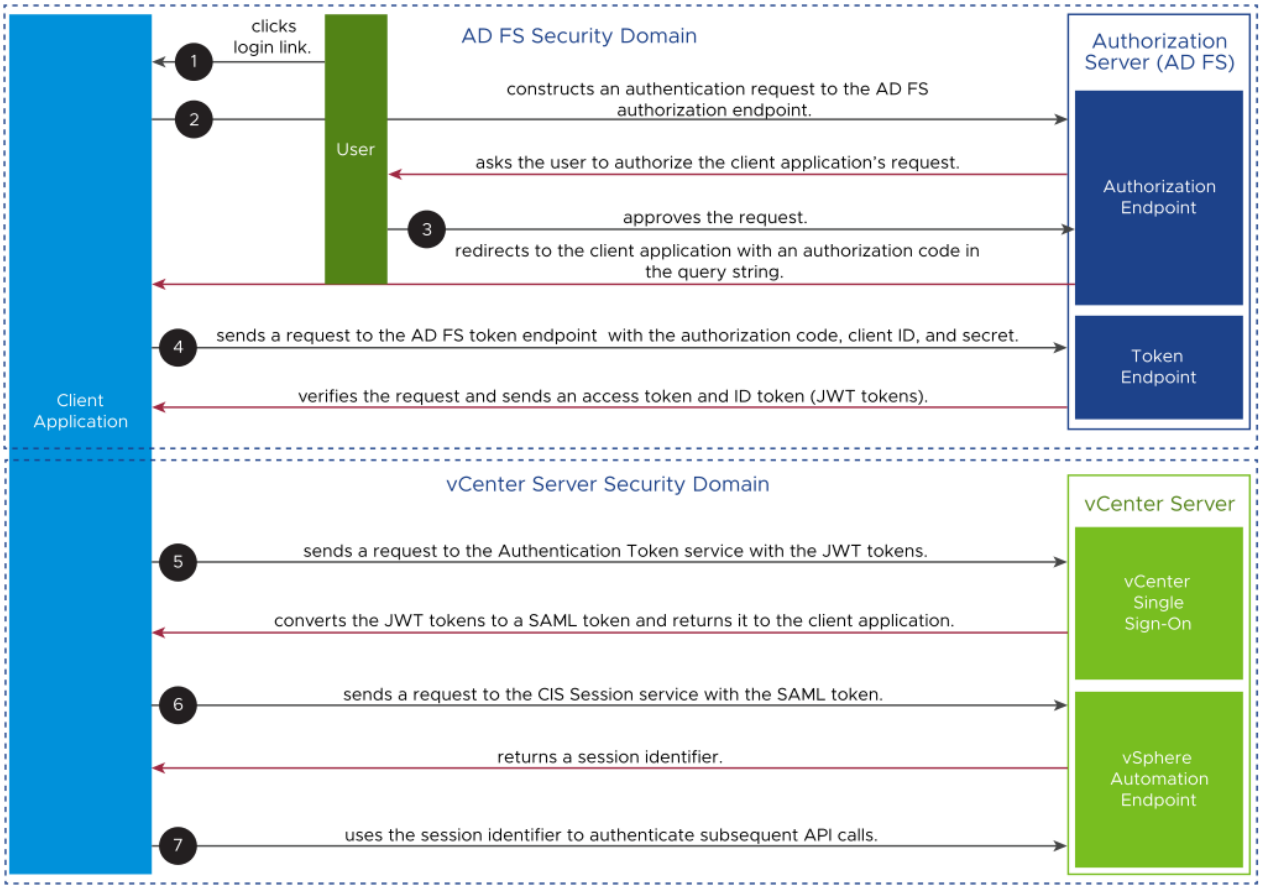
Authenticate to an AD FS-Federated vCenter Server by Using the Authorization Code Grant Type

Authenticate by Using the Authorization Code Grant Type
If your vCenter Server is federated to an enterprise identity provider, such as Active Directory Federation Services (AD FS), you can authenticate your client applications by using the OAuth 2.0 Authorization Code grant type.

- Verify that your vCenter Server is federated to AD FS. For more information, see *Federate vCenter Server to Active Directory Federation Services (AD FS).*
- You must have an AD FS account and user credentials with the necessary permissions to view and manage vCenter Server.
- You must register an OAuth client for your application on the authorization server (AD FS).

You can use the Authorization Code grant type if you want to authenticate client apps to vCenter Server through multifactor authentication (MFA). This grant type is applicable for regular web applications that involve a human user interacting with a server.

Figure 90: Authentication to an AD FS-Federated vCenter Server Through the Authorization Code Grant Type



A principal authenticates to vCenter Server through the Authorization Code grant type as follows.

1. The user clicks **Login** within the client application.
2. The client application constructs a request to the AD FS authorization endpoint.

   To retrieve the authorization endpoint, you must query the AD FS discovery endpoint at, for example, `https://<your_adfs_server.com>/adfs/.well-known/openid-configuration`.

   The client application's request must include the application's client ID and a redirect URI that tells the authorization server where to redirect the user after approving the request.

   As a result, the authorization server opens a prompt page asking the user to authorize the client application's request.
3. The user confirms the request.
   The browser is redirected back to the client application with an authorization code in the query string.
4. The client application sends a request to the AD FS token endpoint with the authorization code, client ID, and secret.
   The authorization server verifies the request and sends an access token and an ID token (JWT tokens).
5. The client application sends the JWT tokens to vCenter Single Sign-On by using the .
   vCenter Single Sign-On returns a SAML token.
6. The client application calls the `Common Infrastructure Services (CIS) Session` service.

   By using the `CIS Session` service, principals can manage session tokens - including creation and deletion of session identifiers, and obtaining information about sessions.

   a) Base-64 decode the SAML token, compress it, and then Base-64 encode it again.
   b) Call the with the SAML token.

   On success (status code 201), the vSphere Automation endpoint returns a session identifier in the response.
7. To authenticate, the client application uses the session identifier in subsequent API calls to the vSphere Automation API endpoints.

VMware Identity Broker - vCenter Server Workflows

You can federate your vCenter Server to an external identity provider through VMware Identity Broker - vCenter Server. It is a built-in container within vCenter Server that enables federation to an external identity provider such as Okta or Azure AD.

What Is VMware Identity Broker - vCenter Server

VMware Identity Broker - vCenter Server is VMware's specialized authentication solution that allows you to federate your apps to external identity providers such as Okta or Azure AD. VMware Identity Broker - vCenter Server functions as a container built-in within your vCenter Server. VMware Identity Broker - vCenter Server comes with its own API Interface that is separate from the vSphere Automation API and the vSphere Web Services API.

You can use VMware Identity Broker - vCenter Server to federate to:
- Okta (starting in vSphere 8.0 Update 1)
- Azure AD (starting in vSphere 8.0 Update 2)

Configure vCenter Server Identity Provider Federation to Okta or Azure AD

To configure your vCenter Server to point to Okta or Azure AD as the identity provider, you must use the Okta or Azure AD interface and the vSphere Client. This integration uses VMware Identity Broker - vCenter Server. For more information, see Configure vCenter Server Identity Provider Federation for Okta and Configure vCenter Server Identity Provider Federation for Microsoft Entra ID from the *vSphere Authentication Guide*.

OAuth Authentication to Your Federated vCenter Server, App, or Script

Once configured, you can use the vSphere Automation and the VMware Identity Broker - vCenter Server APIs to authenticate to your federated vCenter Server, application, or script.

You can use the following OAuth 2.0 grant types:

- Password (not recommended)
- Authorization Code
- Client Credentials
- Refresh Token

Federate Apps or Scripts to an External Identity Provider by Using Your VMware Identity Broker - vCenter Server Configuration

Configure a client application or script for authentication by using an already existing vCenter Server identity provider configuration on VMware Identity Broker - vCenter Server .

Okta and Azure AD requirements:

- You are a customer of Okta or Microsoft and have a dedicated Okta domain space or an Azure AD account.
- You have identified the Okta or Azure AD users and groups that you want to share with your app.

vCenter Server and other requirements:

- vSphere 8.0 Update 1 or later, with the VMware Identity Broker - vCenter Server activated (they are activated by default).
- A vCenter Server that is federated to Okta or Azure AD. (You must know the IP address or the FQDN of the vCenter Server.)
- The users and groups from the identity provider are provisioned in your vCenter Server.
- You must know the tenant that you want to use for your client app or script. The default tenant that is created during vCenter Server federation is called `CUSTOMER`.
- You must have the **VcIdentityProviders › Manage** privilege.

You can leverage the identity provider configuration you created for vCenter Server for other apps or scripts that you want to federate to Okta or Azure AD.
For this automation scenario you must use services from the vSphere Automation API and the VMware Identity Broker - vCenter Server API. The VMware Identity Broker - vCenter Server API endpoints are located also on your vCenter Server but are taxonomically separated from the vSphere Automation API endpoints.

1. Get a session identifier for the vSphere Automation API.

   For more information see Authentication Mechanisms.

2. Get a tenant admin client token.

   The default `tenant` that is created during the vCenter Server federation is called `CUSTOMER`. You can use this tenant for your app or script.

   In the response, you get an access token in JWT format that you use to authenticate to VMware Identity Broker - vCenter Server.

3. Create an OAuth 2.0 client for your app.

   To authenticate to the external identity provider, you must create an OAuth 2.0 client for your app on VMware Identity Broker - vCenter Server. You can do this by using the Create Broker OAuth 2 Client service of the VMware Identity Broker - vCenter Server API.

   For authentication, you must use the tenant admin client token obtained in the previous step.

Example request:

```
curl --location --request POST 'https://<vcenter_server_fqdn>/acs/t/<tenant>

/broker/oauth2-clients' \

--header 'Content-Type: application/
vnd.vmware.horizon.manager.accesscontrol.broker.oauth2client.with.rule.sets+json' \

--header 'Accept: application/
vnd.vmware.horizon.manager.accesscontrol.broker.oauth2client.with.rule.sets+json' \

--header 'Authorization: HZN <my_admin_client_token>' \

--data-raw '{

    "client_id": "<my_app_client_id>",

    "grant_types": [

        "refresh_token",

        "client_credentials",

        "password",

        "authorization_code"

    ],

    "scope": [

        "email",

        "profile",

        "user",

        "openid",

        "group"

    ],

    "secret": "<my_app_client_secret>",

    "redirect_uris": [

        "https://<my_app_redirect_uri1>",

        "https://<my_app_redirect_uri2>"

    ]

}'
```

On success (status code 201), the operation returns the data about the new OAuth 2.0 client.

Authenticate your app or script to the external identity provider by using the OAuth 2.0 grant types.

Obtain JWT Tokens from VMware Identity Broker - vCenter Server by Using the Password Grant Type

Obtain JWT Tokens by Using the Password Grant Type

If your vCenter Server or other application is federated to an external identity provider, such as Okta or Azure AD, through VMware Identity Broker - vCenter Server, you can obtain an access and ID token in JWT format by using the OAuth Password grant type.

- Verify that your vCenter Server or client app is federated to an external identity provider through VMware Identity Broker - vCenter Server.
- You must create an OAuth client for your vCenter Server or client app on VMware Identity Broker - vCenter Server. You must know the client id, client secret, and the redirect URIs for the OAuth client.
- You must know the tenant name for the configuration you want to use. The default tenant that is created during vCenter Server federation is called *CUSTOMER*.
- You must have an Okta or Azure AD account and user credentials with the necessary permissions to view and manage vCenter Server.

You can use the Password grant type to exchange user credentials for an access token and an ID token from the authorization server. From the user's perspective, the password grant type functions similarly to local domain authentication with the difference that principals use their external identity provider, and not their local vCenter Single Sign-On credentials, to authenticate.
The Password grant type involves the client collecting the principal's user name and password directly and exchanging them for an access token from the authorization server. From a security perspective, handling user credentials directly increases the risk of exposure and compromise. From a user experience perspective, users might be reluctant to expose their credentials to third-party apps, as it goes against the principle of separating authentication from authorization. Therefore, using the OAuth 2.0 Password grant type is not recommended.

IMPORTANT
The Password grant type is disallowed by the latest *OAuth 2.0 Security Best Current Practice*. The Password grant is excluded entirely from OAuth 2.1.

1. Obtain an access and ID token.

   Send a request to the VMware Identity Broker - vCenter ServerGet Access Token endpoint with the required grant type, your app's client ID and secret, and the user name and password of a registered user.

   In the authorization header, use the base-64 encoded value of your app's client ID and secret separated by a colon (`client_id:secret`).

   The content type must be *x-www-form-urlencoded*.

   Example request:

```
curl --location --request POST 'https://<vcenter_server_fqdn>/acs/t/<tenant>/token' \
--header 'Authorization: Basic <my_app_client_id>:<my_app_client_secret>' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=password' \
--data-urlencode 'username=<my_user_name>' \
--data-urlencode 'password=<my_password>'
```

   The authorization server returns the JWT tokens corresponding to the grant type: access token , ID token, and refresh token.

Exchange the JWT tokens for a vCenter Server SAML token that you can use to establish an authentication session. For more information, see Exchange JWT Tokens for a SAML Token and Obtain a Session Identifier.

Obtain JWT Tokens from VMware Identity Broker - vCenter Server by Using the Authorization Code Grant Type

Obtain JWT Tokens by Using the Authorization Code Grant Type
If your vCenter Server or other application is federated to an external identity provider, such as Okta or Azure AD, through VMware Identity Broker - vCenter Server, you can obtain an access and ID token in JWT format by using the OAuth 2.0 Authorization Code grant type.

- Verify that your vCenter Server or client app is federated to an external identity provider through VMware Identity Broker - vCenter Server.
- You must create an OAuth client for your vCenter Server or client app on VMware Identity Broker - vCenter Server. You must know the client id, client secret, and the redirect URIs for the OAuth client.
- You must know the tenant name for the configuration you want to use. The default tenant that is created during vCenter Server federation is called *CUSTOMER*.
- You must have an Okta or Azure AD account and user credentials with the necessary permissions to view and manage vCenter Server.

The Authorization Code grant type is appropriate for regular web applications that involve a human user interacting with a server, especially if you want to activate multifactor authentication (MFA). The Authorization Code grant type protects user credentials as they do not leave the API environment.

1. Obtain an authorization code.
   a) Construct a query request in the browser with the following parameters.

client_id	The client identifier you obtained when you registered the OAuth 2.0 client on VMware Identity Services.
redirect_uri	The callback endpoint in your application that must receive the authorization code. It must be one of the `redirect_uris` defined in your OAuth 2.0 client registration. When sending the `redirect_uri` as a URL parameter it must be URL-encoded.
state	A random string that your application generates and that is sent back as a parameter during the URI redirect. Serves as a security mechanism as per the OAuth 2.0 specification.
response_type	Specifies what must be included in the authorization response. For the authorization code grant type, use *code*.

   b) Send the query request to the VMware Identity Broker - vCenter ServerAuthorize endpoint.

   For example:

   ```
   https://<vcenter_server_fqdn>/acs/t/<tenant>/authorize?client_id=<my_app_client
   _id>&redirect_uri=https://<my_app_redirect_uri1>&state=<generated_random_string>
   &response_type=code
   ```

   VMware Identity Broker - vCenter Server opens a dialog box asking the user to authenticate.

   c) The user authenticates to the identity provider.

   The authentication mechanism depends on the specific user configuration on the external identity provider website. It could be basic authentication, biometrics, multifactor authentication, and so on.

   The authorization server directs the user to the client app's redirect URI with the authorization code in the query string.

2. Obtain an access and ID token.

   Send a request to the VMware Identity Broker - vCenter ServerGet Access Token endpoint containing the required grant type, the authorization code, the redirect URI, and your app's client ID and secret.

   In the authorization header, use the base-64 encoded value of your app's client ID and secret separated by a colon (*my_app_client_id*:*my_app_client_secret*).

   The content type must be *x-www-form-urlencoded*.

   Example request:

   ```
   curl -- location --request POST 'https://<vcenter_server_fqdn>/acs/t/<tenant>/token'
   \
   --header 'Authorization: Basic <my_app_client_id>:<my_app_client_secret>' \
   --header 'Content-Type: application/x-www-form-urlencoded' \
   --data-urlencode 'grant_type=authorization_code' \
   --data-urlencode 'code=<my_app_authorization_code>' \
   --data-urlencode 'redirect_uri=https://<my_app_redirect_uri1>'
   ```

   The authorization server returns the JWT tokens corresponding to the grant type: access token , ID token, and refresh token.

Exchange the JWT tokens for a vCenter Server SAML token that you can use to establish an authentication session. For more information, see Exchange JWT Tokens for a SAML Token and Obtain a Session Identifier.

Obtain JWT Token from VMware Identity Broker - vCenter Server by Using the Client Credentials Grant Type

Obtain JWT Token by Using the Client Credentials Grant Type
If your vCenter Server, app, or script is federated to an external identity provider, such as Okta or Azure AD, through VMware Identity Broker - vCenter Server, you can obtain an access token in JWT format by using the OAuth Client Credentials grant type.

- Verify that your vCenter Server or client app is federated to an external identity provider through VMware Identity Broker - vCenter Server.
- You must create an OAuth client for your vCenter Server or client app on VMware Identity Broker - vCenter Server. You must know the client id and secret for the OAuth client.
- You must know the tenant name for the configuration you want to use. The default tenant that is created during vCenter Server federation is called *CUSTOMER*.

You can use the Client Credentials grant type if you want to authenticate vCenter Server or another application to your external identity provider without the presence of a human user. The Client Credentials grant type is appropriate for headless automation when you want a web service to authenticate automatically against an identity provider.

1. Obtain an access token for your app.

   Send a request to the VMware Identity Broker - vCenter ServerGet Access Token endpoint with the required grant type.

In the authorization header, use the base-64 encoded value of your app's client ID and secret separated by a colon (`client_id:secret`).

The content type must be *`x-www-form-urlencoded`*.

Example request:

```
curl --location --request POST 'https://<vcenter_server_fqdn>/acs/t/<tenant>/token' \
--header 'Authorization: Basic <my_app_client_id>:<my_app_client_secret>' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=client_credentials'
```

The authorization server returns the JWT token corresponding to the grant type: access token.

Exchange the JWT token for a vCenter Server SAML token that you can use to establish an authentication session. For more information, see Exchange JWT Tokens for a SAML Token and Obtain a Session Identifier.

Exchange JWT Tokens for a SAML Token and Obtain a Session Identifier

You can exchange the JWT tokens that you obtained from the authorization server for a vCenter Server SAML token and create an authenticated session.

- You must have a valid access and ID token (in JWT format) from the authorization server (AD FS or VMware Identity Broker - vCenter Server) for the user or application you want to authenticate.

1. Prepare the SAML token for the API call.

   Base-64 decode the SAML token, compress it, and then Base-64 encode it again.

2. To obtain a session identifier, call the with the SAML token.
3.

You created an authenticated session and now you can access and use the vSphere Automation services.

Bash Shell

This Bash script illustrates the use case where you get JWT tokens from the authorization server (AD FS or VMware Identity Broker - vCenter Server), exchange them for a vCenter Server SAML token, and obtain an authentication session identifier from the vSphere Automation endpoint.

This script consists of three parts:

1. Obtain JWT tokens from the authorization server (AD FS or VMware Identity Broker - vCenter Server) by using the OAuth 2.0 Password grant type.
2. Exchange the JWT tokens for a vCenter Server SAML token.
3. Use the SAML token to obtain a session identifier for the vSphere Automation API.

```
#!/bin/bash


: '
```

```
Variable definitions:

$vcip = The IP address or FQDN of your vCenter Server.

$ACCESS_TOKEN = The access token in JWT format that you received from the
authentication server.

$ID_TOKEN = The ID token in JWT format that you received from the authentication
server.

'

if [ -z "$1" ] || [ -z "$2" ] || [ -z "$3" ] || [ -z "$4" ] || [ -z "$5" ] || [ -z
"$6" ]; then

  echo "Usage: <vc-ip> <pwgrant-userid> <pwgrant-password> <client-id> <client-
secret> <token-endpoint>"

  exit 0

fi



vcip="$1"

userid="$2"

password="$3"

clientid="$4"

clientsecret="$5"

tokenendpoint="$6"


echo "Obtaining JWT access and ID tokens for user $userid ..."



PWGRANT_OUTPUT=$(curl -k --silent --location -u "$clientid:$clientsecret" --request
POST "$tokenendpoint" \

  --header "Content-Type: application/x-www-form-urlencoded" \

  --data-urlencode "grant_type=password" \

  --data-urlencode "username=$userid" \

  --data-urlencode "password=$password")

ACCESS_TOKEN=$(echo $PWGRANT_OUTPUT | jq -r '.access_token')

ID_TOKEN=$(echo $PWGRANT_OUTPUT | jq -r '.id_token')



echo

echo "Access token: $ACCESS_TOKEN"
```

```
echo

echo "ID token: $ID_TOKEN"

echo


echo "Exchanging JWT tokens for vCenter SAML token ..."


TOKEN_EXCHANGE_OUTPUT=$(curl -k --silent --location --request POST "https://$vcip/
api/vcenter/authentication/token" \

  --header "Content-Type: application/x-www-form-urlencoded" \

  --header "Authorization: Bearer $ACCESS_TOKEN" \

  --data-urlencode "subject_token_type=urn:ietf:params:oauth:token-type:access_token"
\

  --data-urlencode "subject_token=$ACCESS_TOKEN" \

  --data-urlencode "actor_token_type=urn:ietf:params:oauth:token-type:id_token" \

  --data-urlencode "actor_token=$ID_TOKEN" \

  --data-urlencode "grant_type=urn:ietf:params:oauth:grant-type:token-exchange" \

  --data-urlencode "requested_token_type=urn:ietf:params:oauth:token-type:saml2")


echo

echo "$TOKEN_EXCHANGE_OUTPUT"

echo


SAML_TOKEN=$(echo $TOKEN_EXCHANGE_OUTPUT | jq -r '.access_token')


echo "vCenter SAML token: $SAML_TOKEN"

echo


echo "Establishing vCenter session with SAML token ${SAML_TOKEN::7}..."

echo

COMP_TOKEN=$(echo $SAML_TOKEN | base64 -d | gzip | base64 -w0)


SESSION_OUTPUT=$(curl -k --silent --location --request POST "https://$vcip/api/
session" \
```

```
  --header "Authorization: SIGN token=\"$COMP_TOKEN\"")


echo "Create Session Response: $SESSION_OUTPUT"


SESSION_ID=$(echo "$SESSION_OUTPUT" | tr -d '"')


echo "Tagging categories:"

curl -k "https://$vcip/api/cis/tagging/category" --header "vmware-api-session-id:
$SESSION_ID"


echo

echo "Done!"
```

ESXi Hosts

Use the vSphere Automation APIs to run general operations on the ESXi hosts in your vSphere environment.

You can retrieve information about the hosts, create a standalone host, disconnect, and reconnect an ESXi host to a vCenter Server system.

How to Retrieve Information About ESXi Hosts

You retrieve information about the ESXi hosts running in a vCenter Server instance by listing only the ESXi hosts that you are interested in.

To filter the ESXi hosts on a vCenter Server instance and get only the ones you want, Combine several filter criteria by including one or more of the following parameters:

- The name or unique identifier of the host.
- Clusters, data centers, or folders that contain the host.
- Connection state of the host which can be one of the following: `CONNECTED`, `DISCONNECTED`, or `NOT_RESPONDING`.
- Power state of the host which can be one of the following: `POWERED_OFF`, `POWERED_ON`, or `STANDBY`.

How to Add a Standalone ESXi Host to vCenter Server

You can use the vSphere AutomationAPIs to add a standalone host to a vCenter Server instance.

How to Disconnect and Reconnect ESXi Hosts

You can use the vSphere Automation APIs to connect ESXi hosts to a vCenter Server instance and make the hosts managed. You can temporarily disconnect a managed host from a vCenter Server instance and reconnect the host, for example, to refresh the ESX agents on the host.

When you add a host to a vCenter Server instance, the host is connected to vCenter Server and becomes a managed host. To disconnect a managed host from a vCenter Server instance, . The managed host and its associated virtual machines remain in the inventory but vCenter Server temporarily stops managing and monitoring them.

To reconnect a managed host to a vCenter Server instance, . As a result, the connection status of the host changes, and vCenter Server resumes managing the host and its associated virtual machines.

If you want to delete a host and all its associated virtual machines from the inventory, you can remove the host from the vCenter Server instance. To delete a disconnected host from a vCenter Server instance, .

How to Configure ESXi Entropy

You can use the vSphere Automation API to feed external entropy data to an ESXi host in your inventory. You can also query the current entropy levels on the host and add external entropy data when needed.

Entropy is a measure of the randomness or diversity of a data-generating function. In releases prior to vSphere 8.0 Update 1, ESXi supported only CPU-based entropy data generated from interrupts or manufacturer provided interfaces, such as RDSEED and RDRAND. High-quality entropy is important for the proper functioning of security-related operations such as generating encryption keys for secure communication over the network. Starting with vSphere 8.0 Update 1, you can add external entropy sources to an ESXi host and in this way ensure the high quality of the entropy data on that host. You provide external entropy data by using devices such as hardware security modules (HSMs) which are FIPS 140-3 and EAL4 certified.

You can configure the ESXi entropy sources by using the VMkernel boot options. To use external entropy sources, set the `entropySources` value to more than or equal to `8`. For more information about how to set the desired entropy sources by using the VMkernel boot options, see Controlling ESXi Entropy in the *`vSphere Security`* documentation.

You can also configure external entropy sources in the kickstart file for the ESXi scripted installation. See Configuring External Entropy Sources During Scripted Installation.

> **NOTE**
> If a host is configured to use only external entropy sources, that is, `entropySources` is set to `8`, you must keep supplying the external entropy data through the vSphere Automation API. In case the entropy in the host gets exhausted, the host becomes unresponsive and might require a hard reboot or re-installation to recover the host from this situation.

Querying Entropy Data on a Host

To retrieve details about the external entropy available on an ESXi host, use the `ExternalPool` service. You must have the **Host.Entropy.Read** privilege.

You can check whether an external entropy source is added to a host by .

Table 48: Details for the External Entropy Data on a Host

	Description
	Indicates whether an external entropy source is added for a host.

Table continued on next page

Continued from previous page

	Description
	Shows the maximum capacity of external entropy data in bytes that a host can store in the VMkernel entropy pool. When you feed the host with additional external entropy data, make sure that you do not exceed this maximum capacity. Otherwise, all extra entropy data is discarded.
	Indicates the current amount of entropy data in bytes available in the VMkernel entropy pool on the host.
	Indicates the amount of time in seconds that can elapse without any activity between the host and the external entropy source. Your application must check periodically the levels of entropy data on the host and send entropy data from the external source when required. When this timeout exceeds an `esx.audit.entropy.external.source.disconnected` VMkernel Observation (VOB) is logged. This system event suggests possible loss of connection to the external entropy source. For the full list of available entropy VOBs in vSphere 8.0, see the VMware knowledge base article at https://kb.vmware.com/s/article/89074.
	Indicates the threshold in bits sufficient for the in-memory cache to operate successfully. When the in-memory entropy cache is running low and the threshold is reached, an `esx.audit.entropy.available.low` VOB is logged. For more information about the entropy VOBs, see the VMware knowledge base article at https://kb.vmware.com/s/article/89074.

Adding External Entropy Data to a Host

When the external entropy data reaches the host, the entropy daemon stores it first in the in-memory cache and the storage cache. Then the external entropy data is pushed to the VMkernel entropy pool from which it is fed to the applications in need.

The entropy data in the storage cache persists on the host disk and is only consumed during the ESXi booting. The default storage file size is 4MiB and can be configured through the ESXCLI commands. See the *`ESXCLI Command Reference`* documentation.

Managing the Life Cycle of Hosts and Clusters

You can manage the life cycle of ESXi hosts collectively by using the vSphere Lifecycle Manager feature through the vSphere Automation API. Starting with vSphere 8.0, you can manage the lifecycle of a standalone ESXi host by using an image only through the vSphere Lifecycle Manager automation API.

You can automate the life cycle management of a cluster or standalone host by performing the following operations:

- Retrieve information about the current state of the cluster or the standalone host.
- Create a desired state that includes a specific version of the ESXi host. You can also add some compatible partner software and firmware components and add-ons.
- Validate the desired state to detect any discrepancies between the desired state and the host hardware.
- Check the compliance of a cluster or host against the desired state and determine whether some additional steps must be taken to ensure the success of the cluster or host remediation.
- Apply the desired state on a cluster or a standalone host.

You can use the vSphere Lifecycle Manager to manage the life cycle of hosts in a cluster that meet the following requirements:

- Hosts must be of version 7.0 and later.
- Hosts must be stateful.
- All hosts in the cluster must be from the same vendor and with identical hardware.
- The hosts must include only integrated solutions, such as VMware vSAN™, vSphere Supervisor, NSX and VMware vSphere® High Availability.

A standalone host is a host that is added to a vCenter Server instance but is not part of any cluster. For more information about how to add, connect, and disconnect standalone host, see Hosts. You can manage the life cycle of a standalone host by performing almost all vSphere Lifecycle Manager operations that you can perform on a cluster level. The only limitation for managing the life cycle of a standalone host through the vSphere Automation API, is that you can't update the firmware of the host.

vSphere Lifecycle Manager Terms

You must understand the basic terminology that is used within this chapter to be able to use the vSphere Lifecycle Manager functionality efficiently.

vSphere Lifecycle Manager Terminology

Term	Definition
Upgrade, update, and patch	You can upgrade to another major version of the software running on an ESXi host, and install patches and updates that include smaller changes, bug fixes, or other small improvements.
Depot	A depot is a well-defined folder structure that is used for distributing payloads and their metadata. Depots are consumed by different products and features such as the vSphere Lifecycle Manager and ESXCLI. The vSphere Lifecycle Manager works with three types of depots: online, offline, and UMDS. See Software Depots.
Component	A component is the smallest unit that the vSphere Lifecycle Manager uses during the installation and update processes. Software vendors use components to encapsulate a group of payloads that can be managed together.
Base image	A base image is a collection of components that shape the bootable ESXi used for the installation or upgrade process. Base images are currently distributed only by VMware and support x86 servers. VMware provides new versions of the base image for each upgrade, update, and patch release of the ESXi. Base images are hosted at the VMware online depot that is available by default to the vSphere Lifecycle Manager. Furthermore, you can download a different base image version, in the form of an offline ZIP bundle, from https://my.vmware.com/web/vmware/downloads.
Vendor add-on	An add-on is a collection of components that different OEMs provide on top of a base image. Vendors use an add-on to group some customizations for a family of servers. Unlike base images, add-ons are not complete

Table continued on next page

Continued from previous page

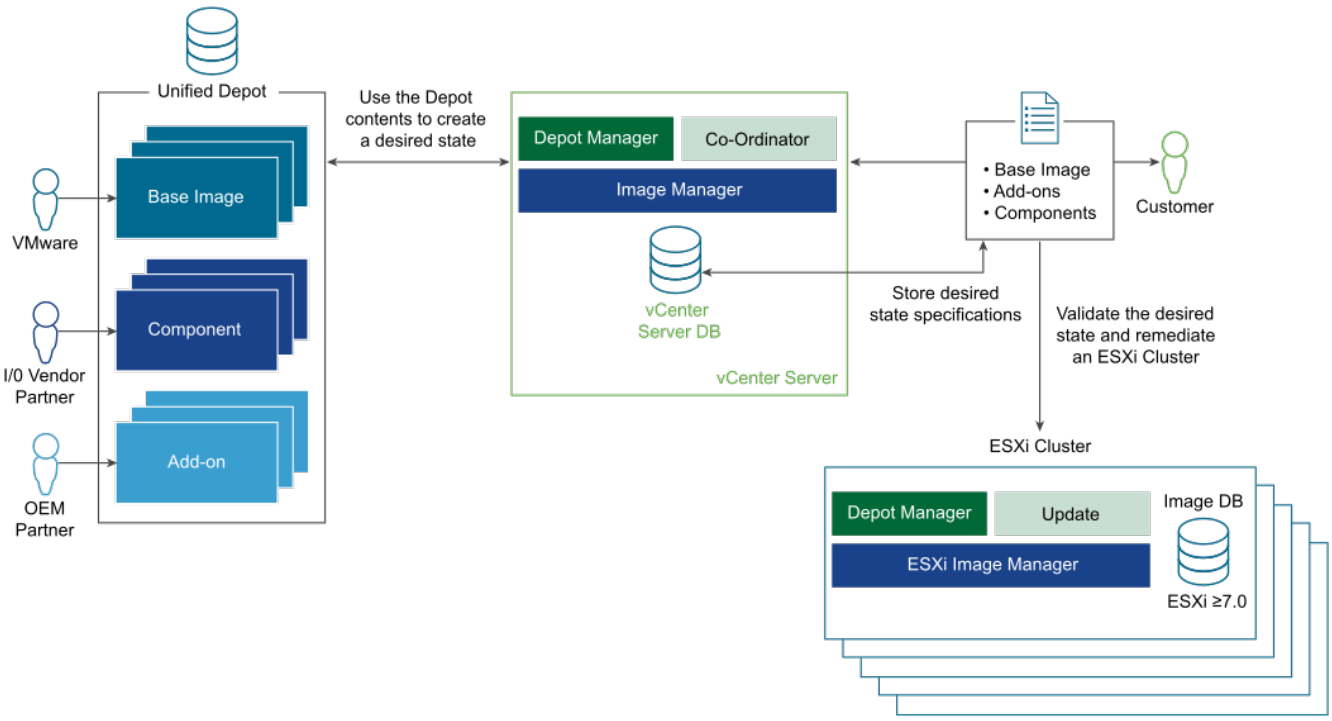
Term	Definition
	and are not sufficient to boot an ESXi. Each add-on must have a unique name and version. An add-on allows vendors to add, remove, or update components that are part of the VMware base image, if there are no unresolved dependencies and conflicts between the components.
Solution	A solution contains one or more components, and provides information about its constraints and compatibility with the different ESXi versions. For example, from the perspective of the vSphere Lifecycle Manager solutions are VMware NSX, VMware vSphere® High Availability, vSAN.
Desired state	A desired state of a cluster is represented with a software specification. The desired state defines a set of components that a user wants to install on a single ESXi host or on a cluster of hosts.
OEMs	Original Equipment Manufacturers. VMware partners enrolled in the VMware Partner Connect application, such as Dell, Inc., HP Inc., Lenovo Group Ltd., and so on.
IOVP	I/O Vendor Partner. Qualified VMware partners providing certified I/O device drivers for network and storage host bus adapters.
Third-party software providers	Providers of I/O filters, device drivers, CIM modules, and so on, that are not part of VMware partner programs.
vSphere Configuration Profiles	With vSphere Configuration Profiles, you can manage the configuration of all hosts in a cluster collectively. This ensures consistency in host configuration at a cluster level. You can enable vSphere Configuration Profiles only on clusters that have the vSphere Lifecycle Manager enabled.

vSphere Lifecycle Manager Overview

The vSphere Lifecycle Manager feature provides means for managing the life cycle of hosts on a cluster level. The functionality of this feature is achieved through several major services that are running on both vCenter Server and ESXi.

After you install vSphere 8.0 Update 1, you can access the feature through several major services available on vCenter Server and on each ESXi host.

Figure 91: vSphere Lifecycle Manager System Architecture



Depot Manager

For each upgrade and patch release, VMware, OEMs, and other third-party companies make the software updates of their products available to the customers. Software updates are distributed to different locations and in different formats depending on the way they are accessed, downloaded, and used. Depot Manager allows these different software resources to be presented to the vSphere customers in a unified format and as a result, makes them easier to use.

Depending on your environment and specific use case, Depot Manager gives you access to the software updates within three different types of depots: online, offline, and UMDS. See Software Depots.

Depot Manager that runs on the vCenter Server instance achieves the following goals:

- Represents the contents of all depots in a unified way.
- Caches the payloads and their metadata locally prior to their use.
- Enables depot overrides on a cluster or host level for Remote Office/Branch Office (ROBO) environments, or Edge computing environments.

Depot Manager that runs on the hosts serves as a proxy to the vCenter Server Depot Manager. If ROBO cluster or ROBO host, or Edge computing environments are enabled, Depot Manager that runs on the ESXi hosts serves as a proxy to the nearby vSphere Lifecycle Manager compatible depot.

Image Manager

Image Manager allows you to create a desired state that you can apply on a cluster or a standalone host. The specification describes all components, add-ons, and the base image that you can use to update or upgrade the hosts in your environment. Image Manager supports validation of the desired state and the detecting drifts from the desired state.

Coordinator and ESXi Updater Managers

Coordinator Manager runs on the vCenter Server instance and makes sure that the desired state is applied to all hosts in the cluster. This module also runs pre-checks to evaluate how each host in the cluster is affected by the remediation and

whether you must take some additional steps to ensure the success of the procedure. Coordinator Manager also allows you to query the status of the remediation operation for the cluster and for each host part of the cluster.

The ESXi Updater Manager takes care of the actual remediation happening on each host.

Options for Managing the ESXi Life Cycle

Based on your needs and environment setup, you can choose from several methods for managing the life cycle of the ESXi hosts. The vSphere Lifecycle Manager provides means for updating all hosts in a cluster or a standalone host with a desired software state.

To manage the life cycle of the hosts in your environment, you can use the vSphere Lifecycle Manager through the vSphere Client. See Managing Host and Cluster Lifecycle.

This chapter of the vSphere Automation Programming Guide discusses how you can access and use the functionality provided by the vSphere Lifecycle Manager Automation API.

vSphere Lifecycle Manager Features

You can use the vSphere Automation API to manage the life cycle of all hosts in a cluster or of standalone hosts by using a vSphere Lifecycle Manager image. You can access and use the following vSphere Lifecycle Manager functionality:

- Depot management. You can add, remove, explore the contents of different types of depots. See Software Depots. The content of the depots is provided by VMware and VMware partners. Partners can use the ESXi Packaging Kit (EPK) to assemble a custom bootable ESXi image. The custom image can then be shared to other third-party customers and used through the Depot Manager service. For more information about how to create custom ESXi images, see *`ESXi Packaging Kit (EPK) Development Guide`*.
- Desired software state. You can create, edit, and delete a desired software state for a cluster or a standalone host on which the vSphere Lifecycle Manager is enabled. A desired software state must contain at least a single ESXi image provided by VMware. You can also set an add-on provided by OEMs, and one or more components by different software vendors. Furthermore, during the process of creating the desired software state, you can check the validity of the specification and compare the current state of the hosts in the cluster or the standalone host with the desired software state.
- Cluster remediation. You can apply the desired state on each of the hosts in a cluster which current state is different from the desired specification. Applying a desired state on a cluster level has the following prerequisites:
  - The cluster must have the vSphere Lifecycle Manager enabled.
  - All hosts in the cluster must store their data on a local or remote disk, or on a USB drive.
  - All hosts in the cluster must be of version 7.0 or higher.
  - All hosts must contain only components that the vSphere Lifecycle Manager can recognize and maintain. If a host contains some old content that the vSphere Lifecycle Manager does not recognize, the content is removed from the host during remediation.
- Standalone host remediation. You can apply the desired software state on a standalone host that is managed with baselines or on a standalone host that is managed with images but its current state differs from the desired state.

Required Privileges for Using vSphere Lifecycle Manager and vSphere Configuration Profiles

To configure vSphere Lifecycle Manager settings and to use successfully vSphere Lifecycle Manager baselines and images, you must have the proper privileges. Similarly, you need the proper privileges to use vSphere Configuration Profiles.

You can assign vSphere Lifecycle Manager and vSphere Configuration Profiles privileges to different roles in the vSphere Client.

vSphere Lifecycle Manager Privileges For Using Images

When you use vSphere Lifecycle Manager images, you need a different set of privileges for each task.

Table 49: VMware vSphere vSphere Lifecycle Manager Privileges For Using Images

Task	Privilege in the vSphere Client	Privilege in the API
Set Up Image	• **VMware vSphere Lifecycle Manager › Lifecycle Manager: Image Privileges › Read** • **VMware vSphere Lifecycle Manager › Lifecycle Manager: Image Privileges › Write** • **VMware vSphere Lifecycle Manager › Lifecycle Manager: Settings Privileges › Read**	• `VcIntegrity.lifecycleSoftwareSpecification.Read` • `VcIntegrity.lifecycleSoftwareSpecification.Write` • `VcIntegrity.lifecycleSettings.Read`
Import Image	• **VMware vSphere Lifecycle Manager › Lifecycle Manager: Image Privileges › Read** • **VMware vSphere Lifecycle Manager › Lifecycle Manager: Image Privileges › Write** • **VMware vSphere Lifecycle Manager › Lifecycle Manager: Settings Privileges › Read** • **VMware vSphere Lifecycle Manager › Upload File › Upload upgrade images and offline bundles**	• `VcIntegrity.lifecycleSoftwareSpecification.Read` • `VcIntegrity.lifecycleSoftwareSpecification.Write` • `VcIntegrity.lifecycleSettings.Read` • `VcIntegrity.FileUpload.com.vmware.vcIntegrity.ImportFile`
Export Image	**VMware vSphere Lifecycle Manager › Lifecycle Manager: Image Privileges › Read**	`VcIntegrity.lifecycleSoftwareSpecification.Read`
Edit Image	• **VMware vSphere Lifecycle Manager › Lifecycle Manager: Image Privileges › Read** • **VMware vSphere Lifecycle Manager › Lifecycle Manager: Image Privileges › Write** • **VMware vSphere Lifecycle Manager › Lifecycle Manager: Settings Privileges › Read**	• `VcIntegrity.lifecycleSoftwareSpecification.Read` • `VcIntegrity.lifecycleSoftwareSpecification.Write` • `VcIntegrity.lifecycleSettings.Read`
Work with Recommendations	• **VMware vSphere Lifecycle Manager › Lifecycle Manager: Image Privileges › Read** • **VMware vSphere Lifecycle Manager › Lifecycle Manager: Image Privileges › Write** • **VMware vSphere Lifecycle Manager › Lifecycle Manager: Settings Privileges › Read**	• `VcIntegrity.lifecycleSoftwareSpecification.Read` • `VcIntegrity.lifecycleSoftwareSpecification.Write` • `VcIntegrity.lifecycleSettings.Read`

Table continued on next page

Continued from previous page

Task	Privilege in the vSphere Client	Privilege in the API
Work with Depot	• **VMware vSphere Lifecycle Manager › Lifecycle Manager: Settings Privileges › Read** • **VMware vSphere Lifecycle Manager › Lifecycle Manager: Settings Privileges › Write**	• `VcIntegrity.lifecycleSoftwareSpecification.Read` • `VcIntegrity.lifecycleSoftwareSpecification.Write`
Manage Depot Overrides	• **VMware vSphere Lifecycle Manager › Lifecycle Manager: Settings Privileges › Read** • **VMware vSphere Lifecycle Manager › Lifecycle Manager: Settings Privileges › Write**	• `VcIntegrity.lifecycleSoftwareSpecification.Read` • `VcIntegrity.lifecycleSoftwareSpecification.Write`
Check Compliance	**VMware vSphere Lifecycle Manager › Lifecycle Manager: Image Privileges › Read**	`VcIntegrity.lifecycleSoftwareSpecification.Read`
Run Remediation Pre-Check	• **VMware vSphere Lifecycle Manager › Lifecycle Manager: General Privileges › Read** • **VMware vSphere Lifecycle Manager › ESXi Health Perspectives › Read** • **VMware vSphere Lifecycle Manager › Lifecycle Manager: Image Remediation Privileges › Read**	• `VcIntegrity.lifecycleGeneral.Read` • `VcIntegrity.lifecycleHealth.Read` • `VcIntegrity.lifecycleSoftwareRemediation.Read`
Remediate Against an Image	• **VMware vSphere Lifecycle Manager › Lifecycle Manager: General Privileges › Read** • **VMware vSphere Lifecycle Manager › ESXi Health Perspectives › Read** • **VMware vSphere Lifecycle Manager › Lifecycle Manager: Image Remediation Privileges › Read** • **VMware vSphere Lifecycle Manager › Lifecycle Manager: Image Remediation Privileges › Write** • **VMware vSphere Lifecycle Manager › Lifecycle Manager: Settings Privileges › Read**	• `VcIntegrity.lifecycleGeneral.Read` • `VcIntegrity.lifecycleHealth.Read` • `VcIntegrity.lifecycleSoftwareRemediation.Read` • `VcIntegrity.lifecycleSoftwareRemediation.Write` • `VcIntegrity.lifecycleSettings.Read`
Edit Remediation Settings	• **VMware vSphere Lifecycle Manager › Lifecycle Manager: Settings Privileges › Read** • **VMware vSphere Lifecycle Manager › Lifecycle Manager: Settings Privileges › Write**	• `VcIntegrity.lifecycleSettings.Read` • `VcIntegrity.lifecycleSettings.Write`

Table continued on next page

Continued from previous page

Task	Privilege in the vSphere Client	Privilege in the API
Update Firmware	• **VMware vSphere Lifecycle Manager › Lifecycle Manager: Image Privileges › Read** • **VMware vSphere Lifecycle Manager › Lifecycle Manager: Image Privileges › Write** • **VMware vSphere Lifecycle Manager › Lifecycle Manager: Settings Privileges › Read**	• `VcIntegrity.lifecycleSoftwareSpecification.Read` • `VcIntegrity.lifecycleSoftwareSpecification.Write` • `VcIntegrity.lifecycleSettings.Write`
View Hardware Compatibility	**VMware vSphere Lifecycle Manager › Lifecycle Manager: Hardware Compatibility Privileges › Access Hardware Compatibility**	`VcIntegrity.HardwareCompatibility.Read`
Stage an Image	**VMware vSphere Lifecycle Manager › Lifecycle Manager: Image Remediation Privileges › Write**	`VcIntegrity.lifecycleSoftwareRemediation.Write`

For more information about managing users, groups, roles, and permissions, see the *vSphere Security* documentation.

Required Privileges for Using vSphere Configuration Profiles

When you use vSphere Configuration Profiles, you need a different set of privileges for each task.

Table 50: Required Privileges For Using vSphere Configuration Profiles

Task	Privilege in the vSphere Client	Privilege in the API	Description
Create Cluster with vSphere Configuration Profiles enabled	**Host › Inventory › CreateCluster**	`Host.Inventory.CreateCluster`	Create a cluster along with its initial configuration.
View Host Settings in the Cluster Configuration	**VMware vSphere Lifecycle Manager › Desired Configuration Management Privileges › Read-only access to desired configuration management platform**	`VcIntegrity.ClusterConfiguration.View`	View the draft configuration document, current configuration settings, compliance reports, and pre-check reports.
Check Cluster Compliance			
View Compliance Results			
Run Remediation Pre-Check			
View Remediation Pre-Check Results			
View Draft Pre-Check Results			
Remediate a Cluster Against the Desired Cluster Configuration	**VMware vSphere Lifecycle Manager › Desired Configuration Management Privileges › Remediate cluster to the desired configuration.**	`VcIntegrity.ClusterConfiguration.Remediate`	Remediate a cluster against the draft configuration or transition a cluster to the desired configuration.
Transition to vSphere Configuration Profiles			

Table continued on next page

Continued from previous page

Task	Privilege in the vSphere Client	Privilege in the API	Description
Export a Configuration or Configuration Schema	**VMware vSphere Lifecycle Manager › Desired Configuration Management Privileges › Export desired cluster configuration**	`VcIntegrity.ClusterConfiguration.Export`	Export the desired cluster configuration document.
Extract a Configuration Document from a Reference Host Create a Draft Configuration for a Cluster Import a Configuration	**VMware vSphere Lifecycle Manager › Desired Configuration Management Privileges › Modify desired cluster configuration.**	`VcIntegrity.ClusterConfiguration.Modify`	Modify the desired cluster configuration by crating a draft configuration, extracting a configuration document from a reference host, or importing a configuration.

Software Depots

A software depot represents a well-defined file structure used for storing and hosting the ESXi software updates, patches, and upgrades that VMware, partners, and third-party vendors provide. You can use the vSphere Automation APIs to manage the life cycle of the hosts in your environment by applying software updates hosted on different depots.

Software depots are managed by the Depot Manager which is part of the vSphere Lifecycle Manager. Software depots contain the actual payloads and the metadata of the software updates. Depending on the way you access the software updates, the Depot Manager recognizes three types of software depots: online, offline, and UMDS.

The following section of the documentation explains the concept of a software depot in terms of the vSphere Lifecycle Manager feature. You can also find common use cases available through the APIs for working with the different types of depots and their content.

Types of Software Depots

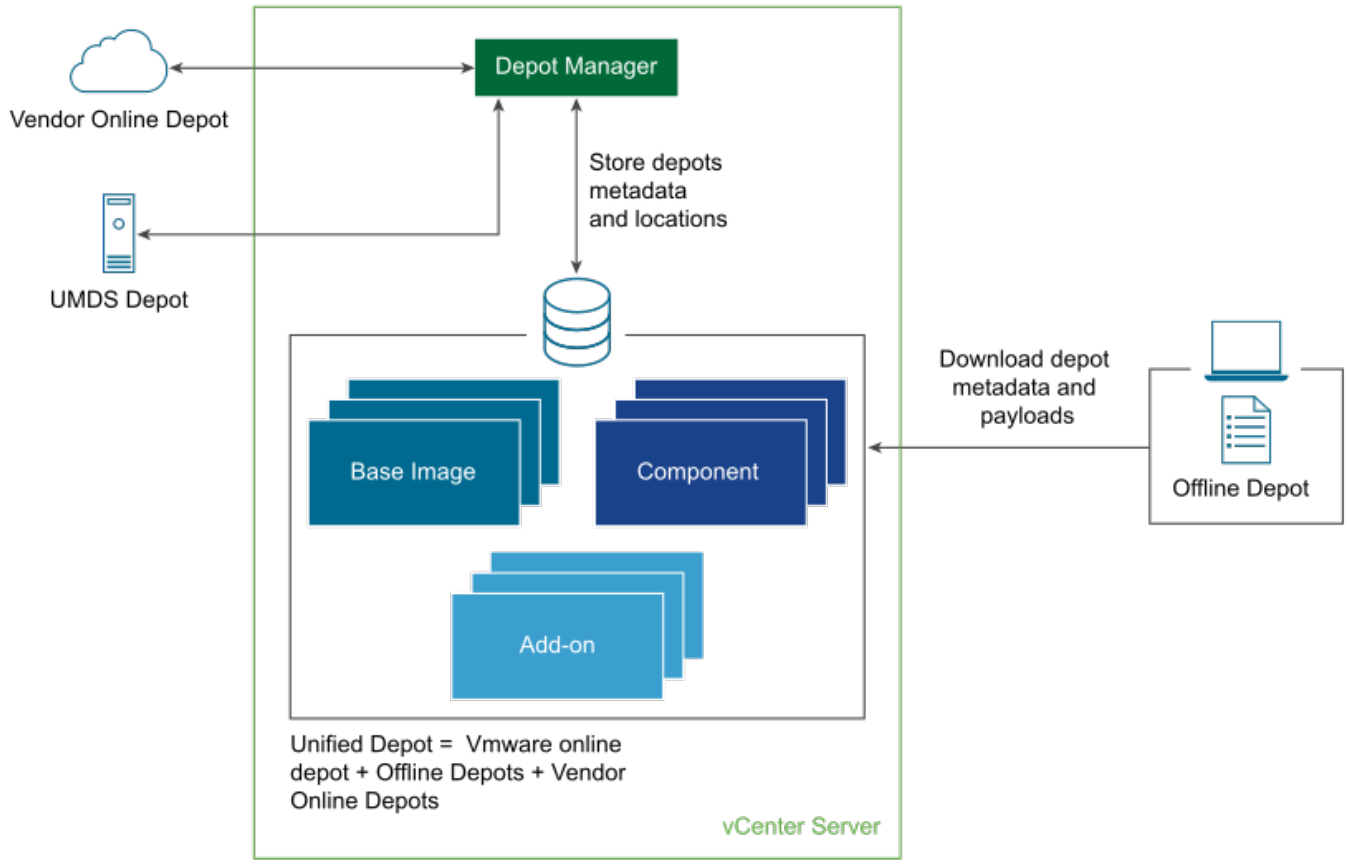
Depot Manager works with the software updates provided by three different types of software depots: online, offline, and UMDS.

Regardless of the different way in which you access each type of software depot, all depots have the same structure. The same depot structure allows content from different vendors to be uploaded to one depot. By default, you can access the content of the VMware online depot at `https://dl.broadcom.com/<Download_Token>/PROD/...`.

> **NOTE**
> The default online depot URLs are now unique for each customer. To update the default URLs of the online depot, see the following KB articles:KB 390098 and KB 390121.

Furthermore, partners and third-party customers can use the ESXi Packaging Kit to build and distribute software updates in the form of offline or online depots. You can access their content by adding the online vendor depot to the vSphere Lifecycle Manager or by downloading the content of the offline depot to the vCenter Server instance.

Figure 92: Types of Software Depots



Online Depot

VMware and partners upload software updates to the VMware online depot at `https://dl.broadcom.com/<Download Token>/PROD/...` or to a custom online depot. Software updates can be patches of the ESXi base image, different versions of the partner add-ons, the IOVP drivers certified by VMware, and the VMware Tools™ updates. Online depots are accessible through a URL. By default, you can see the base images, add-ons, and components provided within the VMware online depot at the following locations:

- `https://dl.broadcom.com/<Download Token>/PROD/COMP/ESX_HOST/main/vmw-depot-index.xml`
- `https://dl.broadcom.com/<Download Token>/PROD/COMP/ESX_HOST/addon-main/vmw-depot-index.xml`
- `https://dl.broadcom.com/<Download Token>/PROD/COMP/ESX_HOST/iovp-main/vmw-depot-index.xml`
- `https://dl.broadcom.com/<Download Token>/PROD/COMP/ESX_HOST/vmtools-main/vmw-depot-index.xml`

When you deploy the vCenter Server, the vSphere Lifecycle Manager is configured to access the VMware online depot, by default. You can use the vSphere Automation APIs to add a custom online depot to be managed by Depot Manager. The metadata of the newly added online depot is not synchronized immediately. To synchronize the metadata, you can run a synchronization operation or wait for the scheduled synchronization to take place.

The Depot Manager stores in the vCenter Server database only the metadata of the software updates and the location of the added online depots. You can create a schedule to synchronize the software updates metadata stored in the vCenter

Server with the metadata available in the accessible depots. The payloads of the software updates are downloaded only during the cluster remediation process.

To add, remove, list, and retrieve information about the online depots, you can use the . See Working with Online Depots.

Offline Depot

The offline depot is also called an offline bundle and is distributed as a downloadable ZIP file. Offline depots contain both the metadata and the payloads of the software update. Partners and third-party customers can use the ESXi Packaging Kit to build and distribute offline bundles. You can download offline bundles from the Broadcom Support Portal Downloads site or from the websites of third-party vendors. When you add an offline depot to the vSphere Lifecycle Manager depot, the software updates are downloaded to the vCenter Server database.

To manage offline depots, you can use the . See Working with Offline Depots.

UMDS Depot

In case, the vCenter Server instance is in an air-gapped environment and has no access to any wire or wireless network, you can use a UMDS depot. The Update Manager Download Service (UMDS) is available as a `VMware-UMDS-8.0.1-build_number.tar.gz` file within the ISO image of the vCenter Server appliance 8.0 UMDS is a 64-bit application and requires a 64-bit Linux-based system. Install UMDS on a machine that has Internet access and is different from the machine on which the vSphere Lifecycle Manager is running. For further information about how to install and configure the UMDS module, see the *Managing Host and Cluster Lifecycle*.

You can set up a synchronization schedule for downloading specific software updates from online vendor depots to the UMDS depot. Then use these updates to create desired software state for the clusters in your environment.

To manage UMDS depots through the vSphere Automation API, you can use the .

Working with Online Depots

You can use the vSphere Automation APIs to add online depots to the list of currently configured online software depots.

Use online depots to add new content over time to the management scope of Depot Manager. Depot Manager periodically updates the software depots metadata stored on the vCenter Server instance. In case, new software updates are uploaded to the online depots, the Depot Manager makes sure that the metadata stored on the vCenter Server database is updated accordingly.

To add an online depot to the Depot Manager, you must first create the online depot specification by using the . To specify the URI to the `vendor-index.xml` file of the online depot, . Optionally, you can add a description and enable the depot. By default, when you add an online depot to the Depot Manager, the depot is enabled and its metadata is synchronized following the defined schedule. If you want to synchronize the added online depot immediately, . When you complete the depot specification, .

You can edit the depot description and deactivate the depot by .

You can remove an online depot from the list of currently configured depots by using the . This operation does not remove the already downloaded metadata and payloads from the deleted depot. You cannot delete the default VMware online depot, you can only deactivate it.

To retrieve a list of currently configured online depots, . You can also retrieve information about a currently configured online depot by using the .

Working with UMDS Depots

In an air-gapped vCenter Server environment, you can use the vSphere AutomationAPI to add a UMDS depot to the depots managed by Depot Manager.

After you install and configure the Update Manager Download Service (UMDS) on a physical machine with Internet access, you can add the UMDS depot to Depot Manager.

> **NOTE**
> The URLs of the official VMware online depot are now unique for each customer. To update the download sources URLs, see the following KB article: KB 390123.

Only one UMDS depot can be added at a time to Depot Manager. When you add a UMDS depot, its content is not immediately synchronized. To synchronize the content of the UMDS depots, you must or wait for the scheduled synchronization to take place.

To add a UMDS depot, . The UMDS specification must contain the URI location to the `index.xml` file of the depot. Optionally, you can set a description and indicate whether the depot must be enabled. By enabling the UMDS depot, you instruct Depot Manager to synchronize only the content that is available on that depot.

You can always edit the initial UMDS depot settings, by

To retrieve information about the currently configured UMDS depot, use the . You can remove a currently configured UMDS depot and all its downloaded content from Depot Manager by

Synchronizing Software Depots

The VMware online depot, the vendor online depots, and the UMDS depot must be synchronized regularly if you want to have the most recent software updates delivered by VMware, partners, and other third-party vendors. Use the vSphere Automation APIs to create a synchronization schedule or to synchronize the added depot immediately.

The Depot Manager does not synchronize immediately the metadata of the newly added online and UMDS depots. If you want to force the synchronization and not wait for the scheduled synchronization to take place, . You can also define a custom schedule to sync the metadata from the currently configured online or UMDS depots.

To create a custom schedule for checking for new software updates, you must first define the schedule parameters by using the . Then you can add the schedule to the schedule specification by using the . Optionally, you can use the schedule specification to add an email to which notifications will be sent and define whether updates will be downloaded automatically. To apply the custom schedule, .

The default schedule is set to update the metadata daily at a random time. To reset the schedule to the default settings, .

Working with Offline Depots

An offline depot is a ZIP file that contains the metadata and payloads of software updates and follows the same structure as the online depot. Use the vSphere Automation APIs to import the content of an offline depot to the vCenter Server database.

To add an offline depot to the depots managed by the Depot Manager, you must create an offline depot specification by using the . When you define the offline depot parameters, . Depending on the location of the offline depot, when you create the offline depot specification, you must provide either the URI location or the file ID returned by the Jetty Web server embedded in the vSphere Lifecycle Manager. You set the type of the source from which the offline depot is downloaded by using the .

Pull Depot Content from a URI

To indicate that the offline depot resides on a URI location, . If you provide an HTTPS location to the offline depot, make sure you also provide a certificate trusted by the VMware Certificate Authority (VMCA) or a custom certificate from the VMware Endpoint Certificate Store (VECS). For detailed information about how to manage certificates, see the *vSphere Authentication* documentation.

Push Depot Content to the Depot Manager

To push the content of an offline depot to the Depot Manager, you must first upload the ZIP file to the Jetty Web server at the `https://<vcenter_FQDN>:9087/vum-fileupload` URL. The server returns a file identifier that you can.

Managing Depot Overrides

In case, you have a smaller Remote Office/ Branch Office (ROBO) cluster environment, or Edge computing environment, your clusters have no, or limited access to the Internet and limited connection to a vCenter Server instance. In such an environment, you can use Depot Manager to fetch the metadata and payloads of a desired software state from a local to the cluster depot.

To remediate a ROBO cluster, you must have access to a local software depot that hosts the components of the desired state. You can use Depot Manager to either export the whole vSphere Lifecycle Manager depot to an offline bundle, or export only the content of the desired state required for remediating the ROBO cluster.

To export the desired state image from the vSphere Lifecycle Manager depot, . To move the content of the offline bundle to the ROBO location, you must physically copy the ZIP file, unarchive, and mount its content to an HTTP server inside the ROBO environment.

To redirect a ROBO cluster to download software updates from a local repository and not from the vSphere Lifecycle Manager depot on the vCenter Server instance, the vSphere AutomationAPI offer the cluster `DepotOverrides` service. To add a depot override location to a ROBO cluster, . Specify the URI location of the local depot with the . During the ROBO cluster remediation, Depot Manager instructs the hosts in the ROBO cluster to download the software updates from the configured local depots within the ROBO cluster.

Inspecting Depot Contents

You can use the vSphere AutomationAPI to inspect the contents of the already synchronized and imported depots. You can list the available base images, add-ons, and components, or retrieve some detailed information about a specific software update.

To retrieve a list of the base images available on a vCenter Server instance,

To retrieve a list of all currently available add-ons in the vSphere Lifecycle Manager depot,

To retrieve a list of all components currently available in the vSphere Lifecycle Manager depot, you can

Enabling a Cluster to Use a Software Specification

If you want to use the vSphere Lifecycle Manager to manage the life cycle of clusters in your environment, you have two options for enabling this feature. You can enable the vSphere Lifecycle Manager when you create the cluster. You can also turn an already created cluster into one managed by the vSphere Lifecycle Manager.

Creating a Cluster with Enabled vSphere Lifecycle Manager

You can use only the vSphere Web Services API or the Virtual Infrastructure Management API to create a cluster in your virtual environment and specify the initial desired state of the cluster. Use the vSphere AutomationAPI to specify the detailed desired state after the cluster creation.

Starting with vSphere 8.0 Update 1, you can use the Virtual Infrastructure Management API which introduced a new way of accessing and using the vSphere Web Services. That is, you can now use the JSON data format over HTTP and create REST-like requests to the vSphere Web Services. Use the Virtual Infrastructure API or the vSphere Web Services API to create a cluster. For more information about how to authenticate and use the Virtual Infrastructure Management API, see the *vSphere Web Services SDK Programming Guide*, *An Introduction to vSphere APIs*, and *Getting*

Started with vSphere APIs documentation. You can also check the *Virtual Infrastructure JSON API* reference documentation at https://developer.vmware.com/apis/vi-json/latest/.

You can use the `POST https://<vcenter_ip_address_or_fqdn> /sdk/vim25/<release_number>/Folder/<moID>/CreateClusterEx` request and submit the name of the cluster and a `ClusterConfigSpecEx` data object in the request body. In the data object, among other properties, you can specify the desired state for the cluster. The `desiredSoftwareSpec` property in the `ComputeResourceConfigSpec` data object contains the desired software specification for the cluster. This property is available for applications using the vSphere Web Services API of version 7.0 and later. You can create a `DesiredSoftwareSpec` data object and specify the base image that must be applied on the cluster with the `baseImageSpec` property. Optionally, you can specify a vendor add-on to be added to the software specification with the `vendorAddOnSpec` property.

Starting with vSphere 8.0, you can also enable the cluster to use vSphere Configuration Profiles by setting the `enableConfigManager` property of the `ComputeResourceConfigSpec` data object to `true`.

Enabling an Existing Cluster to Use vSphere Lifecycle Manager

If you want to manage the life cycle of a cluster by using a single software specification, you must first enable vSphere Lifecycle Manager on that cluster. You can use the vSphere AutomationAPI to enable a cluster to use the vSphere Lifecycle Manager feature.

Before you enable vSphere Lifecycle Manager on a cluster, you can check whether the cluster meets all prerequisites. vSphere Lifecycle Manager can be enabled for a cluster only if the following requirements are met:

- All hosts in the cluster are of version 7.0 or later.
- All hosts in the cluster are stateful.
- All hosts in the cluster include only components that belong to integrated solutions, such as VMware vSAN™ and VMware vSphere® High Availability.
- None of the hosts in the cluster are in the process of active remediation through the VMware vSphere®Update Manager™.
- The cluster has a desired state already created for it.

If you want to run a preliminary check about whether all hosts in the cluster meet these requirements, .

The cluster ID represents the unique identifier for a cluster resource.

to specify which checks can be skipped during the cluster preliminary check. Though you can skip some checks with this operation, the Image Manager runs all checks during the enablement operation. If you leave the check specification empty, all checks are run for each host in the cluster. You can select among the following checks to be skipped when running a pre-check operation:

- `SOFTWARE`. Checks whether there are any orphaned vSphere Installation Bundles (VIBs) and any software that cannot co-exist with vSphere Lifecycle Manager.
- `VERSION`. Checks whether all hosts in the cluster are of version greater than a predefined one.
- `STATELESSNESS`. Checks whether there are any stateless hosts in the cluster. vSphere Lifecycle Manager can be enabled only if the cluster does not contain stateless hosts.
- `VUM_REMEDIATION`. Checks whether any of the hosts in the cluster are currently remediated through the VMware vSphere®Update Manager™.
- `SOFTWARE_SPECIFICATION_EXISTENCE`. Checks whether there is a software specification already associated with this cluster. In case, this check reports that the cluster does not have a software specification, you must first create a draft software specification for this cluster and then commit the draft.
- `VSAN_WITNESS_ELIGIBILITY`. Checks whether the software specification can be used on any vSAN witness hosts in the cluster. For information about how you can manage a vSAN cluster by using vSphere Lifecycle Manager, see *vSAN Clusters and vSphere Lifecycle Manager* chapter in the *Managing Host and Cluster Lifecycle* documentation.

You can also get information about which clusters in your environment are managed with a single software specification.

Enabling a Standalone Host to Use a Software Specification

Starting with vSphere 8.0, you can use the vSphere Lifecycle Manager to manage the life cycle of standalone hosts in your vCenter Server system.

You can add a standalone host to a vCenter Server instance under a data center object or into a folder. See Hosts. If you remove a host from a cluster but leave it in the data center, the host is also considered a standalone host.

To manage a standalone host with a software specification, you must first enable the vSphere Lifecycle Manager on that host. You can run a preliminary check to establish whether the standalone host meets all requirements for enabling the vSphere Lifecycle Manager. . The pre-checks that you can skip before enabling the vSphere Lifecycle Manager on a standalone host are the same as for a cluster.

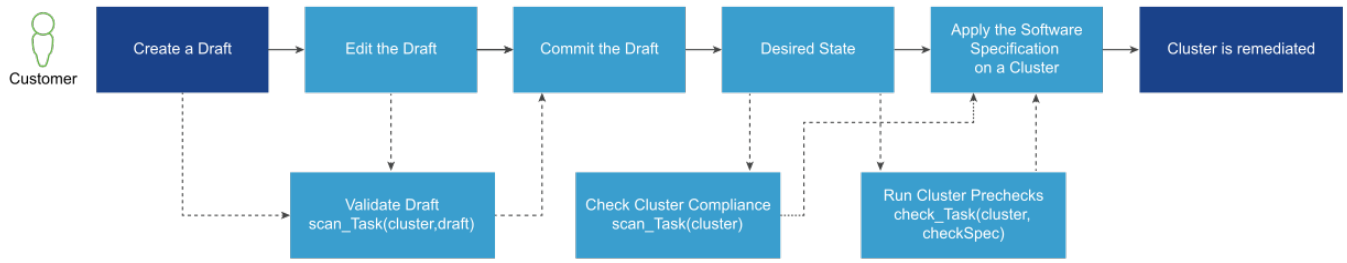
To enable vSphere Lifecycle Manager on a standalone host, . Currently, you can skip only the check for orphaned vSphere Installation Bundles (VIBs).

Working with Draft Software Specifications

You create a draft software specification to describe the components of the desired state that you want to apply on a cluster or a standalone host.

The draft software specification is the working copy of the desired software state. Only one user at a time is allowed to edit a single draft for a cluster or a standalone host. Before saving the changes to the edited draft version, you can validate the content of the draft.

Figure 93: Drafts Workflow for a Cluster



A typical workflow for working with draft software specifications starts with creating a draft software specification for a specific cluster or a standalone host. If the cluster or the standalone host already have a software specification defined, this action takes the latest committed draft and you can edit its contents according to your needs. If the cluster or the standalone host have no software specification created yet, this method creates an empty draft. The only mandatory item for a draft is a base image of a specific version.

After you complete adding components, an add-on, and a base image, you can save your changes by committing the draft. This operation results in setting the committed draft as the current desired state of the cluster or the standalone host. Before committing the draft software specification, you can validate the contents of the draft or check whether all hosts in the cluster or the single standalone host are compliant with the draft.

If the commit operation is successful, the draft becomes the desired state for the cluster or the standalone host. You can now export the software specification and use it, for example, in a ROBO cluster scenario or another standalone host. You can also validate the compliance of the hosts in the cluster or the standalone host against the desired state and then apply the software specification, if feasible.

Creating a Draft Software Specification

To describe the components of a desired state for a cluster or a standalone host, create a draft software specification and save the desired state when ready.

To edit an existing desired state or to create an empty draft software specification:
- For a cluster, .
- For a standalone host, .

As a result, you receive a draft ID which you can use to add a base image, an add-on, or some components to the draft.

Editing a Draft Software Specification

After you have created a draft software specification, use the vSphere AutomationAPI to edit its items.

To set a base image to a draft software specification:
- For a cluster, .
- For a standalone host, .

If the draft contains a base image, this method overwrites the existing image. The base image specification contains the version of the bootable ESXi that must be included in the desired state. To retrieve details about the base image that is currently present in a draft, .

To add an OEM add-on to a draft software specification:
- For a cluster, .
- For a standalone host, .

Pass as arguments to these methods the cluster or the standalone host ID, the draft ID, and the add-on specification. If you want to remove an add-on from a draft, .

You can add a component, change the version, or delete an existing component from a draft software specification. To change the version of a component included in a draft:
- For a cluster, .
- For a standalone host, .

As a result, you add the component specified with the `component_ID` and `version` arguments to the draft, if it is missing. To remove a component from a draft, . You can change multiple components in a draft by .

To retrieve information about all components present in a draft, or a single component, .

Validating the Draft Software Specification

Before saving a draft and turning it into a desired state for a cluster or a standalone host, you can check whether the specification is complete and valid. You can also check whether the draft specification drifts in any way from the current state of the cluster or the host.

The `Drafts` service offers two methods for validating the draft software specification. To check whether a draft is complete and there are no conflicts between the draft components, or unresolved dependencies, . You validate whether there were any other drafts committed for this cluster or standalone host, which can make the current commit operation invalid. The method also validates whether all components defined in the software specification are available in the depot metadata. This method does not run compliance checks against the cluster or the standalone host.

To check whether all hosts in the cluster or the standalone host are compliant with the draft software specification, . This method results in running a comparison between the draft specification and the current state of each host in the cluster or the current state of the standalone host.

Committing the Draft Software Specification

When you commit a draft software specification, it becomes the desired state for the cluster or the standalone host.

To save the draft that you created for a cluster or a standalone host, . The Image Manager component runs a validation check before the draft gets saved to the database. This method returns an identifier of the commit operation. You can use the ID to retrieve information about a specific commit such as the author of the commit operation, the time when the draft was committed, and so on.

Working with Desired Software States

When you commit a draft software specification, you make the committed draft the desired state for that cluster or standalone host. If all hosts in the cluster are compliant with the desired state, you can remediate the cluster. If the single standalone host is compliant with the desired state, you can update the ESXi host to that state.

You can use the methods provided with to manage a desired state for a cluster or a standalone host. Before you apply a desired state on a cluster or a standalone host, you can run pre-checks to ensure that all hosts in the cluster or the single host are in a good state to be remediated. The pre-checks verify whether any of the hosts in the cluster or the single standalone host must be rebooted or are in maintenance mode. You can also check the compliance of the cluster or the standalone host against the desired state. See Checking the Compliance Against the Desired State.

You can export a software specification created for a cluster or a standalone host by using one of the following formats:
- An offline bundle in a ZIP file format.
- An ISO image.
- A JSON file.

Use the vSphere Lifecycle Manager API to import a software specification as a draft and then edit it. You have several options for running the import operation depending on the location and format of the desired software state.

Exporting and Importing a Desired State

Use the vSphere AutomationAPI to export the desired software state of a cluster or a standalone host. Then you can import the desired state to a different cluster or host in the same or a different vCenter Server instance.

Exporting a Desired State

To export a desired state, you can use one of the following methods:
- For a cluster, .
- For a standalone host, .

This method does not export any information about the solutions available on the cluster or the standalone host since the constraints set by these solutions might not be applicable for another cluster or host. .

You can choose how to export the desired software specification.
- Export as an ISO image. Use the exported ISO image for performing clean installs and for bootstrapping purposes. You can upload the ISO file into the Jetty Web server on the target vCenter Server instance but you cannot use ISO files to manage the life cycle of clusters or a standalone host through the vSphere Lifecycle Manager feature.
- Export as an offline bundle in a ZIP file format. You can use the exported offline bundle to create a depot and add its components to the resources managed by the Depot Manager module.
- Export as a JSON file holding the desired state specification. You can then reuse the JSON file to apply the desired state that it contains to another cluster or standalone host in the same or in a different vCenter Server instance. Note that the JSON file holds only the description of the desired state. You must check whether all components described in the JSON file are available in the depot for the target cluster. See Importing a Desired State Specification for information about how you can use a desired state specification for another cluster or standalone host.

Importing a Desired State Specification

To import a desired state of a cluster or a standalone host and assign it to another cluster or standalone host in the same or different vCenter Server instance:
- For a cluster, .
- For a standalone host, .

Use the instance of the import specification to describe the download source and the source type of the imported software specification. Depending on the location and the source type of the exported desired state, you can choose from the following import options:

- Import a file from the vCenter Server or your local file system. You can also use this option to import a specification file that resides on your local file system. Make sure you set the source type of the import specification to `PUSH` through the .
- Import a file that resides on a URI location. .
- Import a desired state as a JSON string.

Checking the Compliance Against the Desired State

Before applying a desired state on a cluster, you can scan all hosts in the cluster against the desired state and check the cluster compliance against the desired state. Before applying a desired state on a standalone host, you can scan the host and check its compliance against the desired state.

To check the compliance of all hosts in a cluster or of the single standalone host:
- For a cluster, .
- For a standalone host, .

Pass as an parameter the cluster or the host identifier. The method compares the desired state against the current state of each host in the cluster or of the single host and as a result calculates the compliance.

You can retrieve the cluster or the standalone host compliance status by :
- The overall cluster compliance status regarding the target version of the components described within the cluster desired state.
- The impact of applying the desired state on the cluster in case the cluster is non-compliant.
- A list of all compliant hosts in the cluster.
- A list of the incompatible hosts in the cluster.
- A list of the non-compliant hosts in the cluster.
- A list of the unavailable hosts which cannot be checked for compliance against the desired state.
- The compliance status of each host in the cluster.
- The notifications returned by the compliance check operation.
- The time that the compliance check takes.
- The ID of the committed draft for which the compliance check is performed. , the compliance check is run against a draft software specification.
- The compliance overall stage status of the cluster or the standalone host and is relevant when the compliance status of the cluster or the standalone host is `NON_COMPLIANT`.
- The compliance of the OEM add-on on the standalone host with respect to the add-on in the desired state.
- The compliance of the base image on the standalone host with respect to the base image in the desired state.
- The compliance of all effective components and all components present on the standalone host in respect to the components in the desired state.
- The compliance of all DPU devices on the standalone host with respect to the desired state.
- The compliance of the hardware support on the standalone host with respect to the hardware support defined in the target state.

A cluster or a standalone host can have one of the following compliance statuses regarding the target versions:
- `COMPLIANT`. The target versions of the components described in the desired state of the cluster or the standalone host are the same as the versions of the components currently present on the hosts in the cluster or on the standalone host.
- `NON_COMPLIANT`. The desired state of the cluster or the standalone host describes components with higher versions than the versions of the components currently present on the hosts in the cluster or on the standalone host. Non-compliant clusters or hosts are those clusters or hosts which have orphaned VIBs, or components on the hosts that are not present in the desired state specification.
- `INCOMPATIBLE`. One or more hosts in the cluster or in the standalone host have components with higher versions than the components described in the desired state specification.
- `UNAVAILABLE`. The current state of one or more hosts in the cluster or of the standalone host cannot be retrieved and as a result the compliance check cannot be performed.

You can check the compliance impact of applying the desired state on a non-compliant cluster or a standalone host by . Use it to retrieve information about the steps you must take to remediate the cluster or the standalone host successfully. You might need to reboot a host or put a host into maintenance mode to remediate the cluster or the standalone host successfully.

Hardware Compatibility Data

The hardware compatibility data contains information about the compatibility between ESXi hosts and ESXi versions.

You can use the service to retrieve information about the compatibility data or to update the local compatibility data on a vCenter Server instance. To retrieve information about the compatibility data stored on your vCenter Server instance, . To update the local compatibility data with the latest version available from the official VMware source, .

Checking the Hardware Compatibility of an ESXi Host

You can query the hardware compatibility for a host before upgrading to a new ESXi version. You can also download the information generated by the hardware compatibility report.

To use services form the package, you must verify that you have accepted to participate in the CEIP and there is available compatibility data.

You can use the to list available releases for generating a compatibility report for a specific ESXi host. To list the locally available ESXi releases for the host that can be used to generate a compatibility report,. The list includes only major and update releases. Patch releases are not listed.

You can use the to generate a hardware compatibility report for an ESXi host against a specific ESXi release. To return the last generated hardware compatibility report for a specific host, . To generate a hardware compatibility report for a specific host against specific ESXi release, .

You can use the to download information generated by the hardware compatibility report. To retrieve the URI location for downloading a compatibility report, .

Configuring Remediation Settings

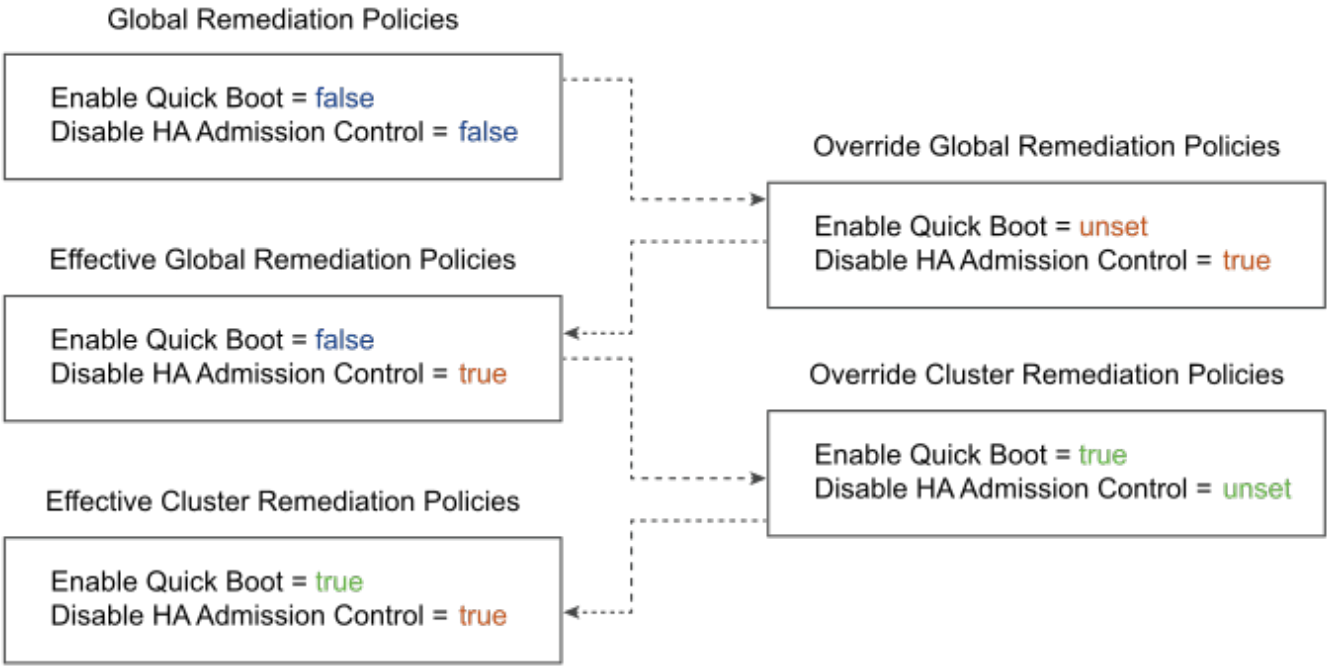
You can control the behavior of the ESXi hosts and virtual machines during the remediation process. You can create a global remediation policy that applies to all clusters and standalone hosts in a vCenter Server instance. You can also set a remediation policy to a specific cluster and a standalone host.

When you run cluster and standalone host compliance checks, the Coordinator module runs a series of checks on each host to determine their state and whether some additional actions must be taken to ensure the success of the remediation operation. In case one or more hosts in the cluster and any of the standalone hosts are evaluated as non-compliant, additional checks are run on those hosts to evaluate whether they must be rebooted or put into maintenance mode. Currently, VMware provides a set of behavior controls (remediation policies) regarding the virtual machines and the hosts in a cluster or a standalone host. This set of remediation policies might change with the next vSphere release.

How Remediation Policies Overrides Work

The vSphere Lifecycle Manager provides a default global policy configuration that must be applied on each cluster and standalone host during remediation. Through the vSphere AutomationAPI, you can change the global policies and create some cluster-and host-specific policies. Before remediating a cluster and standalone host, you can use the API to determine the effective global and cluster-and host-specific remediation policies. The following graphic describes how the mechanism of the policy overrides works.

Figure 94: How Remediation Policies Work



All clusters and standalone hosts in a vCenter Server instance inherit the default or the overridden global policy settings unless the global policy is explicitly overridden on a cluster and host level.

Editing Global or Cluster- and Host-Specific Remediation Policies

Remediation Policy Options for Clusters

To describe a global or cluster-specific remediation policy, use . For the vSphere 8.0 release, VMware provides the following methods to configure a global or cluster-specific policy.

	Description
	Deactivate the VMware Distributed Power Management (DPM) feature for all clusters or for a specific cluster. DPM monitors the resource consumption of the virtual machines in a cluster. If the total available resource capacity of the hosts in a cluster is exceeded, DPM powers off (or recommends powering off) one or more hosts after migrating their virtual machines. When resources are considered underutilized and capacity is needed, DPM powers on (or recommends powering on) hosts. Virtual machines are migrated back to these hosts. During the cluster remediation, the vSphere Lifecycle Manager cannot wake up and remediate hosts that are automatically put into a stand-by mode by DPM. These hosts stay non-compliant when DPM turns them on. The vSphere Distributed Resource Scheduler (DRS) is unable to migrate virtual machines to the hosts which are not remediated with the desired state for the cluster.

Table continued on next page

Continued from previous page

	Description
	To deactivate DPM during the cluster remediation, . By default, the vSphere Lifecycle Manager temporarily deactivates DPM and turns on the hosts to complete the remediation. DPM is enabled again when the cluster remediation finishes.
	Deactivate the vSphere HA admission control. vSphere HA uses admission control to ensure that a cluster has sufficient resources to guarantee the virtual machines recovery when a host fails. If vSphere HA admission control is enabled during remediation, putting a cluster into maintenance mode fails because vMotion cannot migrate virtual machines within the cluster for capacity reasons. To allow the vSphere Lifecycle Manager to temporary deactivate vSphere HA admission control, . By default, the vSphere HA admission control is enabled because DRS should be able to detect issues with the admission control and deactivate it to allow the remediation to complete.
	Migrate the suspended and powered off virtual machines from the hosts that must enter maintenance mode to other hosts in the cluster. To enable this remediation policy, . By default, this setting is deactivated in the global remediation policy.
	Specify what actions vSphere Lifecycle Manager must take if a host fails to enter maintenance mode during the remediation. To configure this policy on a global or cluster-specific level, . You can set the number of times that vSphere Lifecycle Manager tries to put a host into maintenance mode and the delay between the tries. When the threshold is reached and the host failed to enter maintenance mode, the cluster remediation fails. By default, vSphere Lifecycle Manager tries to put a host into maintenance mode three times with a five minute delay between each try before the cluster remediation fails.
	Prevents the remediation of vSAN clusters if vSphere Lifecycle Manager reports hardware compatibility issues during the hardware compatibility check performed as part of the remediation pre-check or the remediation tasks of the cluster., detected hardware issues are reported as warnings and do not prevent the remediation of the vSAN cluster.
	Enable simultaneous remediation of all hosts that are in maintenance mode with in the cluster. to indicate the maximum number of hosts that can be remediated in parallel.

Table continued on next page

Continued from previous page

	Description
	NOTE If the hosts have NSX virtual distributed switches that are ready to be migrated to vSphere Distributed Switches, you must manually set the maximum number of parallel remediations to no more than 4. In cases when host switch migration is needed, if more than 4 hosts are remediated in parallel, the remediation might fail, because the host switch migration takes more time than the time vSphere Lifecycle Manager needs to complete the parallel remediation.
	Specify how the power state of the virtual machines must change before the host enters maintenance mode. If DRS is not enabled on a cluster or the automation level of a DRS cluster is not set to fully automated, the Coordinator module fails to remediate the cluster if the remediation requires a reboot or maintenance mode. You can set a policy that powers off or suspends the virtual machines on hosts that must be rebooted or must enter maintenance mode during remediation. The DRS takes care of changing the power state of the virtual machines when the host enters and exits maintenance mode. To set a policy for the power state of the virtual machines during the remediation, one of the following values: • `DO_NOT_CHANGE_VMS_POWER_STATE`. Indicates that the power state of the virtual machines must not be changes. • `POWER_OFF_VMS`. Indicates that the virtual machines must be powered off before the hosts enter maintenance mode. • `SUSPEND_VMS`. Indicates that the virtual machines must be suspended before the hosts enter maintenance mode. • `SUSPEND_VMS_TO_MEMORY`. Indicates that the virtual machines must be suspended before the hosts enter maintenance mode. By default, the Coordinator must leave the power state of the virtual machines unchanged.
	Reduce the reboot time of an ESXi host by skipping all the hardware initialization processes and restarting only the hypervisor. This policy is applicable only if the host platform supports the Quick Boot feature. To enable the Quick Boot feature on the hosts during the remediation, . By default, this policy is deactivated.

Remediation Policy Options for Standalone Hosts

To describe a global or cluster-specific remediation policy, use the . For the vSphere 8.0 release, VMware provides the following methods to configure a global or host-specific policy.

	Description
	Optimize the host patching and upgrade operations by reducing the reboot time of an ESXi host. Since the patching and upgrading operations do not affect the hardware of the host, the hardware initialization processes can be skipped. This policy is applicable only if the host platform supports the Quick Boot feature. For more information about which hosts are Quick Boot compatible, see the following KB article https://kb.vmware.com/s/article/52477.
	Specify how the power state of the virtual machines must change before the standalone host enters maintenance mode. You can choose between the following power actions: • `DO_NOT_CHANGE_VMS_POWER_STATE`. Indicates that the power state of the virtual machines must not be changed. • `POWER_OFF_VMS`. Indicates that the virtual machines must be powered off before the standalone host enters maintenance mode. • `SUSPEND_VMS`. Indicates that the virtual machines must be suspended before the standalone host enters maintenance mode. • `SUSPEND_VMS_TO_MEMORY`. Indicates that the virtual machines must be suspended before the standalone host enters maintenance mode.
	Specify what actions vSphere Lifecycle Manager must take if a standalone host fails to enter maintenance mode during the remediation. You can set the number of times that vSphere Lifecycle Manager tries to put a standalone host into maintenance mode and the delay between the tries. When the threshold is reached and the standalone host failed to enter maintenance mode, the host remediation fails.

Remediating an ESXi Cluster and a Standalone Host

You can use the vSphere Automation API to initiate the remediation of a cluster or a standalone host through the vSphere Lifecycle Manager.

To remediate a cluster with a desired state, . You can also set the minimum commit ID of the draft software specification that must be used for remediating the cluster. Upon successful completion of the remediation task, all hosts in the cluster have the same software state.

To remediate a standalone host with a desired state, . Use the software apply specification to set the minimum commit identifier of the draft software specification. See Committing the Draft Software Specification. You can also specify whether the VMware End User License Agreement (EULA) must be accepted.

Integrate Third-Party Solutions with vSphere Lifecycle Manager

A solution is an ESXi software package that extends the functionality and capabilities of a host and integrates with the vCenter Server system. To be able to manage the life cycle of your third-party solutions on a cluster managed with images, you must integrate the solutions with vSphere Lifecycle Manager.

Examples of VMware integrated solutions are VMware NSX®, vSAN, and vSphere Supervisor. For more information about the VMware integrated solutions, see *vSphere Lifecycle Manager Images and Other VMware Products and Solutions* section in the *Managing Host and Cluster Lifecycle* documentation.

Third-party software providers can use the vSphere APIs, VMware vSphere APIs for I/O Filtering (VAIO), VMware Daemon Software Development Kit, and others to develop third-party solutions for their vSphere platforms.

You can use the vSphere Lifecycle Manager automation APIs to manage the life cycle of third-party solutions on a cluster managed with a single image. First you need to package and upload your solution components to the vSphere Lifecycle Manager depot. Use the depot to store and manage the software updates for your third-party solutions. To make a solution available on a cluster, create a software specification that contains the solution and remediate all hosts in the cluster with that image. vSphere Lifecycle Manager manages the life cycle of the solution components by consuming the software updates from the vSphere Lifecycle Manager depot.

> **NOTE**
> If you export an image from the cluster where your third-party solution is running, the solution components are not part of the exported image.

Prerequisites

To enable vSphere Lifecycle Manager to manage your third-party solutions, you must use the ESXi Packaging Kit (EPK) to create installable packages. As of the vSphere 7.0 release, partner development kits generate components as installation packages. For more information about how to use the EPK to create components, assemble components into add-ons, then merge the add-ons with the base image to author a depot, see the *ESXi Packaging Kit (EPK) Development Guide* documentation.

Integrate a Third-Party Solution to Work with vSphere Lifecycle Manager

1. Create an online or offline depot to host your third-party solutions. See Working with Online Depots and Working with Offline Depots.
   - The following example creates an online depot that can be accessed through the `http://my_online_depot.com` URL.

     ```
     POST https://<vcenter_server_ip_or_fqdn>/api/esx/settings/depots/online
     ```

     ```
     {
        "description" : "My online depot adds the My Solution component to the ESXi 7.0U3d base image",
        "ownerdata" : "ACME Company",
        "location" : "http://my_online_depot.com",
        "enabled" : true
     }
     ```

   - The following example creates an offline depot, also called an offline bundle, that can be imported to the vSphere Lifecycle Manager depot.

     ```
     POST https://{server}/api/esx/settings/depots/offline?vmw-task=true
     ```

     ```
     {
        "file_id" : "string",
        "description" : "string",
     ```

```
        "ownerdata" : "string",

        "source_type" : "PULL",

        "location" : "http://myurl.com"

    }
```

2. Synchronize the online depot to download the depot metadata and make the vSphere Lifecycle Manager aware of your solutions. See Synchronizing Software Depots.
   `POST https://<vcenter_server_ip_or_fqdn>/api/esx/settings/depots/online?vmw-task=true&action=sync`
3. Create a draft software specification. See Creating a Draft Software Specification.
   `POST https://<vcenter_server_ip_or_fqdn>/api/esx/settings/clusters/<cluster_id>/software/drafts`
4. Add your solution to the created draft software specification. See Editing a Draft Software Specification.
   `PUT https://<vcenter_server_ip_or_fqdn>/api/esx/settings/clusters/<cluster_id>/software/drafts/<draft_id>/software/components/<component_id>`
5. Save the created draft software specification to make it the desired state for the cluster. See Committing the Draft Software Specification.
   `POST https://<vcenter_server_ip_or_fqdn>/api/esx/settings/clusters/<cluster_id>/software/drafts/<draft_id>?action=commit&vmw-task=true`
6. Remediate the ESXi cluster with the desired state that contains your solution to apply the desired state on all hosts in that cluster. See Remediating an Cluster and a Standalone Host.
   `POST https://<vcenter_server_ip_or_fqdn>/api/esx/settings/clusters/<cluster_id>/software?action=apply&vmw-task=true`

```
{

    "hosts" : [

        "obj-103",

        "obj-103"

    ],

    "commit" : "obj-103",

    "accept_eula" : true

}
```

Results

You set up an image for the cluster which includes your third-party solution. You now manage all hosts in the cluster collectively with a single image. Upon remediation, the image is installed on all hosts in the cluster.

What to do next

You can update, delete or add new solutions to the draft software specification and then commit the changes to make the draft the desired state for the cluster.

Enable vSphere Lifecycle Manager on a Cluster Managed with Baselines

For more information about how to convert a cluster to use vSphere Lifecycle Manager images instead of baselines, see Enabling an Existing Cluster to Use .

vSphere Lifecycle Manager APIs Equivalent to the vSphere Host Patch Manager APIs

To manage the life cycle of a single host with baselines, you use the vSphere Host Patch Manager APIs which are part of the vSphere Web Services API. To manage the life cycle of all hosts in a cluster collectively with an image, you use the vSphere Lifecycle Manager APIs which are part of the vSphere Automation APIs. The following table compares the APIs for life cycle management of the hosts and clusters in your environment.

Operation	vSphere Lifecycle ManagerAPI	vSphere Host Patch Manager API
Check whether hosts can be remediated	`com.vmware.esx.settings.clusters.Software.check`	`vim.host.PatchManager.CheckHostPatch_Task`
Check the compliance of the cluster or host with the desired state	`com.vmware.esx.settings.clusters.Software.scan`	`vim.host.PatchManager.ScanHostPatchV2_Task`
Remediate the cluster or host with the desired state	`com.vmware.esx.settings.clusters.Software.apply`	`vim.host.PatchManager.InstallHostPatchV2_Task`
Retrieve information about the cluster or host that you want to remediate	• `com.vmware.esx.settings.clusters.Software.get` - View information about the current desired state. • `com.vmware.esx.settings.clusters.software.Components.list` - View all components in the current desired state. • `com.vmware.esx.settings.clusters.software.Components.get` - View detailed information about a specific component.	`vim.host.PatchManager.QueryHostPatch_Task` - View information about the bulletins installed on an ESXi host.
Stage a desired state on a cluster or host	`com.vmware.esx.settings.clusters.Software.stage`	`vim.host.PatchManager.StageHostPatch_Task`
Uninstall a component	`com.vmware.esx.settings.clusters.Software.apply` - When a software component is removed from the desired software specification and then the cluster is remediated, the component will be uninstalled from all hosts in that cluster.	`vim.host.PatchManager.UninstallHostPatch_Task`

Virtual Machine Configuration and Management

A virtual machine is a software computer that, like a physical computer, runs an operating system and applications.

The virtual machine consists of a set of specification and configuration files and is backed by the physical resources of a host. Each virtual machine encapsulates a complete computing environment and runs independently of the underlying hardware.

Starting with vSphere 6.5, you can create virtual machines, configure virtual machine settings, and perform power operations on the virtual machines through the vSphere Automation APIs.

Starting with vSphere 7.0, you can use the vSphere Automation APIs to perform various virtual machine management operations. For example, you can deploy virtual machines by using several approaches, clone an existing virtual machine, create an instant clone of a running virtual machine. You can also install VMware Tools which enables you to manage the life cycle and customize the networking and identity settings of the guest operating system installed on the virtual machine.

Creating Virtual Machines

You can use the vSphere Automation APIs to create virtual machines depending on your needs and infrastructure setup.

You can create a basic virtual machine and then configure it according to your needs. You can also create a more comprehensive virtual machine and then edit its settings. To create a virtual machine, you must specify the datastore, resource pool, folder, or host where the virtual machine is placed. Later, you can customize the virtual machine by specifying the boot options, number of CPUs, the guest OS, and virtual NIC. See Creating a Virtual Machine Without a Clone or Template and Configuring Virtual Machines.

You can use a turned off virtual machine to create a VM template from which to deploy other virtual machines. See Create a VM Template in a Content Library from a Virtual Machine. You can also mark a virtual machine as template by calling the `VirtualMachine.MarkAsTemplate` method from the vSphere Web Services APIs. See *`vSphere Web Services SDK Programming Guide`*.

You can capture a virtual machine or a vApp in an OVF template and store the template in a content library. Then you can use the OVF template to deploy a virtual machine or a vApp in your environment. See Creating Virtual Machines and vApps from Templates in a Content Library.

Creating a Virtual Machine Without a Clone or Template

You can create a virtual machine by using the REST API.

When you create a virtual machine without a template or clone, you can configure the virtual hardware, including processors, hard disc, memory. To create a virtual machine, you must specify the virtual machine attributes by using the `VMCreateSpec` data structure. For example, you can specify a name, boot options, networking, and memory for the new virtual machine.

All attributes are optional except the virtual machine placement information that you must provide by using the `VMPlacementSpec` data structure. Use the virtual machine placement specification to set the datastore, cluster, folder, host, or resource pool of the created virtual machine. You must make sure that all these vSphere objects are located in the same data center in a vCenter Server instance.

For more information, refer to the *`API Reference`* documentation.

Configuring Virtual Machines

You configure a virtual machine in the process of creation.

You can later view and edit virtual machine settings by adding or changing the type of the storage controllers, configure the virtual disks, boot options, CPU and memory information, or networks. Virtual machine settings can be configured when cloning, registering and relocating an existing virtual machine.

Name and Location

You specify the display name and the location of the virtual machine by using the `VMCreateSpec` and `VMPlacementSpec` data structures.

When you create your virtual machine, use the `name` property of the `VMCreateSpec` data structure to set the display name of the virtual machine.

You must create also a `VMPlacementSpec` instance that describes the location of the virtual machine in regards to the resources of a given vCenter Server instance. Use the `placement` property of the `VMCreateSpec` data structure to set the placement information for the virtual machine. You can set one or all of the following vSphere resources: datastore, cluster, folder, host, and resource pool.

Hardware Version

The hardware version of a virtual machine reflects the virtual hardware features supported by a virtual machine. These features depend on the physical hardware available on the ESXi host on which the virtual machine is running.

Virtual hardware features include the BIOS and Extensible Firmware Interface (EFI), the maximum number of CPUs, the maximum memory configuration, and other hardware characteristics.

When you create a virtual machine, the default hardware version of the virtual machine is the most recent version available on the host where the virtual machine is created. For information about the latest VMware products and virtual hardware versions, see Virtual machine hardware versions (1003746).

To set a hardware version different than the default, use the `hardware_version` enumerated type of the `VMCreateSpec` data structure. For information about the hardware features available for the virtual hardware versions, see Hardware features available with virtual machine compatibility settings (2051652).

You can set a lower virtual hardware version of a virtual machine than the highest supported by the ESXi host on which the virtual machine is running. Setting a lower hardware version can provide flexibility and is useful in the following cases.

- To help you standardize testing and deployment in your environment.
- In case you do not need the hardware features of the latest hardware version of the host.
- To maintain compatibility with hosts with a lower hardware version.

Boot Options

You can configure the boot options of a virtual machine by using the `VmHardwareBootCreateSpec` data structure.

By specifying the properties of the `VmHardwareBootCreateSpec` data structure, you can select one of the following settings when booting the virtual machine.

- Delay - Indicates a delay in milliseconds before starting the firmware boot process when the virtual machine is powered on.
- Retry - Indicates whether the virtual machine automatically retries to boot after a failure.
- Retry delay - Indicates a delay in milliseconds before retrying the boot process after a failure.
- Enter setup mode - If set to `true`, indicates that the firmware boot process automatically enters BIOS setup mode the next time the virtual machine boots. The virtual machine resets this flag to `false` once it enters setup mode.
- EFI legacy boot - If set to `true`, indicates that the EFI legacy boot mode is used.

Guest Operating System

The guest operating system that you specify affects the supported devices and available number of virtual CPUs.

When you create a virtual machine, you specify the guest operating system by using the `VmGuestOS` enumerated type of the `VMCreateSpec` data structure. The `VmGuestOS` enumerated type defines the valid guest OS types that you can choose from for configuring a virtual machine.

After the create operation finishes successfully, you can install the guest operating system on the mew virtual machine in the same way as you install it on a physical machine. For further information on installing a guest operating system, refer to the *Guest Operating System Installation Guide* at http://partnerweb.vmware.com/GOSIG/home.html and the *vSphere Virtual Machine Administration* guide.

Starting with vSphere 7.0, you can use the vSphere Automation APIs to install the VMware Tool on the quest operating system and perform some guest OS customizations. See Installing VMware Tools.

CPU and Memory

The `VMCreateSpec` data structure allows you to specify the CPU and memory configuration of a virtual machine.

To change the CPU and memory configuration settings, use the `VmHardwareCpuUpdateSpec` and `VmHardwareMemoryUpdateSpec` data structures.

You can set the number of CPU cores in the virtual machine by using the `count` property of the `VmHardwareCpuUpdateSpec` data structure. The supported range of CPU cores depends on the guest operating system and virtual hardware version of the virtual machine. If you set the `hot_add_enabled` and `hot_remove_enabled` properties to `true`, you allow virtual processors to be added or removed from the virtual machine at runtime.

You can set the memory size of a virtual machine by using the `size_MiB` property of the `VmHardwareMemoryUpdateSpec` data structure. The supported range of memory sizes depends on the configured guest operating system and virtual hardware version of the virtual machine. If you set the `hot_add_enabled` property to `true` while the virtual machine is not powered on, you enable adding memory while the virtual machine is running.

Networks

You configure network settings so that a virtual machine can communicate with the host and with other virtual machines. When you configure a virtual machine, you can add network adapters (NICs) and specify the adapter type.

You can add virtual Ethernet adapters to a virtual machine by using the `VmHardwareEthernetCreateSpec` data structure. The `VmHardwareEthernetEmulationType` enumerated type defines the valid emulation types for a virtual Ethernet adapter. The `VmHardwareEthernetMacAddressType` enumerated type defines the valid MAC address origins for a virtual Ethernet adapter. You can set the MAC address type to `MANUAL`, `GENERATED`, or `ASSIGNED`. To specify the MAC address explicitly, select `MANUAL`.

You can specify also the physical resources that back a virtual Ethernet adapter by using the `VmHardwareEthernetBackingSpec` data structure. The `VmHardwareEthernetBackingType` enumerated type defines the valid backing types for a virtual Ethernet adapter. You can set the backing type to `STANDARD_PORTGROUP`, `HOST_DEVICE`, `DISTRIBUTED_PORTGROUP`, or `OPAQUE_NETWORK`.

Managing Virtual Machines

Virtual machines can be configured like physical computers.

You can change the guest operating system settings after installing VMware Tools. You can add and remove virtual machines from the vCenter Server inventory. You can also move virtual machines from one host or storage location to another.

Filtering Virtual Machines

You can retrieve commonly used information about virtual machines that match specific criteria. You can retrieve information for up to 4000 virtual machines in a single vCenter Server instance.

You can retrieve a list of virtual machines in a single vCenter Server instance by filtering the results based on a specific requirement. For example, you can use as filter criteria the power state of the virtual machines, or the host, cluster, data center, folder, or resource pool that must contain the virtual machines. In case you specify multiple filter criteria, only virtual machines that match all filter criteria are returned.

Installing VMware Tools

VMware Tools is a set of drivers and utilities that you install on the guest operating system of a virtual machine to enhance the performance of the guest OS. VMware Tools also improves the management of the virtual machine. For each guest OS, VMware provides a specific binary-compatible version of VMware Tools.

Before you install VMware tools, you must install and boot the guest operating system.

To mount and unmount the VMware Tools installer CD as a CD-ROM for the guest operating system, use the connect and disconnect operations of the `tools/installer` service.

To mount the VMware Tools, use the `POST https://<api_host>/api/vcenter/vm/<vm>/tools/installer?action=connect` HTTP request. On Windows guest operating systems with activated Autorun feature, this request automatically initiates the installation of the VMware Tools and requires a user input to complete. On other guest operating

systems, this method only mounts the VMware Tools disk image on the virtual CD/DVD drive and the user is required to do some guest OS-specific actions. For example, for some Linux distributions, the user is required to extract the contents of the VMware Tools installation archive and run the installer.

To unmount the VMware Tools installer CD image from the virtual CD/DVD drive, use the `POST https://<api_host>/api/vcenter/vm/<vm>/tools/installer?action=disconnect` HTTP request.

To monitor the status of the VMware Tools installation, you can check the `Tools.Info.version-status` and `Tools.Info.run-state` from `Tools.get`.

To upgrade the VMware Tools, use the `POST https://<api_host>/api/vcenter/vm/<vm>/tools?action=upgrade` HTTP request. The request takes as arguments the ID of the virtual machine on which the VMware Tools is installed and running. Use the `command_line_options` string to specify the command-line options that you want to pass to the installer to modify the tools installation procedure. You can monitor the upgrade operation in the same way you monitor the installation operation.

To update the properties of the VMware Tools, use the `PATCH https://<api_host>/api/vcenter/vm/<vm>/tools` HTTP request. Pass as argument a `VmToolsUpdateSpec` data structure and define the tools upgrade policy settings for the virtual machine by using the `VmToolsUpgradePolicy` enumerated type.

Performing Virtual Machine Power Operations

You can start, stop, reboot, and suspend virtual machines by using the operations of the `Power` service.

A virtual machine can have one of the following power states:
- `POWERED_ON` - Indicates that the virtual machine is running. If a guest operating system is not currently installed, you can perform the guest OS installation in the same way as for a physical machine.
- `POWERED_OFF` - Indicates that the virtual machine is not running. You can still update the software on the physical disk of the virtual machine, which is impossible for physical machines.
- `SUSPENDED` - Indicates that the virtual machine is paused and can be resumed. This state is the same as when a physical machine is in standby or hibernate state.

To perform a power operation on a virtual machine, you can use one of the operations of the `Power` service. Before you use one of the operations to change the power state of a virtual machine, you must first check the current state of the virtual machine by using the `GET https://<api_host>/api/vcenter/vm/<vm>/power` HTTP request. Pass as argument the virtual machine identifier.

Following is a list if the power operations, which you can use with a `POST` HTTP request:
- `power?action=start` - Powers on a powered off or suspended virtual machine. The method takes as argument the virtual machine identifier.
- `power?action=stop` - Powers off a powered on or suspended virtual machine. The method takes as argument the virtual machine identifier.
- `power?action=suspend` - Pauses all virtual machine activity for a powered on virtual machine. The method takes as argument the virtual machine identifier.
- `power?action=reset` - Shuts down and restarts the guest operating system without powering off the virtual machine. Although this method functions as a `stop` operation that is followed by a `start` operation, the two operations are atomic with respect to other clients, meaning that other power operations cannot be performed until the `reset` operation completes.

Registering and Unregistering Virtual Machines

When you create a virtual machine, it becomes part of the vCenter Server inventory and is registered to the host and vCenter Server. If you remove a virtual machine from the vCenter Server inventory, it becomes unusable. Virtual machine files remain in the same datastore but you cannot power on the virtual machine when it is not registered in the inventory.

You can temporarily remove a virtual machine from vCenter Server by unregistering it. Virtual machine files are not deleted from the datastore. Though all high-level information about the virtual machine such as statistics, resource pool association, permissions, and alarms, is removed from the host and the vCenter Server instance. To remove a virtual machine from the inventory, use the `POST https://<api_host>/api/vcenter/vm/<vm>?action=unregister` HTTP request and specify the ID of the virtual machine.

To restore a virtual machine to the vCenter Server inventory, and make it usable again, use the `POST https://<api_host>/api/vcenter/vm/<vm>?action=register` HTTP request. You pass as argument a `VMRegisterSpec` data structure that contains information about the current location of the virtual machine files on the datastore. You can also define the location within the vCenter Server inventory, for example, the cluster, folder, or the host, where you want to register the virtual machine. After registration, the virtual machine takes its resources (CPU, memory, and so on) from the resource pool or host to which it is registered.

If you no longer need a virtual machine and you want to free up datastore space, you can permanently delete a virtual machine from the inventory. Use the `DELETE https://<api_host>/api/vcenter/vm/<vm>` HTTP request and specify the ID of the virtual machine. Upon a successful completion of the operation, the virtual machine files are removed from the datastore, including the configuration file and the virtual disk files.

Managing Data Sets

You can use data sets to share information between a virtual machine and its guest operating system.

Data sets provide a communication infrastructure that you can use to share information between the host and guest. This information can then be used by guest agents or applications to configure the guest itself or guest applications. The information is grouped into data sets, each of which contains key-value entries comprising the data. Each application that uses the service should have at least one unique data set in which to store its data to avoid conflict with other applications. Each data set has attributes that define its access control and interoperability configuration.

NOTE
You should not store sensitive data, such as passwords or private keys, in plain text.

To perform data set operations, the virtual machine must be running virtual hardware version 20 or later.

You should modify each data set by using the application that originally created that data set. If your application modifies a data set owned by another application, the other application might stop working.

For more information about data sets, see *vSphere Virtual Machine Administration* and the *VMware Guest SDK Programming Guide*.

Data Set Operations

You can use the vSphere Automation API to manipulate entries in data sets.

You can use data set operations to share information between a virtual machine and its guest operating system. The available operations include listing data sets, retrieving data set information, creating, modifying, and deleting a data set. You can also manipulate individual entries in a data set. The available operations include listing entry keys in a data set, retrieving the value of an entry in a data set, modifying and deleting an entry in a data set.

Table 51: User Operations

Operation	Description
List data sets	You can list all available data sets of a virtual machine.
Get data set	You can retrieve information about a specific data set.
Create data set	You can create a new data set.

Table continued on next page

Continued from previous page

Operation	Description
Update data set	You can modify the attributes of a data set.
Delete data set	You can delete a data set.
List data set entry keys	You can list all entry keys in a data set.
Get data set entry key value	You can retrieve the value of a specific entry in a data set.
Set data set entry key value	You can create or update an entry in a data set. If the new entry does not contain a key that matches an existing key, the operation creates a new entry. If the new entry contains a key that matches an existing key, the new key overwrites the value of the existing key.
Delete data set entry key value	You can delete an entry in a data set.

Using Content Libraries

Use the vSphere Automation API to access the Content Library service and use it to store and share different types of content in your vSphere environment. Content libraries are vSphere container objects for storing and sharing OVF and OVA packages, VM templates, vApp templates, and other types of files.

You can use content libraries to share VM and vApp templates, and other types of files, such as ISO images, text files, and so on, across the vCenter Server instances in the same or different locations. Sharing templates across your virtual environment promotes consistency, compliance, efficiency, and automation in deploying workloads at scale.

You use library items to store and manage content in a content library. A single library item can contain one or more files. For example, an OVF package consists of a `.vmdk` file, manifest file, and others, but it is represented by a single library item.

Starting with vSphere 7.0 Update 3, you can add a security policy to a local or subscribed content library and thus protect your system when you synchronize or download library content from third party providers.

What Is a Content Library

A content library instance represents a container for a set of library items. A content library item instance represents the logical object stored in the content library, which might be one or more usable files.

You create and manage the content of a content library on a single vCenter Server instance, but you can distribute the content to other vCenter Server instances. Depending on your needs, you can maintain two types of content libraries: local and subscribed. You can shape the contents of a library item and then combine several library items in a local content library. Furthermore, you can publish the library to make its content available to other users.

Content Library Types

You can create two types of libraries, local and subscribed.

Local library.

You can create a local library as the source for content you want to save or share. Create the local library on a single vCenter Server instance. You can add items to a local library or remove them. You can publish a local library and as a result this content library service endpoint can be accessed by other vCenter Server instances in your virtual environment. When you publish a library, you can configure a password for authentication.

Subscribed library.

You can create a subscribed library and populate its content by synchronizing to a local library. A subscribed library contains copies of the local library files or just the metadata of the library items. The local library can be located on the same vCenter Server instance as the subscribed library, or the subscribed library can reference a local library

on a different vCenter Server instance. You cannot add library items to a subscribed library. You can only add items to the source library. After synchronization, both libraries will contain the same items.

Content Library Items

Library items are VM templates, vApp templates, or other VMware objects that can be contained in a content library.

VMs and vApps have several files, such as log files, disk files, memory files, and snapshot files that are part of a single library item. You can create library items in a specific local library or remove items from a local library. You can also upload files to an item in a local library so that the libraries subscribed to it can download the files to their NFS or SMB server, or datastore.

For information about the tasks that you can perform by using the content library service, see .

Content Library Storage

When you create a local library, you can store its contents on a datastore managed by the vCenter Server instance or on a remote file system.

Depending on the type of storage that you have, you can use Virtual Machine File System (VMFS) or Network File System (NFS) for storing content on a datastore.

For storing content on a remote file system, you can enter the path to the NFS storage that is mounted on the Linux file system of the vCenter Server instance. For example, you can use the following URI formats: `nfs://<server>/<path>?version=4` and `nfs://<server>/<path>`. If you have a vCenter Server instance that runs on a Windows machine, you can specify the Server Massage Block (SMB) URI to the Windows shared folders that store the library content. For example, you can use the following URI format: `smb://<unc-server>/<path>`.

Java

This example is based on the code in the `LibraryCrud.java` sample file and shows how to store library content on a datastore.

NOTE
For a complete and up-to-date version of the Java sample code, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...


// Create a StorageBacking instance for storing the library content on a datastore.


     StorageBacking libraryBacking = new StorageBacking();

     libraryBacking.setType(Type.DATASTORE);


/*

 * Pass the value of the datastore ManagedObjectReference.

 * See the  vSphere Web Services SDK Programming Guide

 * and the vSphere Web Services SDK samples. In addition, the  vSphere
                                                Automation SDK for Java provides
```

```
 * the VimUtil utility class in the vmware.samples.vim.helpers package.
 * You can use the utility to retrieve the ManagedObjectReference
 * of the datastore entity.
 */
     libraryBacking.setDatastoreId("datastore-123");


// Create a LibraryModel that represents a local library backed on a datastore.


      LibraryModel libraryModel = new LibraryModel();
      libraryModel.setName("AcmeLibrary");
      libraryModel.setDescription("Local library backed by VC datastore");
      libraryModel.setType(LibraryType.LOCAL);
      libraryModel.setStorageBackings(Collections.singletonList(libraryBacking));
```

Python

This example is based on the code in the `library_crud.py` sample file and shows how to store library content on a datastore.

> **NOTE**
> For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...


# Create a StorageBacking instance of datastore type.
library_backing = library_client.StorageBacking()
library_backing.type = library_client.StorageBacking.Type.DATASTORE


# Pass the value of the datastore managed object reference.
# The                                                vSphere Automation SDK for Python
 contains
# the GetDatastoreByName class, which sample resource is located in the
# /client/samples/src/com/vmware/vcloud/suite/sample/vim/helpers/ directory.
# You can use the utility to retrieve the managed object reference of the datastore
entity.
library_backing.datastore_id = 'datastore-123'
```

```
# Create a LibraryModel that represents a local library backed on a datastore.
library_model = content_client.LibraryModel()
library_model.name = 'AcmeLibrary'
library_model.storage_backings = [library_backing]
```

Querying Content Libraries

You can create queries to find libraries that match your criteria. You can also retrieve a list of all libraries or only the libraries of a specific type.

Listing All Content Libraries

You can retrieve a list of all content library IDs in your virtual environment, regardless of their type, by using the `Library` service. You can use the `List Library` operation to retrieve all local and subscribed libraries in your system.

Listing Content Libraries of a Specific Type

You can use the vSphere Automation API to retrieve content libraries of a specific type. For example, you can list only the local libraries in your virtual environment.

If you want to retrieve only a list of the local libraries, you must retrieve the `LocalLibrary` service and use the `List Local Library` operation on the `LocalLibrary` service. To list only subscribed libraries, you must retrieve the `SubscribedLibrary` service and call the `List Subscribed Library` operation on the `SubscribedLibrary` service.

Listing Content Libraries by Using Specific Search Criteria

You can filter the list of content libraries and retrieve only the libraries that match your specific criteria. For example, you might want to publish all local libraries with a specific name.

To filter with specific search criteria, use the `Find Library` operation of the `Library` service. Specify the properties of the `LibraryFindSpec` data structure that contains your search criteria. Upon a successful completion of the operation, you receive a list of all content libraries that match your search criteria.

Working with Content Libraries

The Content Library API provides services that you can use to create and manage content libraries.

You can create a local library and publish it for the entire virtual environment. You can also subscribe to use the contents of a local library and enable automatic synchronization to ensure that you have the latest content.

You can perform content library operations by using HTTP requests. Some operations require you to specify parameters in the body of the HTTP request according to your environment. For details about the syntax of each HTTP request body, see the *API Reference* documentation.

Creating a Local Content Library

You can create a local content library by using the `POST https://<vcenter_ip_address_or_fqdn>/api/content/local-library` HTTP request.

Publishing an Existing Content Library

To make the library content available for other vCenter Server instances across the vSphere Automation environment, you must publish the library. Depending on your workflow, select a method for publishing the local library. You can publish a local library that exists in your vSphere Automation environment by using the `POST https://`

`<vcenter_ip_address_or_fqdn>/api/content/local-library/<library_id>?action=publish` HTTP request.

Subscribing to a Content Library

You can subscribe to local content libraries. When you subscribe to a library, you must specify the backing storage for the library content. If the library requires basic authentication, you must also provide the correct user name and password. You can create a subscribed library by using the `POST https://<vcenter_ip_address_or_fqdn>/api/content/subscribed-library` HTTP request.

Synchronizing a Subscribed Content Library

When you subscribe to a published library, you can configure the settings for downloading and updating the library content. You can force the synchronization of a subscribed library by using the `POST https://<vcenter_ip_address_or_fqdn>/api/content/subscribed-library/<library_id>?action=sync` HTTP request.

Editing the Settings of a Content Library

You can update the settings of content library types in your virtual environment. You can update the settings of a local library by using the `PATCH https://<vcenter_ip_address_or_fqdn>/api/content/local-library/<library_id>` HTTP request. You can update the settings of a subscribed library by using the `PATCH https://<vcenter_ip_address_or_fqdn>/api/content/subscribed-library/<library_id>` HTTP request.

Removing the Content of a Subscribed Library

You can free storage space in your virtual environment by removing the subscribed library content that you no longer need.

You can create a subscribed library with the option to download the library content on demand. As a result, only the metadata for the library items is stored in the associated with the subscribed library storage. When you want to deploy a virtual machine from a VM temple in the subscribed library, you must synchronize the subscribed library to download the entire published library content. When you no longer need the VM template, you can use the `POST https://<vcenter_ip_address_or_fqdn>/api/content/subscribed-library/<library_id>?action=evict` HTTP request. You must provide the subscribed library ID to this function. As a result, the subscribed library content that is cached on the backing storage is deleted.

If the subscribed library is not configured to synchronize on demand, an exception is thrown. In this case the subscribed library always attempts to have the most recent published library content.

Delete a Content Library

When you no longer need a content library, you can use the `Delete` operation on either the `Local Library` or the `Subscribed Library` service depending on the library type. You can delete a local library by using the `DELETE https://<vcenter_ip_address_or_fqdn>/api/content/local-library/<library_id>` HTTP request. You can delete a subscribed library by using the `DELETE https://<vcenter_ip_address_or_fqdn>/api/content/subscribed-library/<library_id>` HTTP request.

Working with Library Items

A library item groups multiple files within one logical unit. You can perform various tasks with the items in a content library.

You can upload files to a library item in a local library and update existing items. You can download the content of a library item from a subscribed library and use the item, for example, to deploy a virtual machine. You can remove the content of a library item from a subscribed library to free storage space and keep only the metadata of the library item. When you no longer need local library items, you can delete them and they are removed from the subscribed library when a synchronization task is completed.

You can create a library item from a specific item type, for example, `.ovf` and VM template. The Content Library service must support the library item type to handle the item correctly. If no support is provided for a specified type, the Content Library service handles the library item in the default way, without adding metadata to the library item or guiding the upload process. For information about the supported VM template types, see the *vSphere Virtual Machine Administration* documentation.

Creating an Empty Library Item

You can create as many library items as needed and associate them with a local content library. You can create a library item by using the `POST https://<vcenter_ip_address_or_fqdn>/api/content/library/item` HTTP request.

Querying Library Items

You can perform numerous query operations on library items.

You can retrieve a list of all items in a library, retrieve a library item that has a specific type or name, and find a library item that is not cached on the disk. You can then update the library item content from the subscribed library.

You can use the `GET https://<vcenter_ip_address_or_fqdn>/api/content/library/item?library_id` HTTP request to retrieve a list of all items in a particular library.

You can filter the items contained in a library and retrieve only the items matching specific criteria. For example, you might want to remove or update only specific items in a library. You can use the `POST https://<vcenter_ip_address_or_fqdn>/api/content/library/item?action=find` HTTP request to filter items in a particular library.

Editing the Settings of a Library Item

You can use the `PATCH https://<vcenter_ip_address_or_fqdn>/api/content/library/item/<library_item_id>` HTTP request to edit the properties of a library item such as name, description, and type.

Uploading a File from a Local System to a Library Item

You can upload different types of files from a local system to a library item that you want to use in the vSphere Automation environment.

To upload a file, you must create an empty library item and verify that you have access to the `UpdateSession` and `File` services.

1. Use the `LibraryItemUpdateSessionModel` data structure to track the changes that you make to the library item.
2. Create an update session by using the `UpdateSession` service.
3. Use the `LibraryItemUpdatesessionFileAddSpec` data structure to describe the upload method and other properties of the file to be uploaded.
4. Create the request for changing the item by using the `File` service.
5. Upload the file that is on the local system.
6. Complete and delete the update session to apply the changes to the library item.

Upload a File from a URL to a Library Item

You can upload different types of files from a URL to a library item that you want to use in the vSphere Automation environment.

To upload a file, you must first create an empty library item and verify that you have access to the `UpdateSession` and `File` services.

1. Use the `LibraryItemUpdateSessionModel` data structure to track the changes that you make to the library item.
2. Create an update session by using the `UpdateSession` service.
3. Create a file specification to describe the upload method and other properties of the file to be uploaded.
4. Specify the location of the file you want to upload by using the `LibraryItemTransferEndpoint` data structure.
5. Add the file source endpoint to the file specification.
6. Create a request for changing the item by using the configured file specification.
7. Complete the update session to apply the changes to the library item.

Downloading Files to a Local System from a Library Item

You might want to download files to a local system from a library item and then make changes to the files before you use them.

1. Create a download session model to specify the item, which contains the file that you want to download.
2. Access the `File` service and retrieve the file that you want to export to your system within the new download session.
3. Prepare the files that you want to download and wait until the files are in the prepared state.
4. Retrieve the download endpoint URI of the files.
5. Download the files by using an HTTP GET request.
6. Delete the download session after all files are downloaded.

Synchronizing a Library Item in a Subscribed Content Library

The items in a subscribed library have features that are distinct from the items in a local library. Synchronizing the content and the metadata of an item in a subscribed library depends on the synchronization mechanism of the subscribed library. You can use the `PATCH https://<vcenter_ip_address_or_fqdn>/api/content/library/subscribed-item/<library_item_id>?action=sync` HTTP request to force the synchronization of an individual library item in a subscribed library.

Removing the Content of a Library Item

You can remove the content from a library item to free space on your storage.

If you create a subscribed library with the option to synchronize library content on demand, only the metadata for the library items is stored. When you want to use the items in the library, you must force synchronization on the items to download their content. When you no longer need the files in an item, you can remove the cached content of the library item and free storage space. You can use the `POST https://<vcenter_ip_address_or_fqdn>/api/content/library/subscribed-item/<library_item_id>?action=evict` HTTP request to evict the cached content of a library item in a subscribed library.

Deleting a Library Item

You can remove a library item from a local library when you no longer need it.

You can use the `DELETE https://<vcenter_ip_address_or_fqdn>/api/content/library/item/<library_item_id>` HTTP request to remove a library item from a library. The item content is asynchronously removed from the storage.

You cannot remove items from a subscribed library. If you remove an item from a local library, the item is removed from the subscribed library when you perform a synchronization task on the subscribed library item.

Content Library Support for OVF and OVA Packages

You can use the objects and methods provided by the Content Library API to manage OVF and OVA packages.

Open Virtualization Format (OVF) is an industry standard that describes metadata about a virtual machine image in an XML format. An OVF package includes an XML descriptor file and optionally disk images, resource files (such as ISO files), manifest files, and certificate files.

An OVA package is a single file that contains all OVF package files in an archived form. After you upload an OVA package to a content library, the OVA file is converted to the standard OVF package.

When you try to upload signed content to a content library, you might receive preview warnings. Signed content can be either OVF or OVA packages that contain manifest and certificate files. If you do not respond to the preview warnings, the upload fails. To complete an upload operation successfully, you must ignore any preview warnings by using the .

With the vSphere Automation API, you can use the OVF package in a content library to deploy virtual machines and vApps on hosts, resource pools, and clusters. You can also use the API to create OVF packages in content libraries from vApps and virtual machines on hosts, resource pools, and clusters.

When you create library items to store OVF packages, you must set the item type to `ovf`. To comply with the specific standards of the OVF packages, the vSphere Automation API provides the .

Working with OVF and OVA Packages in a Content Library

You can upload an OVF or OVA package to a library item by using the `UpdateSession`. You can also download an OVF and OVA packages from a content library to your local file system.

In case you want to upload an OVF package, the location of the content determines whether you can pull the content from a URL or push the content directly to a content library. For information about uploading content to library items, see and .

To download the files that are included in an OVF or OVA package to your local file system, use the `DownloadSession`. For more information, see .

Upload an OVF or an OVA Package from a URL to a Library Item

You can upload an OVF or an OVA package from a Web server to a library item.

- Create a new local content library or retrieve the desired existing content library.
- Required privileges: **Content library › Add library** item and **Content library › Update files** on the library.

> **NOTE**
> If you try to upload a signed OVF package and it returns preview warnings, you must ignore the preview warnings to complete the upload.

1. Create an empty library item.
2. Create an update session object.
3. Create an to describe the properties and the upload location of the OVF descriptor file or of the OVA package file.
4. Link the to the update session.

   All files that are included in the OVF package are automatically uploaded.

5. Complete the asynchronous transfer.

Upload an OVF or OVA Package from a Local File System to a Library Item

You can upload an OVF or OVA package from a local file system. This procedure describes how to use the after you have created a library item and initiated an update session.

- Create a new local content library or retrieve the desired existing content library.
- Required privileges: **Content library › Add library item** and **Content library › Update files** on the library.

NOTE
If you try to upload a signed OVF package and it returns preview warnings, you must ignore the preview warnings to complete the upload.

1. Create a library item.
2. Create an update session.
3. Create to describe the properties and the upload locations of the OVF descriptor file or of the OVA package file.
4. Link the to the update session.
5. Create for each VMDK file included in the OVF package.
6. Add all to the update session.

   If you upload an OVF package and it has a VMDK file included, you must repeat steps 5 and 6. If you are uploading a signed OVF package, steps 5 and 6 must also be repeated for the manifest and certificate files included in the OVF package.

7. Initiate the upload operation.
8. Complete the update session.
9. Delete the session.

Creating Virtual Machines and vApps from Templates in a Content Library

You can create VM and OVF templates from virtual machines and vApps in your inventory. You can then deploy virtual machines and vApps from the templates that are stored in a content library.

Create a VM Template in a Content Library from a Virtual Machine

By using the vSphere AutomationAPI, you can create a VM template in a content library from an existing virtual machine in your vCenter Server inventory.

- Verify that you have administrative privileges on your vCenter Server instance.
- Verify that you created a vSphere Automation session to your vCenter Server.
- Verify that you created a local library by using the vSphere Client or the vSphere Automation APIs.

When you call the `create` function of the `com.vmware.vcenter.vm_template.LibraryItems` service, a VM template is created as a library item in your local content library. If the operation is successful, the `LibraryItems` service returns the ID of the newly created library item.
To create a library item that contains a VM template, you can use the . You can review the information about a VM template by using the . For information about how to create a VM template by using the vSphere Client, see the *`vSphere Virtual Machine Administration`* documentation.
For information about the available and mandatory parameters, see the *`API Reference`* documentation.

1. Get the ID of your ESXi host on which you want to store the VM template.

   You can use the .

2. Get the ID of the datastore on which you want to store the VM template files.

   You can use the .

3. Get the ID of the virtual machine that you want to save as a VM template.

   You can use the .

4. Get the ID of your local library.

You can get the list of the local libraries in your vCenter Server and review the information about each library by using the .

5. Create a library item specification for the VM template.

   You can use the .

   a) Specify the local library, source virtual machine, and the name of the library item by using the `library`, `source_vm`, and `name` parameters. You must use the IDs of the local library and source virtual machine.
   b) Specify the placement information for your VM template.

      You can use the . To specify the host, resource pool, cluster, and folder, you must use their IDs.

   c) Specify the datastore on which you want to store the log, configuration, and disk files of your VM template.

      To specify the storage backing for the VM template, you can use the . You must use the ID of the datastore.

   d) Include the placement and storage specifications in the library item specification.
6. Create a library item for storing the VM template.

   You can use the .

If the operation is successful, the `LibraryItems` service returns the ID of the library item that contains the VM template. For information about the available responses, see the *API Reference* documentation.

- Review the information stored in the library item by using the . If you did not save the ID of the library item holding the VM template, you can check the UUID by using the vSphere Client. The URN ends with the ID of the library item and has the following format: `urn:vapi:com.vmware.content.library.Item:<VMTemplateItemID>`.

Create an OVF Template in a Content Library from a Virtual Machine or vApp

You can create library items from existing virtual machines or vApp. Use those library items later to deploy virtual machines and vApps on hosts and clusters in your vCenter Server environment.

1. Create a instance to specify the source virtual machine or vApp to be captured in an OVF template.
2. Create a instance to identify the content library where the OVF template is stored.
3. Create a instance to specify the properties of the OVF template.
4. Initiate a synchronous create operation by .
5. Verify the outcome of the create operation.

Deploy a Virtual Machine from a VM Template in a Content Library

By using the vSphere Automation APIs, you can deploy a virtual machine from a VM template stored in a content library.

- Verify that you have administrative privileges on your vCenter Server instance.
- Verify that you created a vSphere Automation session to your vCenter Server instance.

To deploy a virtual machine from a VM template in a content library, . You can specify the power state and customize the guest operation system prior to the virtual machine deployment.
For information about the available and mandatory parameters, see the *API Reference* documentation.

1. Review the information stored in the VM template library item.

   You can use the . If you did not save the ID of your item, you can select the UUID of your VM template item by using the vSphere Client. The URN ends with the ID of the item and has the following format: `urn:vapi:com.vmware.content.library.Item:<VMTemplateItemID>`.

2. Get the ID of the host on which you want to deploy the virtual machine.

   You can use the .

3. Get the ID of the resource pool to which you want to add your virtual machine.

   You can use the .

4. Get the ID of the `VIRTUAL_MACHINE` folder to which you want to add your virtual machine.

   You can use the .

5. Get the ID of the datastore on which you want to store log, configuration, and disk files of the virtual machine.

   You can use the .

6. Create a deployment specification.

   You can use the .

   a) Specify a name and description of the virtual machine that you want to deploy.
   b) Specify the place in your inventory on which you want to deploy the virtual machine such as an ESXi host, resource pool, and VM folder.

      You can use the . You must use the IDs of your inventory objects.

   c) Specify the datastore on which you want to store the log, configuration, and disk files of the virtual machine. You must use the ID of the datastore.

      You can use the .

   d) Specify the guest operating system and hardware customization specifications that you want to apply to the virtual machine during the deployment process. Add this information to the deployment specification.

      You can use the . You can get a list of the guest operating system customization specifications that are available in your vCenter Server by using the .

   e) Include the placement and storage specifications in the deployment specification.

7. Deploy a virtual machine from your VM template.

   You can use the .

If the operation is successful, the ID of the deployed virtual machine is returned. For information about the possible exceptions, see the *`API Reference`* documentation.

Deploy a Virtual Machine or vApp from an OVF Template in a Content Library

You can use the service to deploy a virtual machine or vApp on a host, cluster, or resource pool from a library item.

1. Create a instance to specify the deployment location of the virtual machine or vApp.
2. Create a instance to define all necessary parameters for the deployment operation.
   For example, you can assign a name for the deployed virtual machine or vApp, and accept the End User License Agreements (EULAs) to complete the deployment successfully.
3. Retrieve information from the descriptor file of the OVF template and use the information during the OVF template deployment.
4. .
5. Verify the outcome of the deployment operation.

vSphere Tag Service

The vSphere Automation Tag Service supports the definition of tags that you can associate with vSphere objects or vSphere Automation resources.

Starting with vSphere 6.5, the vSphere Automation APIs provide programmatic access to creating and managing vSphere tags in your vSphere inventory.

For example, if you want to tag your virtual machines by guest operating system type, you can create a category called `operating system`. You can specify that it applies to virtual machines only and that only a single tag can be applied to a virtual machines at any time. This category can include the following tags: `Windows`, `Linux`, and `Mac OS`.

Creating vSphere Tags

You create a vSphere tag to add metadata to objects in the vSphere inventory.

Tags are grouped in categories and each tag must have at least one category related to it. After you create the tag, you can associate the tag with a vSphere object.

Tags and categories can span multiple vCenter Server instances.

- If multiple on-premises vCenter Server instances are configured to use Enhanced Linked Mode, tags and tag categories are replicated across all these vCenter Server instances.
- When you use Hybrid Linked Mode, tags and tag categories are maintained across your linked domain. That means the on-premises SDDC and the VMware Cloud on AWS SDDC share tags and tag attributes. For more information about Hybrid Linked Mode, see "Hybrid Linked Mode" in the *VMware Cloud on AWS Product Documentation*.

Creating a Tag Category

You create tags in the context of a tag category. You must create a category before you can add tags within that category.

A tag category has the following properties:

- name
- description
- cardinality, or how many tags it can contain
- the types of elements to which the tags can be assigned

You can associate tags with both vSphere API managed objects and VMware vSphere Automation API resources.

Creating a Tag

After you create a tag category, you can create tags within that category.

A tag has the following properties:

- name
- description
- category ID

Java

This example is based on code in the `TaggingWorkflow.java` sample file.

This example creates a tag specification and then uses it to create the tag. The tag specification references the category identifier that was returned from the category create operation. Use the returned tag identifier for subsequent operations on the tag.

NOTE
For a complete and up-to-date version of the Java sample code, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...


Tag tagStub = myStubFactory.createStub(Tag.class,
                                       myStubConfiguration);


// Set up a tag create spec.
TagTypes.CreateSpec spec = new TagTypes.CreateSpec();
spec.setName("red");
spec.setDescription("My favorite color");
spec.setCategoryId(newCategoryId);


String tagId = tagStub.create(spec);
```

Python

This example is based on code in the `tagging_workflow.py` sample file.

This example creates a tag specification and then uses it to create the tag. The tag specification references the category identifier that was returned from the category create operation. Use the returned tag identifier for subsequent operations on the tag.

NOTE
For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...


# Set up a tag create spec.
tag_create_spec = tag_stub.CreateSpec(name='red',
                                      description='My favorite color',
                                      category_id=fav_category_id)


# Create the tag.
tag_stub = tagging_client.Tag(my_stub_config)
tag_id = tag_stub.create(create_spec)
```

Creating Tag Associations

After you create a tag category and create a tag within the category, you can associate the tag with a vSphere managed object or a vSphere Automation resource.

An association is a simple link that contains no data of its own. You can enumerate objects that are attached to a tag or tags that are attached to an object.

Tag associations are local to a vCenter Server instance. When you request a list of tag associations from a vCenter Server system, it enumerates only the associations that it has stored.

When you associate a tag with an object, the object's type must match one of the associable types specified for the category to which the tag belongs.

Assign the Tag to a Content Library

After you create a tag, you can assign the tag to a vSphere Automation resource.

1. Construct a dynamic object identifier for the library.

   The dynamic identifier includes the type and ID of the object.

2. Attach the tag to the content library.

   ```
   POST https://<vcenter_ip_address_or_fqdn>/api/cis/tagging/tag-association/<tag_id>?
   action=attach
   ```

Assign a Tag to a Cluster

After you create a tag, you can assign the tag to a vSphere managed object. Tags make the inventory objects in your virtual environment more sortable and searchable.

Obtain the managed object identifier for the specified cluster.

To get the managed object identifier of the `ClusterComputeResource`, you must access vCenter Server by using the vSphere Web Services API.

This procedure describes the steps for applying a tag to a cluster object in your inventory.

1. Construct a dynamic object identifier for the cluster.

   The dynamic identifier includes the type and ID of the managed object reference.

2. Attach the tag to the cluster.

   ```
   POST https://<vcenter_ip_address_or_fqdn>/api/cis/tagging/tag-association/<tag_id>?
   action=attach
   ```

Updating a Tag

To update a tag, you must create an update spec for the tag.

In the update spec, you set values for the fields to be changed, and omit values for the other fields. When you do an update operation using the update spec, only the fields that contain values are changed.

For example, you might use a timestamp in a tag description to identify a resource's last reconfiguration. After reconfiguring the resource, you update the tag description to contain the current time.

Java

This example is based on code in the `TaggingWorkflow.java` sample file.

This example adds timestamp in a tag description to identify when a resource was last reconfigured. The tag description is updated with the timestamp after the resources is reconfigured.

NOTE
For a complete and up-to-date version of the Java sample code, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...


String newDateTime = Dateformat.getDateInstance().format(new Date());

String newDescription = String.format("Server tag updated at (%s).", newDateTime);


TagTypes.UpdateSpec updateSpec = new TagTypes.UpdateSpec();

updateSpec.setDescription(newDescription);

tagStub.update(myTagId, updateSpec);
```

Python

This example is based on code in the `tagging_workflow.py` sample file.

This example adds timestamp in a tag description to identify when a resource was last reconfigured. The tag description is updated with the timestamp after the resources is reconfigured.

NOTE
For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...


tag_stub = tagging_client.Tag(my_stub_config


# 1 - Format the current time.

date_time = time.strftime('%d/%m/%Y %H:%M:%S')

description = 'Server tag updated at ' + date_time


# 2 - Set up a tag update spec.

tag_update_spec = tag_stub.UpdateSpec()

tag_update_spec.description = description


# 3 - Apply the update spec to change the tag description.

tag_stub.update(tag_id, tag_update_spec)
```

Using Tags to Create and Manage Compute Policies

Compute policies rely on tags to identify inventory objects on which to enforce a policy.

Create a Compute Policy

You can create a compute policy and check the compliance status of the policy for a specific virtual machine.

1. Retrieve the object ID of a virtual machine.
2. Retrieve the object ID of the host you want the virtual machine to run on.

   NOTE
   The host should be different from the host the virtual machine is already running on.

3. Tag the virtual machine.
   For example, use the *tag-1* tag.
4. Tag the host.
   For example, use the *tag-2* tag.
5. Create a compute policy.
   For example, use the *tag-1* and *tag-2* tags.
6. Check the compliance status of the policy on this virtual machine.

   NOTE
   The compliance status can take up to 3 minutes to update.

vSphere Security

This chapter provides information about securing your vSphere environment for vCenter Server and ESXi.

Managing Certificates

Starting with vSphere 6.7 Update 2, you can use the vSphere Automation API to manage certificates in your vSphere environment. You can not only refresh default certificates that are issued by the VMware Certificate Authority (VMCA) but also add third-party or custom-made certificates to your environment.

Certificate Management Operations

Use the vSphere Automation API to manage trusted root certificate chains, VMware Certificate Authority (VMCA) root certificates, machine SSL (TLS) certificates, and Security Token Service (STS) signing certificates.

With the vSphere Automation API, you can refresh the VMCA-issued certificates but also add external and third-party certificates to your vSphere environment. For more information on vSphere certificate management, see the *vSphere Authentication* guide.

Certificate Management Interfaces

You can use the following interfaces to manage certificates with the vSphere Automation API:

- `Trusted Root Chains`
- `VMCA Root`
- `TLS CSR`
- `TLS`

- Signing Certificate

Certificate Management Operations

You can use the operations listed in the following table to manage certificates.

Table 52: Certificate Management Operations

Operation	Interface	HTTP Request	Description	Introduced in
List trusted root certificates	Trusted Root Chains	GET https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/trusted-root-chains	You can retrieve the identifiers of all trusted root certificates that are present in vCenter Server.	vSphere 6.7 U2
Get trusted root certificate information	Trusted Root Chains	GET https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/trusted-root-chains/<*chain*>	You can retrieve a root certificate chain by providing its identifier. You can retrieve the identifier by using the List trusted root certificates operation.	vSphere 6.7 U2
Add a trusted root certificate	Trusted Root Chains	POST https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/trusted-root-chains	Use this operation to add a trusted root certificate chain to your vCenter Server system.	vSphere 6.7 U2
Delete a trusted root certificate	Trusted Root Chains	DELETE https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/trusted-root-chains/<*chain*>	You can delete a root certificate by providing its unique identifier. You can retrieve the identifier by using the List trusted root certificates operation.	vSphere 6.7 U2
Replace the VMCA root certificate	VMCA Root	POST https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/vmca-root	You can replace the VMCA root certificate with a new VMCA-signed certificate. The operation triggers a restart of the services that are using this certificate.	vSphere 7.0
Generate a CSR	TLS CSR	POST https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/tls-csr	You can generate a CSR and use it to issue a custom certificate. If the operation is successful, you receive a CSR in PEM format.	vSphere 6.7 U2
Get the Machine SSL certificate	TLS	GET https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/tls	You can retrieve the Machine SSL certificate of your vCenter Server system.	vSphere 6.7 U2
Renew the Machine SSL certificate	TLS	POST https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/tls?action=renew	You can renew the validity of the machine SSL certificate for a specified period in days. The duration must be less than or equal to 730 days.	vSphere 6.7 U2

Table continued on next page

Continued from previous page

Operation	Interface	HTTP Request	Description	Introduced in
Replace the Machine SSL certificate with a custom signed certificate	`TLS`	`PUT https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/tls`	You can replace the vCenter Server Machine SSL certificate with a custom certificate signed by an external Certificate Authority (CA).	vSphere 6.7 U2
Replace the Machine SSL certificate with a VMCA-signed certificate	`TLS`	`POST https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/tls?action=replace-vmca-signed`	You can replace the vCenter Server Machine SSL certificate with a VMCA-signed certificate.	vSphere 7.0
Retrieve the STS signing certificate chains	`Signing Certificate`	`GET https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/signing-certificate`	You can retrieve the STS signing certificate chains, which are used for validating tokens signed by vCenter Server.	vSphere 7.0 U3
Replace the STS signing certificate	`Signing Certificate`	`PUT https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/signing-certificate`	You can replace the current STS signing certificate with a certificate of your choice. The accepted file format is PEM.	vSphere 7.0 U3
Refresh the STS signing certificate	`Signing Certificate`	`POST https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/signing-certificate?action=refresh`	You can replace the current STS signing certificate with a new VMCA-signed certificate. The newly-generated certificate is set as the active STS signing certificate for the vCenter Server token service.	vSphere 7.0 U3

Add a Root Certificate to vCenter Server

You can use the `Certificate Management vCenter Trusted Root Chains` interface to add, delete and read trusted root certificate chains.

- Verify that you are connected to a vSphere Automation API server.
- Verify that the root certificate or certificate chain you want to add is available on your machine.
- Verify that you have the required privileges: **CertificateManagement › Manage** and **CertificateManagement › Admi nister**.

If you want to use an enterprise or third-party certificate authority (CA) for certificate management of your vSphere environment, you must first establish trust with that CA. You can do this by adding the root certificate of the external CA to the trusted root store of your vCenter Server system.
Adding a root certificate or certificate chain to the vCenter Server trusted certificate store establishes trust with an enterprise or third-party certificate authority. You can add a root certificate to vCenter Server as a prerequisite for other scenarios such as setting a third-party or enterprise machine SSL certificate.

1. Retrieve the root certificates on your vCenter Server system.

   ```
   GET https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/
   trusted-root-chains
   ```

2. Populate the `TrustedRootChains.CreateSpec` data structure.

Parameter	Type	Description
`cert_chain`	String	Certificate or certificate chain in base64 encoding. The input must be JSON string escaped for newline (\n).

3. Add the certificate or certificate chain.

```
POST https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/
trusted-root-chains
```

If the operation is successful, the system returns the unique identifier of the trusted root certificate you added.

Delete a Root Certificate from vCenter Server

You can use the `Certificate Management vCenter Trusted Root Chains` interface to add, delete and read trusted root certificate chains. This use case demonstrates how to delete a root certificate or certificate chain from the trusted root store of your vCenter Server system.

- Verify that you are connected to a vSphere Automation API server.
- Verify that you have the required privileges for executing the method: **CertificateManagement › Administer** and **CertificateManagement › Manage**.

Deleting certificates is not available through the vSphere Client and you can only do this by using the vSphere Automation API or the CLI tools.

> **CAUTION**
> Deleting a root certificate or certificate chain that is in use might cause breakage of your systems. Proceed to delete a root certificate only if you are sure it is not in use by your vCenter Server or any connected systems.

1. Retrieve the root certificates from your vCenter Server system.

```
GET https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/
trusted-root-chains
```

The system lists the unique identifiers (chains) of the certificates in the trusted root store.

2. Retrieve the certificate you want to delete.

To retrieve and verify a root certificate, use its unique identifier (chain).

```
GET https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/
trusted-root-chains/<chain>
```

3. Delete the certificate by providing its unique identifier (chain).

```
DELETE https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/
vcenter/trusted-root-chains/<chain>
```

The system returns a 204 error, which means that the request was processed but no content is returned.

4. To verify you deleted the certificate, retrieve the root certificates from your vCenter Server system once again.

```
GET https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/
trusted-root-chains
```

Change the Machine SSL Certificate of vCenter Server

You can change the machine SSL certificate of a vCenter Server system by using the TLS and the TLS CSR interfaces of the vSphere Automation API.

- Verify that you are connected to a vSphere Automation API server.
- Verify that the root certificate of the CA you are going to use is available on your machine.
- Verify that you have the required privileges: **CertificateManagement › Administer** and **CertificateManagement › M anage**.

The machine SSL certificate is used for server verification and for secure communication such as HTTPS or LDAPS. The machine certificates are the human-facing certificates in vSphere. They are used to create an SSL socket on the server side to which SSL clients can then connect.
Changing the machine SSL certificate with one issued by an official or enterprise certificate authority is an essential part of the Hybrid Mode of vSphere certificate management. In this mode, you replace the machine SSL certificate and you leave the VMCA to manage all other certificates automatically. The VMCA is a just-enough internal certificate authority that comes integral with your vSphere deployment. It has been purpose-built to serve the certificate needs of your vSphere environment. For more information on vSphere certificate management, see the *vSphere Authentication* guide.

1. Retrieve the current Machine SSL certificate of your vCenter Server system.

   ```
   GET https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/
   tls
   ```

2. Generate a certificate signing request (CSR) by using the `TLS CSR` functionality.
   a) Populate the `CertificateManagementVcenterTlsCsrSpec` data structure.

Parameter	Type	Description
`country`	String	Specifies the country in the certificate subject.
`state_or_province`	String	Specifies the state or province in the certificate subject.
`locality`	String	Specifies the locality in the certificate subject.
`organization`	String	Specifies the organization in the certificate subject.
`organization_unit`	String	Specifies the organization unit in the certificate subject.
`email_address`	String	Specifies the email address in the certificate subject.

   b) Make the request.

   ```
   POST https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/
   vcenter/tls-csr
   ```

   The system returns the CSR in PEM format.
3. Save the CSR to your machine.
4. Send the CSR to the certificate authority of your choice.

   NOTE
   The private key corresponding to the public key generated by the CSR is stored in the vCenter Server keystore and does not exit your system.

5. Save the issued Machine SSL certificate to your machine.
6. Set the new custom certificate to your vCenter Server system by using the `TLS` functionality.
   a) Populate the `CertificateManagementVcenterTlsSpec` data structure.

Parameter	Type	Description
`cert`	String	The Machine SSL certificate in PEM format. You must also paste the intermediate CA certificate, if you have one. The certificates must be JSON string escaped for newline (\n).
`root_cert`	String	The third-party root CA certificate in PEM format. You must also paste the intermediate CA certificate, if you have one. The certificates must be JSON string escaped for newline (\n).

NOTE
You must not provide the private key as it was generated with the CSR and is already saved to your system.

   b) Make the request.

```
PUT https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/
tls
```

The system returns a 204 error, which means that the request was processed successfully but no content is returned.

NOTE
Restart of vCenter Server services after the certificate change is not necessary. The certificate replacement is completed seamlessly and all your sessions remain active.

Refresh the vCenter Server STS Signing Certificate with a VMCA-Issued Certificate

You can refresh the vCenter Server Security Token Service (STS) signing certificate by using the `Certificate Management vCenter Signing Certificate` interface. The STS is an internal entity that issues and verifies tokens so that vSphere services can communicate with and trust each other.

- Verify that you are connected to a vSphere Automation API server.
- Verify that you have the **CertificateManagement › Administer** privilege.

You can refresh the current STS signing certificate of your vCenter Server system with a new VMCA-issued certificate. There are two valid reasons for refreshing your STS signing certificate or certificate chain.

- If it is close to expiry. The standard lifespan of the vCenter Server STS signing certificate is 10 years. Your vCenter Server system will notify you in advance of STS certificate expiry. An alarm is triggered once per week when your STS certificate is 90 days away from expiry, and then daily when seven days away.
- If you already replaced your signing certificate with a third-party or enterprise one and now want to revert back to a default VMCA-issued certificate. This procedure replaces the custom or third-party STS signing certificates you added.

1. Retrieve the current vCenter Server STS signing certificate chain.

   `GET https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/signing-certificate`

2. Refresh the STS signing certificate.
   a) Populate the `SigningCertificateRefreshRequestBody` data structure.

Parameter	Type	Description
`force`	boolean	The default is `false`. You can use this parameter to force a signing certificate refresh in environments that would otherwise prevent refresh from occurring such as mixed-version environments. Use `force` only when it is understood why the refresh fails or if you are instructed to do so by VMware® customer support.

   b) Request the refresh.

   `POST https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/signing-certificate?action=refresh`

If successful, the system returns the x509 certificate chain issued in accordance with the vCenter Server policies.

CAUTION
If you used a forced refresh, you must restart your vCenter Server and all linked services.

Set a Custom STS Signing Certificate to vCenter Server

You can import and replace the vCenter Server STS signing certificate with a custom generated or third-party certificate by using the `Certificate Management vCenter Signing Certificate` interface.

- Verify that you are connected to a vSphere Automation API server.
- Verify that the custom generated or third-party certificate chain and private key are available on your machine.
- Verify that you have the **CertificateManagement › Administer** privilege.

In the usual case, you must not replace the vCenter Server STS signing certificate as it is not an external-facing certificate. The STS is an internal service that enables communication between various vSphere services. A fresh installation of vSphere 7.0 and later comes with a signing certificate that is issued with a default duration of 10 years. Replace the STS signing certificate with a custom or third-party certificate only if your company security policy requires you to do so.

1. Populate the `SigningCertificateSetSpec` data structure.
   a) Populate the correspoding unencrypted PKCS#8 private key as a string in base64-encoded PEM format. It must be JSON string escaped for newline(\n).
   b) Populate the `X509CertChain` data structure.

Parameter	Type	Description
cert_chain	String	The custom generated or third-party certificate chain in base64-encoded PEM format. It must be a valid certificate chain with the leaf cert marked for Digital Signature key usage. The leaf certificate must be first in the sequence and the root must be last. The certificates must be JSON string escaped for newline(\n).

2. Set the STS signing certificate.

   `PUT https://<vcenter_ip_address_or_fqdn>/api/vcenter/certificate-management/vcenter/signing-certificate`

   The system returns a 204 error, which means that the request was processed successfully but no content is returned.

   CAUTION
   The change of the STS signing certificate might leave systems in the local vCenter Server domain in a non-functional state. To prevent system failure, restart your vCenter Server instance and all linked services.

Managing vSphere Native Key Provider

You can use the vSphere Automation to perform VMware vSphere®Native Key Provider™ operations.

In vSphere 7.0 Update 2 and later, you can use the built-in vSphere Native Key Provider to enable encryption technologies, such as virtual TPMs (vTPM).

vSphere Native Key Provider is included in all vSphere editions and does not require an external key server (also called a Key Management Server (KMS) in the industry). You can also use vSphere Native Key Provider for vSphere Virtual Machine Encryption, but you must purchase the VMware vSphere®Enterprise Plus Edition™.

For details about vSphere Native Key Provider, see the *vSphere Security* documentation.

Retrieve vSphere Native Key Providers

You can use to list vSphere Native Key Providers and retrieve details about a specific vSphere Native Key Provider.

1. Retrieve a list of vSphere Native Key Providers.
2. Retrieve information about a vSphere Native Key Provider.

Manage vSphere Native Key Providers

You can use to perform vSphere Native Key Provider management operations.

You can add, update, and remove vSphere Native Key Providers.

1. Add a vSphere Native Key Provider.
2. Update a vSphere Native Key Provider.
3. Remove a vSphere Native Key Provider.

Export and Import vSphere Native Key Provider Configuration

You can use to export and import vSphere Native Key Provider configuration.

You can set a vSphere Native Key Provider as the default for a vCenter Server instance by using the `SetDefaultKmsCluster` method of the `CryptoManagerKmip` managed object from the vSphere Management SDK.

1. Export a vSphere Native Key Provider configuration.

   NOTE
   You must export the vSphere Native Key Provider configuration at least once before you can use a vSphere Native Key Provider.

2. Import a vSphere Native Key Provider configuration.

Retrieve vSphere Native Key Providers on a Host

You can use to list vSphere Native Key Providers on a host and retrieve details about a specific vSphere Native Key Provider.

1. Retrieve a list of vSphere Native Key Providers on a host.
2. Retrieve information about a vSphere Native Key Provider on a host.

vSphere Trust Authority

You can use the vSphere Automation to perform vSphere Trust Authority operations.

vSphere Trust Authority is a foundational technology that enhances workload security. vSphere Trust Authority establishes a greater level of trust in your organization by associating an ESXi host hardware root of trust to the workload itself. For details about vSphere Trust Authority, see the *`vSphere Security`* documentation.

Configure a vSphere Trust Authority Cluster

You can use to perform vSphere Trust Authority Cluster management operations.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can retrieve details about vSphere Trust Authority Clusters, update the state of a cluster, and check the result of the update operation.

1. Retrieve a list of clusters for a vCenter Server instance that are configured as Trust Authority Clusters.
2. Update the state of a cluster.
3. Check the result of the last update operation for the same cluster.

Configure Key Providers

You can use to perform Key Provider management operations.

- Verify that you have access to a working vSphere Trust Authority environment.

- Verify that you have Trusted Infrastructure administrative privileges.

You can retrieve, add, update, remove, and retrieve details about Key Providers.

1. Retrieve a list of Key Providers to see which Key Providers the cluster is using.
2. Add a new Key Provider which all hosts in the cluster can use.
3. Retrieve information about a Key Provider to verify the configuration.
4. Update an existing Key Provider to modify the connection details and primary key for it.
5. Remove a Key Provider.

Establish Trust Between Key Provider and Key Server

You can use to perform trust management operations.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can list and update server certificates, retrieve, generate, and update client certificates, generate a CSR, and set the key server credential.

1. List the remote server certificates on the configured key servers to verify the trusted key servers.
2. Retrieve the list of trusted server certificates.
3. Update the trusted server certificates.

   NOTE
   This operation overwrites the existing list of trusted certificates.

4. Retrieve the existing client certificate.
   If the operation is successful, you receive the client certificate in PEM format.
5. Generate a new self-signed client certificate, used to establish a secure connection to the key server.

   NOTE
   This operation overwrites the existing client certificate.

   If the operation is successful, you can provide the newly generated self-signed client certificate to the key server to establish trust with the Key Provider.
6. Update the client certificate to specify what Key Provider should use to authenticate with the key server.

   NOTE
   If a client certificate exists, this operation overwrites it.

7. Generate a certificate signing request (CSR) for the client certificate.

   NOTE
   If a CSR exists, this operation overwrites it.

   If the operation is successful, you receive the client CSR in PEM format and the host ID which issued it. The generated CSR can later be signed by a third party. The signed CSR should be replicated and set on each host.
8. Set the key server credential for key servers that require a password.

Configure Trusted TPMs of Attested ESXi Hosts on a Cluster Level

You can use to manage remote attestation configuration for TPM trust.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can add, list, remove, and retrieve details about TPM CA certificates and TPM endorsement keys. You can also set and retrieve TPM 2.0 attestation settings.

1. Add a new TPM CA certificate to a Trusted Cluster to specify a trusted platform OEM.
2. Retrieve a list of configured TPM CA certificates on a Trusted Cluster to identify the trusted platform OEMs.
3. Remove a TPM CA certificate from a Trusted Cluster because a platform OEM is no longer trusted.
4. Retrieve details about a specific TPM CA certificate on a Trusted Cluster to get more information about the trusted platform OEM.
5. Add a new TPM endorsement key to a Trusted Cluster to specify a trusted ESXi host.
6. Retrieve a list of configured TPM endorsement keys in a Trusted Cluster to identify the trusted ESXi hosts.
7. Remove a TPM endorsement key from a Trusted Cluster because an ESXi host is no longer trusted.
8. Retrieve details about a specific TPM endorsement key on a Trusted Cluster to get more information about the trusted ESXi host.
9. Set the TPM 2.0 attestation settings by specifying that TPM endorsement keys on a Trusted Cluster do not need to be signed because the trusted OEM does not sign endorsement keys.
10. Determine the TPM 2.0 attestation settings in a Trusted Cluster.

Configure Trusted ESXi Builds on a Cluster Level

You can use to manage trusted instances of ESXi software on a cluster level.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can import, list, remove, and retrieve details about ESXi base images.

1. Import ESXi metadata as a new trusted base image to each host in a vSphere Trust Authority Cluster.
2. Retrieve a list of trusted ESXi base images in a vSphere Trust Authority Cluster.
3. Remove an ESXi base image that should no longer be trusted from a vSphere Trust Authority Cluster.
4. Retrieve details about a trusted ESXi base image version in a vSphere Trust Authority Cluster.

Retrieve vSphere Trust Authority Components Information

You can use to retrieve information about Attestation Service and Key Provider Service instances running on hosts.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can use the retrieved information to connect to the hosts running the vSphere Trust Authority components.

1. Retrieve detailed information, including the certificates, about the Attestation Service instance running on a Trust Authority Host.
2. List Trust Authority Hosts running an Attestation Service instance by using filters.
3. Retrieve detailed information, including the certificates, about the Key Provider Service instance running on a Trust Authority Host.
4. List Trust Authority Hosts running a Key Provider Service instance by using filters.

Configure vSphere Trust Authority Components

You can use to perform Key Provider Service and Attestation Service management operations.

- Verify that you have access to a working vSphere Trust Authority environment.

- Verify that you have Trusted Infrastructure administrative privileges.

You can register, list, remove, and retrieve details about Key Provider Service and Attestation Service instances.

1. Register a Key Provider Service instance in a Workload vCenter Server.
   The Key Provider Service instance is propagated to all Workload ESXi hosts that the Workload vCenter Server manages.
2. Register an Attestation Service instance in a Workload vCenter Server.
   The Attestation Service instance is propagated to all Workload ESXi hosts that the Workload vCenter Server manages.
3. List Key Provider Service instances registered in a Workload vCenter Server by using filters.
4. List Attestation Service instances registered in a Workload vCenter Server by using filters.
5. Remove a registered Key Provider Service instance.
   The Workload ESXi hosts can no longer retrieve keys by using that Key Provider Service instance.
6. Remove a registered Attestation Service instance.
   The Workload ESXi hosts can no longer attest that their configuration is secure by using that Attestation Service instance.
7. Retrieve detailed information, including the certificates, for a registered Key Provider Service instance.
8. Retrieve detailed information, including the certificates, for a registered Attestation Service instance.

Configure vSphere Trust Authority Components for Trusted Clusters

You can use to manage Key Provider Service and Attestation Service instances that a Trusted Cluster is configured to use.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can configure, list, remove, and retrieve details about Key Provider Service and Attestation Service instances.

1. Configure a cluster in a Workload vCenter Server to use a registered Key Provider Service instance.
   You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

   If the operation is successful, the Key Provider Service instance is propagated to all Trusted ESXi hosts in the cluster.
2. Configure a cluster in a Workload vCenter Server to use a registered Attestation Service instance.
   You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

   If the operation is successful, the Attestation Service instance is propagated to all Trusted ESXi hosts in the cluster.
3. List Key Provider Service instances used by a cluster by using filters.
4. List Attestation Service instances used by a cluster by using filters.
5. Remove a Key Provider Service instance from the configuration of a Trusted Cluster.
   If the operation is successful, the Trusted ESXi hosts can no longer retrieve keys by using that Key Provider Service instance.
6. Remove a registered Attestation Service instance from the configuration of a Trusted Cluster.
   If the operation is successful, the Trusted ESXi hosts can no longer attest that their configuration is secure by using that Attestation Service instance.
7. Retrieve detailed information, including the certificates, for a configured Key Provider Service instance used by a Trusted Cluster.
8. Retrieve detailed information, including the certificates, for a registered Attestation Service instance used by a Trusted Cluster.

Establish Trust Between Hosts in a vSphere Trust Authority Cluster and a Workload vCenter Server

You can use to perform trust management operations.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can establish and remove trust between a Workload vCenter Server and the hosts in a vSphere Trust Authority Cluster. You can also list all Workload vCenter Server instances that have established trust with the host in a vSphere Trust Authority Cluster.

1. Establish trust between a vSphere Trust Authority Cluster and a Workload vCenter Server by creating a profile, so that the Workload vCenter Server can retrieve the health status of the vSphere Trust Authority components.
2. Remove the trust between a Workload vCenter Server and the hosts in the vSphere Trust Authority Cluster, so that the Workload vCenter Server stops using the hosts for attestation.
3. List all profiles which the vSphere Trust Authority Cluster trusts.

Check Trusted Cluster Health

You can use to retrieve information about the health of the applied vSphere Trust Authority component configurations in a Trusted Cluster.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can retrieve basic and detailed information about the health of Key Provider Service or Attestation Service configurations applied to a Trusted Cluster with respect to the desired state. You can also retrieve detailed information about the health of all applied vSphere Trust Authority component configurations in a Trusted Cluster.

1. Retrieve a summary about the health status of all Key Provider Service instances configured for use in a Trusted Cluster.
   If the operation is successful, you can verify whether all Key Provider Service configurations of the Trusted Cluster are applied successfully and every host in the cluster is consistent with the desired state.
2. Retrieve detailed information about the health status of a specific Key Provider Service instance configured for use in a Trusted Cluster.
   If the operation is successful, you can verify whether the specified Key Provider Service configuration of the Trusted Cluster is applied successfully and every host in the cluster is consistent with the desired state.
3. Retrieve a summary about the health status of all Attestation Service instances configured for use in a Trusted Cluster.
   If the operation is successful, you can verify whether all Attestation Service configurations of the Trusted Cluster are applied successfully and every host in the cluster is consistent with the desired state.
4. Retrieve detailed information about the health status of a specific Attestation Service instance configured for use in a Trusted Cluster.
   If the operation is successful, you can verify whether the specified Attestation Service configuration of the Trusted Cluster is applied successfully and every host in the cluster is consistent with the desired state.
5. Retrieve detailed information about the health status of all vSphere Trust Authority components configured for use in a Trusted Cluster.
   If the operation is successful, you can verify whether the vSphere Trust Authority component configuration is applied successfully and every host in the cluster is consistent with the desired state.

If there are errors, you can try to remediate the Trusted Cluster. See Remediate a Trusted Cluster.

Remediate a Trusted Cluster

You can use to remediate vSphere Trust Authority component configurations in a Trusted Cluster or remove the configurations.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can update the applied Key Provider Service or Attestation Service configurations in a Trusted Cluster to become consistent with the desired state or you can remove the applied Key Provider Service or Attestation Service configurations. You can also update all applied vSphere Trust Authority component configurations in a Trusted Cluster or remove the configurations. By removing the configurations, you can move hosts from a Trusted Cluster to another cluster.

1. Remediate all Key Provider Service instances configured for use in a Trusted Cluster.
   If the operation is successful, the Key Provider Service configuration of every host in the cluster is consistent with the desired state.
2. Remove all Key Provider Service configurations from a Trusted Cluster.
   If the operation is successful, the applied Key Provider Service configurations are removed from the configuration of every host in the cluster without affecting the desired state.
3. Remediate all Attestation Service instances configured for use in a Trusted Cluster.
   If the operation is successful, the Attestation Service configuration of every host in the cluster is consistent with the desired state.
4. Remove all Attestation Service configurations from a Trusted Cluster.
   If the operation is successful, the applied Attestation Service configurations are removed from the configuration of every host in the cluster without affecting the desired state.
5. Remediate all vSphere Trust Authority components configured for use in a Trusted Cluster.
   If the operation is successful, the vSphere Trust Authority component configuration of every host in the cluster is consistent with the desired state.
6. Remove all vSphere Trust Authority component configurations from a Trusted Cluster.
   If the operation is successful, the applied vSphere Trust Authority component configurations are removed from the configuration of every host in the cluster without affecting the desired state.

You can recheck the Trusted Cluster health after the remediation. See Check Trusted Cluster Health.

Managing Host Hardware Security

You can use the vSphere Automation to perform host hardware management operations.

You can retrieve host hardware TPM information and manage host hardware TPM endorsement keys. You can also register SGX hosts.

Retrieve Host Hardware TPM Information

You can use to retrieve a list of configured TPM devices on a host and information about each TPM device.

1. List configured TPM devices on a host.
2. Retrieve detailed information about a specific TPM device, including the manufacturer, model, and firmware version.

Manage Host Hardware TPM Endorsement Keys

You can use to retrieve a list of configured TPM endorsement keys on a host and information about each endorsement key. You can also retrieve the TPM event log and unseal a secret that is bound to an endorsement key.

You can retrieve the TPM event log for different purposes, such as configuring firmware trust with an attestation service or validating the boot time TPM measurements. You can unseal a secret that is bound to an endorsement key to verify reported measurements. For example, you can verify measurements from the TPM event log.

1. List configured TPM endorsement keys on a host.
2. Retrieve detailed information about a specific TPM endorsement key.
3. Retrieve the event log associated with a TPM device.
4. Unseal a secret that is bound to an endorsement key.

Register an SGX Host

You can use to register hosts by stored Intel Software Guard Extensions (SGX) configuration.

1. Register a host by stored SGX configuration.

vSphere Supervisor Configuration and Management

Starting with vSphere 7.0, you can use the vSphere AutomationAPIs to enable and configure vSphere Supervisor on a vSphere cluster. Then you can run Kubernetes workloads directly on ESXi hosts.

You can use the vSphere Automation to automate the enabling and disabling of vSphere Supervisor on a vSphere cluster, creating and managing namespaces.

The information in this chapter is intended for vSphere administrators who want to use the vSphere AutomationAPIs to configure their environment to run Kubernetes workloads in vSphere. To take fully advantage of these vSphere AutomationAPIs, you must have basic knowledge about the Kubernetes technology and containers.

For more information about how to configure and manage vSphere Supervisor through the vSphere Client, see the *Installing and Configuring vSphere IaaS Control Plane* and *vSphere with Tanzu Configuration and Management* documentation.

vSphere Supervisor Terminology

You must understand the basic terminology in this chapter to be able use the vSphere Supervisor automation APIs effectively.

vSphere Supervisor Basic Terms

Term	Description
Supervisor	A vSphere cluster that has the vSphere Supervisor enabled.
TKG cluster	An upstream Kubernetes cluster provisioned and managed by using the TKG. A TKG resides in a vSphere Namespace. You can deploy workloads and services to such clusters in the same way as you do with standard Supervisor.
vSphere Namespace	A namespace that is created within a Supervisor. Each namespace sets the resource boundaries for CPU, memory, storage, and also the number of Kubernetes objects that can run within the namespace. After a namespace is configured, you can run Kubernetes workloads within the namespace.

Table continued on next page

Continued from previous page

Term	Description
vSphere Pod	A virtual machine with a small footprint that runs one or more Linux containers. A vSphere Pod is equivalent to a Kubernetes pod. vSphere Pods are compatible with the Open Container Initiative (OCI) and can run OCI compatible containers regardless the operating system.
Spherelet	A spherelet is an implementation of the kubelet functionality ported natively on each host in the Supervisor.
Kubernetes Workload	Workloads are applications that are deployed in one of the following ways: • As containers running inside vSphere Pods. • Workloads provisioned through the VM service. • TKGclusters deployed by using the TKG. • Applications running inside the TKG clusters.
Supervisor control plane	vSphere Supervisor creates a Kubernetes control plane directly on the hypervisor layer. The control plane manages the worker nodes and the vSphere Pods in the Supervisor.
Supervisor worker nodes	ESXi hosts that are part of a Supervisor are considered as worker nodes. You run your Kubernetes workloads on the worker nodes.
Container Runtime Executive (CRX)	CRX is an isolated Linux execution environment similar to a VM that works together with ESXi.
VM Service	The VM Service functionality allows DevOps engineers to deploy and manage virtual machines in their Kubernetes environment through standard Kubernetes APIs. vSphere administrators are responsible for providing VM Classes and VM Images for the DevOps engineers to choose from, as well as managing resource allocations to self-service provisioned VMs.
Self-Service Namespace	vSphere administrators can activate the Self-Service Namespace service on a Supervisor and create namespace templates for DevOps engineers to create a vSphere Namespace themselves.
vSphere Zones	vSphere Zones provide high availability against clusters-level failures to workloads deployed on vSphere Supervisor. You can configure a three-zone Supervisor mapped to three vSphere clusters or a one-zone Supervisor mapped to a single vSphere cluster. In a single cluster deployment, the high availability is provided by vSphere HA and is only on a host level.

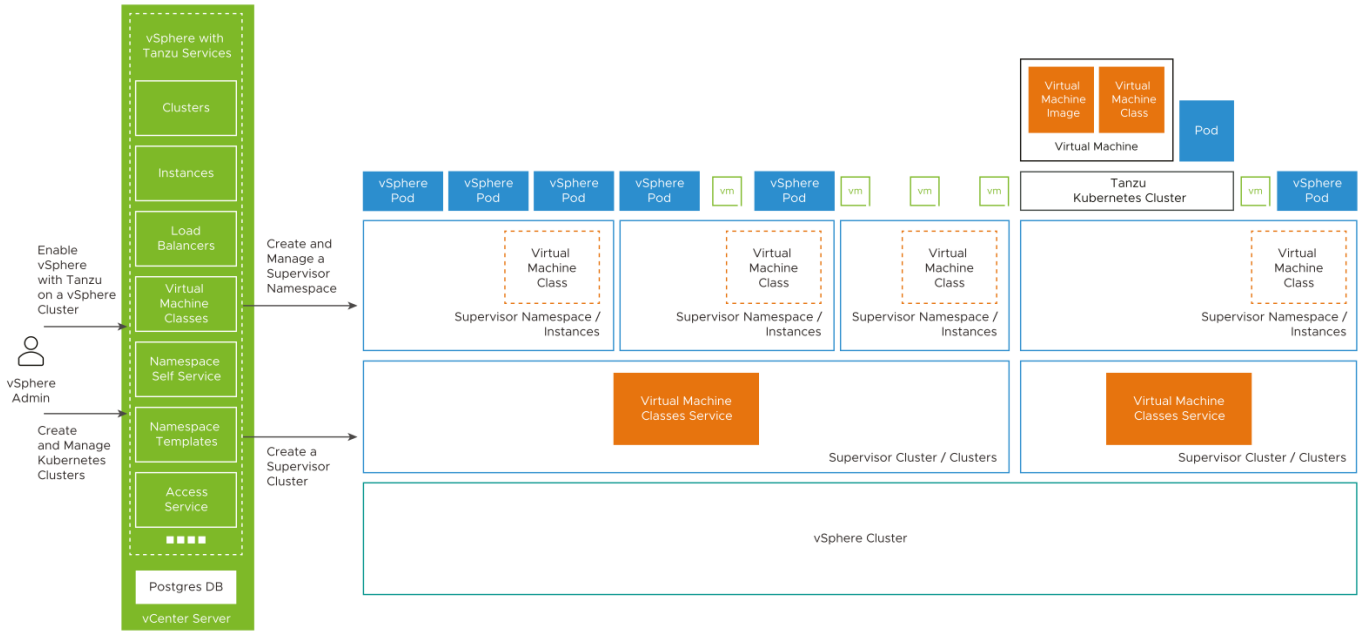
vSphere Supervisor Components and Services

Before you can automate some of the administrative tasks for using vSphere Supervisor, you must first familiarize yourself with the high-level system architecture and components involved.

The vSphere Supervisor API consists of two packages, `namespace_management` and `namespaces`. In the `namespace_management` package, you can find APIs for enabling a vSphere cluster with vSphere Supervisor, configuring the network and storage policies of the Supervisor, upgrading a cluster to the desired version of vSphere

Supervisor, and so on. In the `namespaces` package, you can find APIs for creating, configuring, and deleting a vSphere Namespace, and also for setting the necessary permissions for accessing the namespace.

Figure 95: Services and Components Involved in Using vSphere Supervisor



The vSphere Kubernetes Services component runs on vCenter Server and communicates the vSphere admin requests to the Supervisor control plane. The component comprises of several services which vSphere Automation endpoints you can use to enable vSphere Supervisor on a vSphere cluster and create Kubernetes workloads.

You can use the Cluster Compatibility service to query a vCenter Server instance about the available clusters that meet the requirements for enabling vSphere Supervisor.

You can use the Clusters service to enable or disable vSphere Supervisor on a cluster. You can also reconfigure the settings of a Supervisor.

You can use the Instances service to create, edit, and delete a vSphere Namespace from a Supervisor. You can also change all or some of the settings of an existing namespace.

Starting with vSphere 7.0 Update 1, a Supervisor backed by a vSphere Distributed Switch uses the HAProxy load balancer to provide connectivity to DevOps and external service. The Load Balancer service represents the user provisioned load balancers.

Starting with vSphere 7.0 Update 2a, vSphere administrators can use the VM Service functionality to enable DevOps engineers to deploy and run VMs and containers in one shared Kubernetes environment through a single Kubernetes native interface. Use the vSphere Supervisor APIs to define VM Classes and content libraries to allocate resources to virtual machines provisioned by DevOps engineers.

As of vSphere 7.0 Update 2a, vSphere administrators can also configure a vSphere Namespace as a template on a cluster. Then the DevOps engineers can use it to self-service the creation of vSphere Namespaces and deploy workloads within them.

How to Configure and Manage a Supervisor

You use the `Clusters` service to enable and disable a Supervisor, or edit the configuration of an existing Supervisor. The `Clusters` service is provided within the `namespace_management` package.

You can enable a vSphere cluster to manage Kubernetes workload objects, only after you enable vSphere DRS in a fully automated mode and enable HA on the cluster.

Before you enable a vSphere Supervisor on a vSphere cluster, you must prepare your environment to meet the specific networking, storage, and infrastructure requirements. See the *Installing and Configuring vSphere IaaS Control Plane* documentation.

For more information about how to configure the storage settings to meet the requirements of vSphere Supervisor, see Creating Storage Policies for .

For more information about how to configure the networking settings for Supervisors that are configured with the VMware NSX-T™ Data Center as the networking stack, see Configuring for .

Starting with vSphere 7.0 Update 1, you can enable a Supervisor with vSphere networking or NSX-T Data Center, to provide connectivity between control planes, services, and workloads. A Supervisor that is configured with vSphere networking uses a vSphere Distributed Switch to provide connectivity to Kubernetes workloads and control planes. The cluster also requires a third-party load balancer that provides connectivity to DevOps users and external services. You can install in your vSphere environment the HAProxy load balancer implementation that VMware provides. See Configuring the vSphere Networking Stack for and Installing and Configuring the HAProxy Load Balancer.

Staring with vSphere 7.0 Update 2, if you are using vSphere networking, you can use the VMware NSX®Advanced Load Balancer™ to support TKG clusters provisioned by the TKG. See Using the with vSphere Networking.

Persistent Storage in vSphere Supervisor

Some Kubernetes workloads use persistent storage to store data permanently. vSphere Supervisor integrates with Cloud Native Storage (CNS) to provision persistent storage.

To understand how vSphere Supervisor uses persistent storage, refer to the *vSphere IaaS Control Plane Concepts and Planning* and *vSphere Storage* documentations.

Creating Storage Policies for vSphere Supervisor

Before you enable vSphere Supervisor, you must set up the storage to provision the Kubernetes infrastructure. You achieve this task by creating storage policies to be used in the Supervisor and namespaces.

To automate the creation of a tag-based storage policy, use the VMware® vSphere Management SDK. For more information about how to create a tag-based storage policy through the Web Services API, see the *VMware Storage Policy SDK Programming Guide* and *vSphere Web Services SDK Programming Guide* documentations.

Optionally, you can use the vSphere AutomationAPIs to create and add a tag to the datastore. See the vSphere Tag Service chapter. Currently, you can create a tag-based storage policy only through the Web Services APIs.

Use the vSphere AutomationAPIs to retrieve the default storage policy of a specific datastore by . You can also retrieve commonly used information about the storage policies available in the vCenter Server instance

You can use the storage policies retrieved through the vSphere Automation APIs to perform the following tasks:

- Assign the storage policies to the Supervisor. The storage policies set within the Supervisor enable specification ensure that the Supervisor control plane, the ephemeral disks of all vSphere Pods, and the container images are placed on the datastores that the policies represent. See Configuring for .
- Assign the storage policies to the vSphere Namespace. The storage policies associated with a namespace determine which datastores the namespace can access and use for persistent volumes for the vSphere Pod and the pods inside a TKG cluster. See Create a .

Enabling ReadWriteMany Support

You can enable the ReadWriteMany support in vSphere Supervisor and allow multiple pods and applications to mount simultaneously a single persistent volume.

In vSphere 7.0 Update 3, only TKG clusters support persistent volumes in ReadWriteMany mode. When you enable file volume support for vSphere Supervisor, be aware of the potential security weaknesses:

- The volumes are mounted without encryption. The unencrypted data might be accessed while the data transits the network.
- Access Control List (ACL) is used for the file shares to isolate file share access within a supervisor namespace. It might have risk of IP spoofing.

Follow these guidelines for networking:

- Make sure the vSAN File Services is routable from the Workload network and there is no NAT between the Workload network and vSAN File Services IP addresses.
- Use common DNS server for vSAN File Services and the vSphere cluster.
- If your vSphere Supervisor has NSX networking, use the SNAT IP of the Supervisor namespace and the SNAT IP of the TKG cluster for ACL configuration.
- If you have vSphere Supervisor with vSphere Distributed Switch (VDS) networking, use the TKG cluster VM IP or the IP of the Supervisor namespace for ACL configuration.

Before you activate the file volume support on a Supervisor, you must set up a vSAN cluster with enabled vSAN File Service. To configure a vSAN cluster with enabled vSAN File Service in the vSphere Client, see the *Configure File Services* topic in the *Administering VMware vSAN* documentation. For more information about how to programmatically achieve this task, see the *vSAN SDKs Programming Guide* documentation.

You activate the ReadWriteMany support on a cluster when you enable vSphere Supervisor on it, or reconfigure an existing Supervisor. See , , and Reconfiguring a . Pass the list of vSAN clusters to be used for provisioning file volumes by using. Currently, you can use only the current vSphere cluster for provisioning file volumes if it is a vSAN cluster with enabled vSAN File Service.

To deactivate the persistent volumes support on a Supervisor, pass an empty list when you set the Cloud Native Storage persistent storage support for the cluster. After that existing ReadWriteMany persistent volumes provisioned in the cluster remain unaffected and usable.

Supervisor Networking

You can enable a Supervisor with vSphere networking or NSX to provision connectivity to Kubernetes control planes, services, and workloads.

A Supervisor that uses the vSphere networking stack is backed by a vSphere Distributed Switch and requires a load balancer to provide connectivity to DevOps users and external services. The Avi Load Balancer and the HAProxy load balancers are supported for vSphere 7.0 Update 2.

A Supervisor that is configured with NSX, uses the software-based networks of the solution and an Key definition for "edge_short" not found in the DITA map. load balancer to provide connectivity to external services and DevOps users.

Configuring NSX for vSphere Supervisor

vSphere Supervisor requires specific networking configuration to allow you to connect to the Supervisors, vSphere Namespaces, and all objects that run inside the namespaces.

Follow the instructions for installing and configuring the NSX for managing Kubernetes workloads documented in the *Installing and Configuring vSphere IaaS Control Plane* guide.

First, you need to create a vSphere Distributed Switch and a distributed port group for each Key definition for "edge_short" not found in the DITA map. uplink. To automate this step, use the Web Services APIs as described in the *vSphere Web Services SDK Programming Guide*. Then, you can use the NSX REST APIs to add a compute manager, create transport zones, and perform other steps required for configuring the NSX for vSphere Supervisor.

Configuring the vSphere Networking Stack for vSphere Supervisor

To configure a Supervisor with the vSphere networking stack, you must connect all hosts from the cluster to a vSphere Distributed Switch. Depending on your topology, you must create one or more distributed port groups on the switch and configure them as workload networks to the vSphere Namespaces on the cluster.

Workload networks provide connectivity to the nodes of TKG clusters and to the Supervisor control planes. The workload network that provides connectivity to Supervisor control planes is called primary workload network. Each Supervisor must have one primary workload network represented by a distributed port group.

The Supervisor control planes on the cluster use three IP addresses from the IP address range that is assigned to the primary workload network. Each node of a TKG cluster has a separate IP address assigned from the address range of the workload network that is configured with the namespace where the TKG cluster runs.

To create a vSphere Distributed Switch and port groups for configuring the vSphere networking stack of a Supervisor, you can use the vSphere Web Services APIs as described in the *vSphere Web Services SDK Programming Guide* documentation. When you create a distributed virtual switch, vCenter Server automatically creates one distributed virtual port group. You can use this port group as the primary workload network and use it to handle the traffic for the Supervisor control planes. Then you can create as many distributed port groups for the workload networks as your topology requires. For a topology with one isolated workload network, create one distributed port group that you will use as a network for all namespaces on the Supervisor. For a topology with isolated networks for each vSphere Namespace, create the same number of distributed port groups as the number of namespaces.

To list all workload networks available for a Supervisor and retrieve information about the configuration of a specific workload network, use the `Networks` service from the vSphere AutomationAPIs. To associate a vSphere Distributed port group to a workload network, set the necessary information through the

Installing and Configuring the HAProxy Load Balancer

You can use the vSphere Automation APIs to customize the HAProxy control plane VM after you install the HAProxy in your vSphere Supervisor environment.

If you use the vSphere networking stack in your vSphere Supervisor environment, you need to supply your own load balancer. You can use the open source implementation of the HAProxy load balancer that VMware provides.

For more information about the prerequisites for installation and the deployment procedure through the vSphere Client, see the *Installing and Configuring vSphere IaaS Control Plane* documentation.

You can use the vSphere AutomationAPIs to install and configure the HAProxy load balancer. You can download the latest version of the HAProxy OVA file from the VMware-HAProxy site to a content library item. For more information about how to achieve this task, see Upload an OVF or OVA Package from a Local File System to a Library Item . Then you can create a new VM from the OVA template in the content library as described in Deploy a Virtual Machine or vApp from an OVF Template in a Content Library.

Parameter	Description
	A list of `Server`s that represent the endpoints for configuring the HAProxy load balancers. Each endpoint is

Table continued on next page

Continued from previous page

Parameter	Description
	described by a load balancer IP address and a Data Plane API management port. Each endpoint must be described with the port on the HAProxy VM on which the Data Plane API service listens. The Data Plane API service controls the HAProxy server and runs inside the HAProxy VM. The default port is `5556`. Port `22` is reserved for SSH.
	The administrator user name that is configured with the HAProxy OVA file and is used to authenticate to the HAProxy Data Plane API server.
	The password for the administrator user name.
	The certificate in PEM format that is signed or is a trusted root of the server certificate that the Data Plane API server presents.

Using the Avi Load Balancer with vSphere Networking

If you use the vSphere networking stack for workload management, you can install and configure the Avi Load Balancer, also known as Avi Load Balancer, Essentials Edition, to support the TKG clusters.

For more information about how to install and configure the Avi Load Balancerthrough the vSphere Client, see the *Installing and Configuring vSphere IaaS Control Plane* documentation.

You can use the vSphere Automation APIs to deploy the Avi Controller on your vSphere Management network. You can upload the latest version of the Avi Load Balancer to a library item from your local file system or from a URL. For more information about how to achieve this task, see Upload an OVF or OVA Package from a Local File System to a Library Item . Then you can deploy the Controller VM on your vSphere Management network from the OVA template in the content library as described in Deploy a Virtual Machine or vApp from an OVF Template in a Content Library.

Parameter	Description
	The address of the Avi Controller that is used to configure virtual services.
	The administrator user name that is used for accessing the Controller VM of the Avi Load Balancer.
	The password for the administrator user name.
	The certificate in PEM format that is used by the Controller. You can use the certificate that you assigned during the configuration of the Avi Load Balancer.

Enable vSphere Supervisor on a Cluster with NSX as the Networking Stack

Through the vSphere Automation APIs, you can enable a vSphere cluster for managing Kubernetes workloads. A cluster configured with NSX supports running vSphere Pod and TKG clusters.

- Verify that your environment meets the system requirements for enabling vSphere Supervisor on the cluster. For more information about the requirements, see the *vSphere IaaS Control Plane Concepts and Planning* documentation.
- Verify that the NSX is installed and configured. See Configuring for .
- Create storage policies for the placement of pod ephemeral disks, container images, and Supervisor control plane cache.
- Verify that DRS is enabled in fully automated mode and HA is also enabled on the cluster.
- Configure shared storage for the cluster. Shared storage is required for vSphere DRS, HA, and storing persistent volumes of containers.
- Verify that the user who you use to access the vSphere Automation services has the **Modify cluster-wide configuration** privilege on the cluster.
- Create a subscribed content library on the vCenter Server system to accommodate the VM image that is used for creating the nodes of the TKG clusters.

To enable a vSphere cluster for Kubernetes workload management, you use the services under the `namespace_management` package.

1. Retrieve the IDs of the tag-based storage policies that you configured for vSphere Supervisor.

   Use the `GET https://<vcenter_ip_address_or_fqdn>/api/vcenter/storage/policies` request to retrieve a list of all storage policies and then filter the policies to get the IDs of the policies that you configured for the Supervisor.

2. Retrieve the IDs of the vSphere Distributed Switch and the Key definition for "edge_short" not found in the DITA map. cluster that you created when configuring the NSX for vSphere Supervisor.

   Use the `POST https://<vcenter_ip_address_or_fqdn>/api/vcenter/namespace-management/networks/nsx/distributed-switches?action=check_compatibility` request to list all vSphere Distributed Switches associated with the specific vSphere cluster and then retrieve the ID of the Distributed Switch that you configured to handle overlay networking for the Supervisor.
   Use the `POST https://<vcenter_ip_address_or_fqdn>/api/vcenter/namespace-management/networks/nsx/edges?action=check_compatibility` request to retrieve a list of the created Key definition for "edge_short" not found in the DITA map. clusters for the specific vSphere cluster and associated with the specific vSphere Distributed Switch. Retrieve the ID of the Key definition for "edge_short" not found in the DITA map. cluster that has the tier-0 gateway that you want to use for the namespaces networking.

3. Retrieve the ID of the port group for the management network that you configured for the management traffic.

   To list the visible networks available on the vCenter Server instance that match some criteria, use the `GET https://<vcenter_ip_address_or_fqdn>/api/vcenter/namespace-management/clusters/<cluster_id>/networks` request and then retrieve the ID of the management network you previously configured.

4. Create a `Clusters.EnableSpec` JSON object and define the parameters of the Supervisor that you want to create.

   You must specify the following required parameters of the enable specification:

   - Storage policies settings and file volume support. The storage policy you set for each of the following parameters ensures that the respective object is placed on the datastore referenced in the storage policy. You can use the same or different storage policy for the different inventory objects.

Property	Description
`ephemeral_storage_policy`	Specify the ID of the storage policy that you created to control the storage placement of the vSphere Pods .
`image_storage`	Set the specification of the storage policy that you created to control the placement of the cache of container images.
`master_storage_policy`	Specify the ID of the storage policy that you created to control the placement of the Supervisor control plane cache.

Optionally, you can activate the file volume support by using `cns_file_config`. See Enabling ReadWriteMany Support.

- Management network settings. Configure the management traffic settings for the Supervisor control plane.

Property	Description
`network_provider`	Specify the networking stack that must be used when the Supervisor is created. To use the NSX as the network solution for the cluster, set `NSXT_CONTAINER_PLUGIN`.
`master_management_network`	Enter the cluster network specification for the Supervisor control plane. You must enter values for the following required properties: • `network`- Use the management network ID retrieved in Step 3. • `mode` - Set `STATICRANGE` or `DHCP` for the IPv4 address assignment mode. The `DHCP` mode allows an IPv4 address to be automatically assigned to the Supervisor control plane by a DHCP server. You must also set the floating IP address used by the HA primary cluster by using `floating_IP`. Use the DHCP mode only for test purposes. The `STATICRANGE` mode, allows the Supervisor control plane to have a stable IPv4 address. You can use it in a production environment. • `address_range`- Optionally, you can configure the IPv4 addresses range for one or more interfaces of the management network. Specify the following settings: – The starting IP address that must be used for reserving consecutive IP addresses for the Supervisor control plane. Use up to 5 consecutive IP addresses. – The number of IP addresses in the range. – The IP address of the gateway associated with the specified range.

Table continued on next page

Continued from previous page

Property	Description
	– The subnet mask to be used for the management network.
`master_DNS`	Enter a list of the DNS server addresses that must be used from the Supervisor control plane. If your vCenter Server instance is registered with an FQDN, you must enter the IP addresses of the DNS servers that you use with the vSphere environment so that the FQDN is resolvable in the Supervisor. The list of DNS addresses must be specified in the order of preference.
`master_DNS_search_domains`	Set a list of domain names that DNS searches when looking up for a host name in the Kubernetes API server. Order the domains in the list by preference.
`master_NTP_servers`	Specify a list of IP addresses or DNS names of the NTP server that you use in your environment, if any. Make sure that you configure the same NTP servers for the vCenter Server instance, all hosts in the cluster, the NSX, and vSphere Supervisor. If you do not set an NTP server, VMware Tools time synchronization is enabled.

- Workload network settings. Configure the settings for the networks for the namespaces. The namespace network settings provide connectivity to vSphere Pods and namespaces created in the Supervisor.

Property	Description
`ncp_cluster_network_spec`	Set the specification for the Supervisor configured with the NSX networking stack. Specify the following cluster networking configuration parameters for `NCPClusterNetworkEnableSpec`: • `cluster_distributed_switch` - The vSphere Distributed Switch that handles overlay networking for the Supervisor. • `nsx_edge_cluster` - The Key definition for "edge_short" not found in the DITA map. cluster that has tier-0 gateway that you want to use for namespace networking. • `nsx_tier0_gateway`- The tier-0 gateway that is associated with the cluster tier-1gateway. You can retrieve a list of `NSXTier0Gateway` objects associated with a particular vSphere Distributed Switch and determine the ID of the tier-0 gateway you want to set. • `namespace_subnet_prefix` - The subnet prefix that defines the size of the subnet reserved for namespaces segments. Default is 28.

Table continued on next page

Continued from previous page

Property	Description

	• `routed_mode` - The NAT mode of the workload network. If set to `false`:
	  • The IP addresses of the workloads are directly accessible from outside the tier-o gateway and you do not need to configure the egress CIDRs.
	  • File Volume storage is not supported.
	Default is `true`.
	• `egress_cidrs` - The external CIDR blocks from which the NSX Manager assigns IP addresses used for performing source NAT (SNAT) from internal vSphere Pods IP addresses to external IP addresses. Only one egress IP address is assigned for each namespace in the Supervisor. These IP ranges must not overlap with the IP ranges of the vSphere Pods, ingress, Kubernetes services, or other services running in the data center.
	• `ingress_cidrs`- The external CIDR blocks from which the ingress IP range for the Kubernetes services is determined. These IP ranges are used for load balancer services and Kubernetes ingress. All Kubernetes ingress services in the same namespace share a common IP address. Each load balancer service is assigned a unique IP address. The ingress IP ranges must not overlap with the IP ranges of the vSphere Pods, egress, Kubernetes services, or other services running in the data center.
	• `pod_cidrs` - The internal CIDR blocks from which the IP ranges for vSphere Pods are determined. The IP ranges must not overlap with the IP ranges of the ingress, egress, Kubernetes services, or other services running in the data center. All vSphere Pods CIDR blocks must be of at least /23 subnet size.

`worker_DNS`	Set a list of the IP addresses of the DNS servers that must be used on the worker nodes. Use different DNS servers than the ones you set for the Supervisor control plane.
`service_cidr`	Specify the CIDR block from which the IP addresses for Kubernetes services are allocated. The IP range must not overlap with the ranges of the vSphere Pods , ingress, egress, or other services running in the data center. For the Kubernetes services and the vSphere Pods, you can use the default values which are based on the cluster size that you specify.

- Supervisor size. You must set a size to the Supervisor which affects the resources allocated to the Kubernetes infrastructure. The cluster size also determines default maximum values for the IP addresses

ranges for the vSphere Pods and Kubernetes services running in the cluster. You can use the `GET https://<vcenter_ip_address_or_fqdn>/api/vcenter/namespace-management/cluster-size-info` calls to retrieve information about the default values associated with each cluster size.
- Optional. Associate the Supervisor with the subscribed content library that you created for provisioning TKG clusters. See Creating, Securing, and Synchronizing Content Libraries for Releases.
  To set the library, use `default_kubernetes_service_content_library` and pass the subscribed content library ID.

5. Enable vSphere Supervisor on a specific cluster by passing the cluster enable specification to the `Clusters` service.

A task runs on vCenter Server for turning the cluster into a Supervisor. Once the task completes, Kubernetes control plane nodes are created on the hosts that are part of the cluster enabled with vSphere Supervisor. Now you can create vSphere Namespaces.

Create and configure namespaces on the Supervisor. See Create a .

Enable vSphere Supervisor on a Cluster with the vSphere Networking Stack

Starting with vSphere 7.0 Update 1, you can select between creating a Supervisor with the vSphere networking stack or with NSX as the networking solution. A Supervisor that is configured with the vSphere networking stack only supports TKG clusters. vSphere Pods are not supported.

- Verify that your environment meets the system requirements for enabling vSphere Supervisor on the cluster. For more information about the requirements, see the documentation.
- Verify that DRS is enabled in fully automated mode and HA is also enabled on the cluster.
- Configure shared storage for the cluster. Shared storage is required for vSphere DRS, HA, and storing persistent volumes of containers.
- Create storage policies for the placement of Kubernetes control planes.
- Create a subscribed content library on the vCenter Server system to accommodate the VM image that is used for creating nodes of TKG clusters. See Creating, Securing, and Synchronizing Content Libraries for Releases.
- Add all hosts from the cluster to a vSphere Distributed Switch and create port groups for workload networks. See Configuring the vSphere Networking Stack for .
- Configure an HAProxy load balancer instance that is routable to the vSphere Distributed Switch that is connected to the hosts from the vSphere cluster.
- Verify that the user who you use to access the vSphere Automation services has the **Namespaces.Manage** privilege on the cluster.

To enable a cluster configured with the vSphere networking stack for Kubernetes workloads management, you must use the services under the `namespace_management` package.

1. Retrieve the ID of the cluster which hosts were added to the vSphere Distributed Switch.

   Use the `GET https://<vcenter_ip_address_or_fqdn>/api/vcenter/namespace-management/cluster-compatibility` request to filter the clusters by using their network providers. To retrieve a list of all clusters in the vCenter Server system which are configured with the vSphere networking stack, set the network provider in the filter specification to `VSPHERE_NETWORK`.

2. Retrieve the IDs of the tag-based storage policies that you configured for vSphere Supervisor.

   Use the `GET https://<vcenter_ip_address_or_fqdn>/api/vcenter/storage/policies` request to retrieve a list of all storage policies and then filter the policies to get the IDs of the policies that you configured for the Supervisor.

3. Retrieve the ID of the port group for the management network that you configured for the management traffic.

To list the visible networks available on the vCenter Server instance that match some criteria and then retrieve the ID of the management network you previously configured, use the `GET https://<vcenter_ip_address_or_fqdn>/api/vcenter/namespace-management/clusters/<cluster_id>/networks` request.

4. Create a Supervisor enable specification and define the parameters of the Supervisor that you want to enable.

   You must specify the following required parameters of the enable specification:

   - Supervisor size. You must set a size to the Supervisor which affects the resources allocated to the Kubernetes infrastructure. The cluster size also determines default maximum values for the IP addresses ranges for the vSphere Pod and Kubernetes services running in the cluster. You can use the to retrieve information about the default values associated with each cluster size.
   - Storage policy settings and file volume support. To specify the ID of the storage policy that you created to control the placement of the Supervisor control plane cache, use. Optionally, you can activate the file volume support by using . See Enabling ReadWriteMany Support.
   - Load balancer. To specify the user-provisioned load balancer configuration for the cluster, use the `load_balancer_config_spec` parameter of the enable specification. You must specify the following parameters of the `LoadBalancersTypes.ConfigSpec` specification:

Parameter	Description
`id`	A user-friendly name of the load balancer. The name must be an alphanumeric string with a maximum length of 63 characters which is unique across the namespaces in the vCenter Server instance.
`provider`	The type of the load balancer that you want to use. In vSphere 7.0 Update 2, you can choose between the HAProxy load balancer and the Avi Load Balancer. Pass as a value to this parameter one of the following constants: `HA_PROXY` or `AVI`.
`address_ranges`	The IP address ranges in CIDR format from which HAProxy allocates the IP addresses for the virtual servers. You must provide at least one IP range which is reserved by HAProxy. The CIDR range specified with this parameter must not overlap with the IPs allocated for the Kubernetes control planes and workloads. The IP range that you configure must be on a separate subnet.
`ha_proxy_config_create_spec`	The HAProxy runtime configuration. See Installing and Configuring the HAProxy Load Balancer.
`avi_config_create_spec`	The Avi Load Balancer configuration. See Using the with vSphere Networking.

   - Management network settings. Configure the network parameters for the Kubernetes control planes.

Parameter	Description
`network_provider`	Specify the networking stack that must be used when the Supervisor is created. To use the vSphere

Table continued on next page

Continued from previous page

Parameter	Description
	network as the solution for the cluster, select `VSPHERE_NETWORK`.
`master_management_network`	Enter the cluster network specification for the Supervisor control plane. You must enter values for the following required properties: • `network`- Use the management network ID retrieved in Step 3. • `mode`- Set `STATICRANGE` or `DHCP` for the IPv4 address assignment mode. The `DHCP` mode allows an IPv4 address to be automatically assigned to the Supervisor control plane by a DHCP server. You must also set the floating IP address used by the HA primary cluster by using `floating_IP`. Use the DHCP mode only for test purposes. The `STATICRANGE` mode, allows the Supervisor control plane to have a stable IPv4 address and can be used in a production environment.
`master_DNS`	Enter a list of the DNS server addresses that must be used from the Supervisor control plane. If your vCenter Server instance is registered with an FQDN, you must enter the IP addresses of the DNS servers that you use with the vSphere environment so that the FQDN is resolvable in the Supervisor. The list of DNS addresses must be specified in the order of preference.
`master_DNS_search_domains`	Set a list of domain names that DNS searches inside the Kubernetes control plane nodes, so that the DNS server can resolve them. Order the domains in the list by preference.
`master_NTP_servers`	Specify a list of IP addresses or DNS names of the NTP server that you use in your environment, if any. Make sure that you configure the same NTP servers for the vCenter Server instance, all hosts in the cluster, and vSphere Supervisor. If you do not set an NTP server, VMware Tools time synchronization is enabled.

- Workload network settings. Configure the settings for the network that will handle the networking traffic for Kubernetes workloads running on the Supervisor.

Parameter	Description
`service_cidr`	Specify the CIDR block from which the IP addresses for Kubernetes services are allocated. The IP range must not overlap with the ranges of the vSphere Pods , ingress, egress, or other services running in the data center.

Table continued on next page

Continued from previous page

Parameter	Description
	For the Kubernetes services and the vSphere Pods, you can use the default values which are based on the cluster size that you specify.
`workload_networks_spec`	Enter the workload networks specifications for the cluster. To configure the primary workload network that is used to expose the Supervisor control plane to DevOps and other workloads, create a `NetworksType.CreateSpec` / `NetworksTypes.CreateSpec` instance. Enter the following parameters of the vSphere Distributed Switch: • `network`. The name of the vSphere Distributed Switch that is associated with the hosts in the cluster. The name must be a unique alphanumeric string that does not exceed 63 characters. • `network_provider`. Pass `VSPHERE_NETWORK` as value to this parameter. • `vsphere_network`. Optionally, you can create a `vsphere_DVPG_network_create_spec` instance to describe the configuration of the namespace network backed by the vSphere Distributed port group. You must define the following parameters for the vSphere Distributed port group specification: – `portgroup`. Specify the port group that serves as the primary network to the Supervisor. – `address_ranges`. Set the IP range for allocating IP addresses for the Kubernetes control planes and workloads. You must use unique IP ranges for each workload network. – `gateway`. Set the gateway for the primary network. – `subnet_mask`. Specify the subnet mask of the network.

- Content library settings. Add the subscribed content library that contains the VM images for deploying the nodes of TKG clusters. See Creating, Securing, and Synchronizing Content Libraries for Releases.
  To set the library, use `default_kubernetes_service_content_library` and pass the subscribed content library ID.

5. Enable the Supervisor by passing the enable specification to the `Clusters` service.

A task runs on vCenter Server for enabling vSphere Supervisor on the cluster. Once the task completes, three Kubernetes control planes are created on the hosts that are part of the cluster.

Create and configure namespaces on the Supervisor.

Upgrading a Supervisor

You can use the vSphere SupervisorAPIs to upgrade a single or a group of clusters to a specific version.

vSphere Supervisor supports rolling upgrades through the vSphere AutomationAPIs for Supervisors and for the infrastructure supporting these clusters. This model ensures that there is minimal downtime for the cluster workloads during the upgrade process.

To retrieve a list of all available vSphere Supervisor upgrade versions for a specific vCenter Server system, use the Cluster Available Versions service. You can get information about the release version, name, description, release date, and release notes for each available upgrade.

You must use the Software Clusters service for upgrading a Supervisor. You can retrieve upgrade information about all Supervisors enabled on a vCenter Server system by using the . You receive a list of basic upgrade-related information for each cluster, such as the current software version, the date of the last successful upgrade, the upgrade status of the cluster, and so on. In case some of the clusters are in the process of upgrading, you can retrieve also information about their desired upgrade version. If you want to view a more detailed upgrade-related information about a cluster, you must use the .

After you view the details about the upgrade versions that you can apply on a single or multiple Supervisors, you can create upgrade specifications that define the versions you want to upgrade to. When you upgrade a batch of Supervisor and for some reason one of the clusters fails to upgrade, you receive information about the pre-check exceptions that led to that cluster upgrade failure.

Monitoring the Enable and Upgrade Supervisor Operations

When you run the Supervisor enable and upgrade operations, the status of the tasks is not returned. Since these operations might be time-consuming but critical when automating second- and third-party products with vSphere Supervisor, you can write a logic to track the task status.

To monitor the status of the enable and upgrade Supervisor operations, use the and query the Kubernetes and configuration status of the cluster. Track the status every two minutes or so, until you receive `READY` for and `RUNNING` for . These statuses indicate that the Supervisor reached the desired configuration status and is ready for running Kubernetes workloads.

Reconfiguring a Supervisor

You can change some or all the predefined settings of a Supervisor through the vSphere AutomationAPIs.

To update only some of the Supervisor settings, . To reconfigure entirely the Supervisor, .

The settings you can configure with the `UpdateSpec` and `SetSpec` specifications are the same as the ones that you used for enabling the Supervisor. For example, you can change the storage settings on the Supervisor. Note that the changes that you make to the storage settings after the initial cluster configuration, apply only to the newly created Supervisor control planes.

Deactivating a Supervisor

You can programmatically deactivate vSphere Supervisor on a vSphere cluster by using the vSphere AutomationAPIs.

When you deactivate a Supervisor, the vSphere Kubernetes Service forcefully deletes from the cluster all objects and configurations part of the Kubernetes infrastructure. To deactivate Kubernetes workloads on a cluster,

Content Libraries in vSphere Supervisor

vSphere Supervisor uses content libraries as centralized repositories for templates, VM images, TKG release distributions, and other files related to their deployment.

Creating, Securing, and Synchronizing Content Libraries for TKG Releases

VMware Tanzu distributes Kubernetes software versions as TKG releases. To obtain and use these releases on your TKG clusters, you create subscribed or local content libraries.

A TKG release provides the VMware Kubernetes distribution which can be used with TKG clusters. Each TKG release is distributed as an OVA package. The TKG uses the OVA package to deploy the virtual machine nodes for TKG clusters.

A TKG release is supported on Photon OS. The virtual machine nodes that are built from the OVA package have a 16 GB disk size. You specify the CPU and RAM resource reservations when you use a virtual machine class to size the TKG cluster.

Depending on your need for synchronization frequency and on the access to the published content libraries storing the TKG releases, you can use two approaches for storing TKG releases.

> **NOTE**
> Starting with vSphere 7.0 Update 3, you can protect your content library by a security policy. In such case, make sure that all library items are compliant. If a protected library includes a mix of compliant and non-compliant library items, DevOps engineers are not able to retrieve the list of VM images provided with the library.

Automated Synchronization of TKG Releases

VMware publishes a content library that contains the latest VMware distributions of Kubernetes as an OVA package. If you want to provision TKG clusters, you can create a subscribed content library on the vCenter Server instance where vSphere Supervisor is enabled. When configuring the content library subscription, use the following subscription URL of the publisher : https://wp-content.vmware.com/v2/latest/lib.json. For more information about how to create a subscribed content library, see .

When you create the subscription, you configure the synchronization mechanism for downloading the content of the published library. You can select between on demand and automatic download of the virtual machine image for the TKG cluster nodes. If you choose to synchronize the subscribed library on demand, only the metadata for the library content is updated and as a result storage space is saved. This approach is an important consideration as more images containing different Kubernetes versions are published. However, the first time you decide to use a new virtual machine image version, you have to wait for it to download.

Starting with vSphere 7.0 Update3, you can secure a subscribed content library. The Content Library service verifies the library signing certificate during the synchronization process. If the certificate verification fails, only the library metadata is synchronized and the library content is not downloaded. For more information how to apply a security policy when you update a subscribed content library, see .

You associate the subscribed content library with the Supervisor on which you want to create a TKG cluster, when you first enable vSphere Supervisor on a cluster. See .

The size of the content library can grow over time as new Kubernetes versions and images are published. If the underlying storage runs out of space, you will need to move to a new subscribed content library. After you create a new subscribed content library that has sufficient capacity for the target cluster, update the library association of the Supervisor. See Reconfiguring a .

Manual Synchronization of TKG Releases

In an air-gapped network environment, you can use the storing functionality provided by a local content library for the needed TKG releases. You must first create a local content library, then download the OVA package for each TKG release that you want to import to the library. See .

Starting with vSphere 7.0 Update3, you can secure a local content library. The Content Library service verifies the library signing certificate during the synchronization process. If the certificate verification fails, only the library metadata is synchronized and the library content is not downloaded. For more information how to apply a security policy when you update a local content library, see .

You can find the latest versions of the Kubernetes distribution by navigating to the https://wp-content.vmware.com/v2/latest URL. You must download the `photon-ova.ovf` and `photon-ova-disk1.vmdk` for each distribution you want and then upload these files from your local file system to your local content library. See Upload an OVF or OVA Package from a Local File System to a Library Item .

> **NOTE**
> Make sure that you use as a name for each library item the Photon image version and the Kubernetes version from the directory where you downloaded the files. For example: `photon-3-k8s-v1.20.2---vmware.1-tkg.1.1d4f79a`.

Creating and Managing Content Libraries for VM Provisioning in vSphere Supervisor

To provision new virtual machines in a vSphere Supervisor environment, the DevOps engineers rely on VM templates and images. Your role is to make sure the DevOps engineers have access to these VM templates and images by using the Content Library service.

You can create a local content library and populate it with VM templates in OVF or OVA file format, or other types of files. For more information and a sample of how to create a local content library, see .

You can also create a subscription to download the content of a published local content library as described in the following topic: .

Starting with vSphere 7.0 Update3, you can secure the content library. The Content Library service verifies the library signing certificate during the synchronization process. If the certificate verification fails, only the library metadata is synchronized and the library content is not downloaded. For more information how to apply a security policy when you update a local or subscribed content library, see .

After you create the content library, you must populate it with content either from your local file system or from a Web server. You must use only the VM images available on the VMware Cloud Marketplace web site. For example, download or subscribe to VM Service Image for Ubuntu if you want to enable a DevOps engineer to deploy a VM using this image. For more information about the available ways to populate a content library with content, see .

You must give the DevOps engineers access to the VM templates stored in the content libraries, so that they can use these templates to provision VMs through the VM Service functionality. To give access, you must associate one or more content libraries to the namespace where the VM Service is present. See Associating a Content Library with a Namespace and Virtual Machines in .

Associating a Content Library with a Namespace

You must give access to a source of VM templates, so that the DevOps engineers can use them to provision VMs in a self-service manner. To give access, you associate a content library with VM templates to the namespace used by the DevOps engineers.

You can add multiple content libraries to a namespace that has the VM Service enabled or the same content library to several namespaces. You associate a content library to a namespace when you create a new namespace, update or reconfigure an existing one.

To make the VM Service aware of the content libraries in your environment that the DevOps engineers can use to self-service VMs, you must . The instance contains a list of content libraries that will be used by the VM Service. You can specify this list .

You can also associate one or more VM classes with the namespace. See Associating a VM Class with a .

Managing Namespaces on a Supervisor

You can use the vSphere Automation APIs to create namespaces on a Supervisor and configure them with resource limits and permissions for the DevOps users.

To create and configure a namespace, use the `Instances` service from the `namespaces` package. You can configure the access control to the objects in a namespace by using the `Access` service.

Create a vSphere Namespace

You can use the vSphere Supervisor automation APIs to create namespaces on a Supervisor. You can set resource quotas, storage, as well as permissions for the DevOps users.

- Enable vSphere Supervisor on a vSphere cluster.
- Create users and groups for the DevOps engineers who will use the namespace. For more information about how to create users and groups through the Web Services APIs, see the *vSphere Web Services SDK Programming Guide*.
- Create storage policies for persistent storage used by the vSphere Pods and the pods inside a TKG cluster.
- Create VM Classes and content libraries for DevOps provisioned VMs. See Create a VM Class in and Creating and Managing Content Libraries for VM Provisioning in .
- Required privileges on the Supervisor:
  - **Namespaces › Modify cluster-wide configuration**
  - **Namespaces › Modify namespace configuration**
  - **Virtual Machine Classes › Manage Virtual Machine Classes**

1. Retrieve the Supervisor ID by filtering the clusters available in the vCenter Server system.
2. Retrieve the ID of the storage policy that you configured for placement of the persistent volumes from vSphere Pods and TKG clusters.
3. Configure the access control to the objects in the namespace.

	Description
	Set the domain name of the vCenter Server system on which the namespace is created.
	Set the type of the user accounts that are associated with the specific role for the namespace. You must select between the `USER` and `GROUP` options.
	Set the name of the user or group that have permissions to access the namespace objects.
	Set the role that is associated with the predefined set of privileges that you want to grant the specific user or group. You can select between the `EDIT`, `VIEW` and `OWNER` roles. The owner role is introduced in vSphere 7.0 Update 2a. When a DevOps engineer creates a namespace in a

Table continued on next page

Continued from previous page

	Description
	self-service manner, the Namespace Self-Service grants the owner role to the namespace creator. See Self-Service Namespace Management.

4.

The namespace specification can contain the following information:

	Description
	Set the ID of the Supervisor on which the namespace is created.
	Set a name of the namespace following the DNS label standard defined in RFC 1123. The name must be unique across all namespaces in the current vCenter Server system.
	Optional. You can set the workload networks used by the vSphere Namespace. Pass `null` as a value of this parameter, if the Supervisor is configured to use NSX as networking solution. The workload networking support for such namespaces is provisioned by NSX. If the Supervisor uses the vSphere networking stack, pass the workload network to be associated with the namespace. If you pass `null` as a value of this parameter, the vSphere Namespaces on the cluster are automatically associated with the cluster primary workload network. See Configuring the vSphere Networking Stack for .
	Optional. You can set a description of the namespace.
	Optional. You can set the access control that is associated with the namespace in Step 3.
	Optional. You can set the amount of storage dedicated to each storage policy associated with the namespace and the maximum amount of storage that is used by the namespace. Use the `StorageSpec` specification to configure the storage quotas on the namespace.
	Optional. You can set resource limitations to the namespace. You can limit the CPU, memory, the maximum number of pods that can exist on the namespace, and so on.
	Optional. The Namespace Self-Service populates this parameter with information about the DevOps user who created the namespace with `cubectl`. The user name and domain of the namespace creator are stored with this parameter.
	Optional. The VM Service specification for the Dev-Ops provisioned virtual machines.

5. Create a namespace object on the Supervisor by using the namespace create specification.

Share the namespace with DevOps engineers and provide them with the user or group configured for accessing the namespace.

Updating the Namespace Configuration

You can change the whole namespace configuration or only some of the namespace settings.

To change the configuration of an existing namespace, you must have the **Namespaces.Configure** privilege on the Supervisor.

> **NOTE**
> Before deleting a storage policy from vCenter Server or a vSphere Namespace, or changing the storage policy assignment, make sure that no persistent volume claim with the corresponding storage class runs in the namespace. Also, ensure that no TKG cluster is using the storage class.

Configuring the Access to a Namespace

You can use the vSphere Supervisor APIs to grant access permissions to DevOps engineers on the vSphere Namespaces.

Use the `Access` service to retrieve information about the access control of the DevOps engineers on a specific namespace. You can also set up or remove an access control for a specific user or group on a specific namespace, and add another access control on the namespace. You set up each access control to allow a user or group to access a namespace in a specific vCenter Server system. You can grant access to a DevOps engineer to more than one namespace.

You must have the **Namespaces › Configure ›** privilege to grant permissions to a user. You assign the view and edit access role on the namespace for the user or group.

Starting with vSphere 7.0 Update 2a, you can also assign the owner role to a DevOps engineer. These roles allow the user to deploy workloads, share the namespace with other DevOps engineers, and delete it when it is no longer needed.

Self-Service Namespace Management

You can use the vSphere Supervisor automation APIs to create a vSphere Namespace with specific resource quotas, set permissions, and assign storage policies. DevOps engineers can then use the namespace as a template for self-provisioning namespaces on the cluster.

Starting with vSphere 7.0 Update 2a, the Namespace Self-Service feature is available in vSphere Supervisor. The service enables Kubernetes users to create vSphere Namespaces from templates configured through the automation APIs or vSphere Client. To activate the Namespace Self-Service on a cluster, use one of the following options:

- Create a self-service namespace template and then activate the Namespace Self-Service on the cluster.
- Create or update a self-service namespace template simultaneously with activating the Namespace Self-Service on the cluster.

Currently, only one namespace self-service template is allowed per vSphere Namespace. After a DevOps engineer creates a namespace from the template, the namespace can also be deleted through `kubectl`. You can verify whether a namespace is created from a template by retrieving the value of the

To create a template for a self-service namespace, You use as the cluster ID and the namespace template create specification.

You define the following configuration settings and resource limitations of the template:

	Description
	The identifier of the namespace template must be a unique name across all clusters on the vCenter Server instance. The name must be compliant with DNS.
	The resource quotas, such as CPU and memory, that are reserved for the namespace on the vCenter Server instance. The CPU limit is set in MHz and the minimum value is 10 MHz. The memory and the storage limits are set in MiB. For more options to configure resource limits for the namespace, see the `ResourceQuotaOptionsV1` class in the API Reference documentation.
	The amount of storage in MiB utilized for each storage policy that you associate with the namespace. You must specify at least one policy.
	Optional. The networks associated with the namespace. Currently, you can set only one network for the namespace. Pass `null` as argument if the Supervisor is configured with NSX-T Data Center support. If you pass `null` for a namespace template on a cluster configured with a vSphere networking stack, the namespace is automatically associated with the Supervisor management workload network.
	Optional. The permissions that allow DevOps engineers to use the template to self-provision namespaces through `kubectl`. If , only users with the Administrator role can use the template.

Once you have the template created, you can activate the Namespace Self-Service on the cluster If you want to restrict DevOps users to use the namespace template on a cluster, you can deactivate the Namespace Self-Service feature. Then users are able to delete only the namespaces already created from the template.

You can activate the Namespace Self-Service on the cluster after configuring the namespace template by using the `NamespaceSelfService` service. Depending on the availability of a template on the cluster, this method either creates a namespace template or activates the deactivated service and at the same time updates the existing template.

Virtual Machines in vSphere Supervisor

vSphere Supervisor offers the VM Service functionality to enable DevOps engineers to provision and manage VMs on a namespace in a self-service manner. You use the vSphere Supervisor automation APIs to create VM classes that specify the deployment policy and resource reservations of such VMs.

Starting with vSphere 7.0 Update 2a, DevOps engineers can use the VM Service functionality to deploy and run VMs on a namespace through the `kubectl` commands. You can use the vSphere Supervisor automation APIs to manage the two VM Service components: VM classes and content libraries. For more information about managing content libraries in the context of vSphere Supervisor, see Content Libraries in .

You can use the automation APIs to create and manage VM classes. A VM class specification defines the number of CPUs, memory capacity, and resource reservation settings of the desired virtual machine. vSphere Supervisor currently offers twelve ready-to-use VM classes (T-shirt sizes) that are derived from the most popular VMs in Kubernetes. Based on the resource reservation that a VM specification requests, each predefined VM class has two editions: guaranteed and best effort. The guaranteed VM class fully reserves the configured resources. A best effort VM class does not guarantee any resource reservations and allows their overcommitment.

You associate a VM class with a specific namespace to make it available to the DevOps engineers who have access to that namespace. You can assign any number of existing VM classes or create a custom one. Note that VMs deployed by the DevOps engineers through the VM Service can only be managed with the `kubectl` commands. A VM provisioned by DevOps engineers shares the same resources in a namespace as containers.

Use the `VirtualMachineClasses` interface to create and manage a specification of a VM class object. Through these objects you predefine the number of CPUs, memory capacity, and reservation settings. See Create a VM Class in . To make a VM class available to the DevOps engineers for self-service VM deployment, you must associate it with a specific namespace. See Associating a VM Class with a .

Create a VM Class in vSphere Supervisor

You can use the vSphere Automation Kubernetes APIs to create custom VM classes to be used for VM deployment in vSphere Supervisor.

Required privileges:
- **Namespaces › Modify cluster-wide configuration**
- **Namespaces › Modify namespace configuration**
- **Virtual Machine Classes › Manage Virtual Machine Classes**

A VM class specifies the CPU, memory, and resource reservations for a VM. vSphere Supervisor offers several preconfigured VM classes which you can use as is, edit, or delete. You can also create a custom VM class in your vCenter Server instance and it will be available to all Supervisors and the namespaces created in these clusters. Note that even though a VM class is available to all namespaces, a DevOps user can only use the VM classes associated with the namespaces that he/she can access.

1. Create the specification of the VM class object by defining the following options.

Option	Description
	The identifier of the VM class must follow these DNS requirements: • A unique name in the current vCenter Server instance. • An alphanumeric name with maximum 63 characters. • No uppercase letters of spaces. • A dash can be used anywhere except as a first or last character. Note that after a VM class is created, you cannot edit its ID.
	The number of virtual CPUs (vCPUs) configured for a VM that are deployed with this VM class.
	The memory in MB configured for a VM that are deployed with this VM class. The value must be between 4 MB and 24 TB and a multiple of 4.
	Optional. The description of the VM class.
	Optional. The percentage of total available CPU resources reserved for the VM deployed with the VM class. The percentage you specify with this attribute is multiplied by the minimum CPU available among all cluster nodes to get the CPU resources guaranteed by vSphere for a VM. The resulting value is in MHz.

Table continued on next page

Continued from previous page

Option	Description
	Optional. The percentage of available memory that is reserved for a VM deployed with this VM class. The value can be from 0 through 100%.

2. Create the VM class object.

After you create the custom VM class, you can edit its parameters or delete it from your environment. See Editing or Removing a VM Class from Your Environment.
You can make your VM class available to DevOps engineers by associating it with a namespace. See Associating a VM Class with a .

Editing or Removing a VM Class from Your Environment

You can use the automation APIs to edit the configuration of a VM class that you created or a predefined VM classes that vSphere Supervisor offers. When you no longer need an existing VM class, you can remove it from your vCenter Server instance.

Note that editing a VM class specification does not affect the VMs that are already deployed by the DevOps engineers from this class. Only newly deployed VMs will use the reconfigured VM class.

Deleting a VM class results in its removal from all related namespaces. All VMs deployed with this VM class remain unchanged but DevOps engineers can no longer use it.

You can list all VM classes available for a vCenter Server instance You receive a list of all VM classes and a detailed information about each one of them. Based on the detailed information, you can narrow the list and retrieve the IDs of the VM classes that you want to edit or delete.

To edit a VM class configuration, You can edit all VM class attributes except the ID of the class. Only the attributes modified within the update specification will be edited.

To delete a VM class from your environment,

Associating a VM Class with a vSphere Namespace

You must associate a VM class with a namespace to make it available for DevOps engineers to deploy VMs in a self-service manner.

You can associate one or more VM classes with a single namespace or you can add one VM class to several namespaces. You can use the predefined VM classes that vSphere Supervisor provides or you can create custom ones. See Create a VM Class in .

Since VM Service is the feature responsible for handling VM classes, you must make that service aware of the VM classes available to the engineers using a specific namespace. You can achieve this when you create a new namespace or edit an existing one. See Managing Namespaces on a .

When you create the VM Service specification set the list of VM classes that must be used to create VMs on the specific namespace. You achieve this by

You can also associate one or more content libraries with a namespace that has the VM Service enabled. See Associating a Content Library with a Namespace.

vCenter Server Management

You can use the API to configure, monitor, maintain, and update vCenter Server.

Authorization Model for Administration of vCenter Server

There are three types of authorization levels in vCenter Server.

Table 53: Authorization Levels

Authorization Level	Description
operator	A user has read access to configuration settings.
administrator	A user has read and write access to configuration settings, but cannot manage user accounts.
super administrator	A user has all the capabilities of the other roles, and has the additional capabilities of creating local user accounts and accessing the local Bash shell.

This model applies to the API and all other interfaces to vCenter Server except when you use SSH and log in by using a local account.

Authorization Model Mapping to the vCenter Single Sign-On Domain

The three-level authorization model of vCenter Server maps to local roles and to vCenter Single Sign-On groups, depending on how the user authenticated. This model allows consistent security control regardless of operational context.

The authorization levels map to group and role.

Table 54: Authorization Mapping

Authorization Level	vCenter Single Sign-On Group	vCenter Server Local Role
operator	SystemConfiguration.Administrators	operator
administrator	SystemConfiguration.Administrators	admin
superAdministrator	SystemConfiguration.BashShellAdministrators	superAdmin

When a super administrator adds user accounts, the options available include a choice of the role to assign to the new user.

Using the Operator Role

The **operator** role is the most restricted of the authorization levels available to users who work with vCenter Server.

Operators are allowed to view information about vCenter Server. They are not allowed to alter its configuration. The **operator** role is suited for monitoring and reporting functions. For example, the **operator** role provides access to the following methods.

- `resources.system.health.get`
- `resources.storage.stats.list`
- `services.status.get`

Using the Admin Role

The **administrator** role provides an intermediate authorization level for users who manage vCenter Server.

An **administrator** role is required for users who alter the vCenter Server configuration, exercise control functions, or other operations that can affect regular users.

For example, the **administrator** role provides access the following methods.

- `networking.ip4v.renew`
- `networking.firewall.addr.inbound.add`
- `services.control`
- `shutdown.reboot`

Using the SuperAdmin Role

The **superAdmin** role is the most expansive authorization level for users who manage vCenter Server.

The **superAdmin** role allows unrestricted access to vCenter Server. This role is required for adding or altering user accounts and for using the Bash shell.

Performing Privilege Checks Operations

Privilege checks recorder is a feature which allows you to monitor and subsequently query the privileges that were checked. You can use the recordings to create scripts that automatically create roles with minimum required privileges to run specific workflows of operations.

Currently, it is very hard to find out the minimal set of privileges that are required to run a specific workflow of operations. The `PrivilegeChecks` provides for retrieving a list of the latest privilege checks along with the corresponding sessions, users, managed objects, and operation IDs (`opIDs`).

The following table lists the available privilege checks operations.

Operation	Description
List privilege checks	Returns a list of privilege checks that match criteria which you specified.
Get latest privilege check	Returns a marker to the last recorded privilege check. This allows retrieving all privilege checks up to or after a specified moment in time.

Filtering Privilege Checks

When you retrieve lists of privilege checks, you can refine your search by using the . The following table lists the available filtering options.

Filtering Option	Description
`Objects`	IDs of the managed objects on which the privilege check was performed. If , all objects match.
`OpIDs`	`OpIDs` of the requests for which the check was performed. If , all `opIDs` match.
`Principals`	Principals for which the privilege check was performed. The value matches privilege checks for anonymous sessions. If , all principals match.
`Privileges`	Privileges that were checked. If , all privileges match.

Table continued on next page

Continued from previous page

Filtering Option	Description
`Sessions`	Sessions for which the check was performed. If , all sessions match.

Using Advanced Settings

You can configure advanced settings by using the vSphere Client. Some of the advanced settings are not available in the API, but are required for the privilege checks recorder to function properly. For information about configuring advanced settings, see *vCenter Server Configuration*.

Advanced Setting Name	Description
`config.vpxd.privCheck.pageSize`	Specifies the default page size for privilege checks lists.
`config.vpxd.privCheck.bufferSize`	Specifies the count of privileges to be kept in memory. The default value is 0. If you do not change the default value, the privilege checks recorder does not record any data.
`config.vpxd.privCheck.cleanupInterval`	Specifies the interval on which privilege checks for unused sessions are cleaned up. The default value is 30 minutes.

vCenter Server Installation and Setup

You can use the API to perform operations related to stage 2 of the installation process. You can also perform backup, restore, and troubleshooting operations.

Install Stage 2

The vCenter Server API provides methods for performing stage 2 deployment operations on a newly installed vCenter Server instance.

The vCenter Server instance is deployed in two stages. With stage 1 of the deployment process, you deploy the OVA file, which is included in the installer. With stage 2 of the deployment process, you set up and start the services of the newly deployed vCenter Server instance.

To complete stage 1 of the deployment process, you can use the GUI installer or perform a CLI deployment. For details, see *vCenter Server Installation and Setup*. Alternatively, you can perform a deployment by using the VMware OVF Tool. For details, see *OVF Tool User's Guide*.

Setting Up a Newly Installed vCenter Server Instance

You can use the API to set up a newly deployed vCenter Server instance.

After stage 1 of the deployment process completes successfully, the vCenter Server instance enters in an `INITIALIZED` state. If the instance is not initialized, you cannot run stage 2 of the deployment process. You can get the state of the vCenter Server instance by using the `vcenter deployment` service. The vCenter Server instance can enter six states during the deployment process.

Figure 96: Install Stage 2 State Diagram

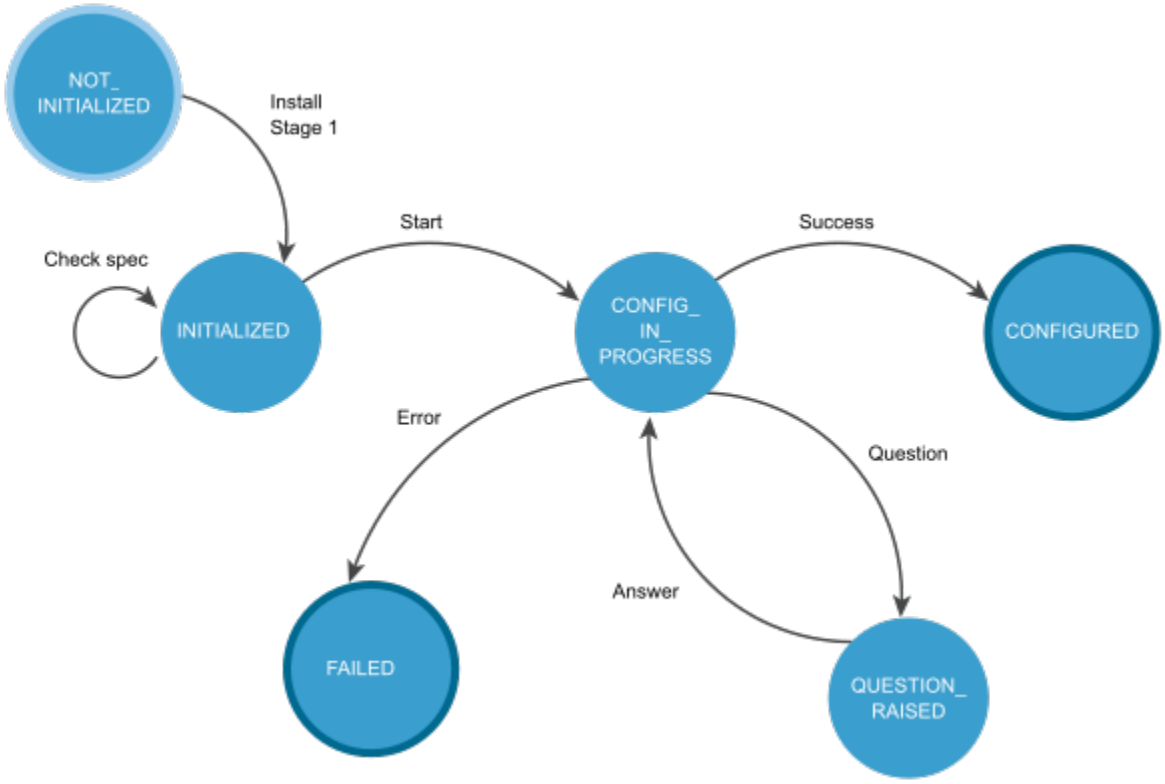


Table 55: vCenter Server Instance States During Install Stage 2

State	Description
`NOT_INITIALIZED`	The install stage 1 phase is in progress, not started, or failed.
`INITIALIZED`	The vCenter Server instance is deployed and ready for setup.
`CONFIG_IN_PROGRESS`	The setup process is in progress.
`QUESTION_RAISED`	You must answer the question to continue the setup process. The vCenter Server instance stays in the `QUESTION_RAISED` state until it receives the correct answer.
`FAILED`	Errors occurred during the setup process. You can check the errors, warnings, and info data structures.
`CONFIGURED`	The vCenter Server instance is installed and configured successfully.

`FAILED` and `CONFIGURED` are final states.

Table : User Operations lists operations that you can perform to set up your newly deployed vCenter Server instance.

Table 56: User Operations

Operation	Description
Get deployment information	You can retrieve information about the current deployment status. This operation is useful both before initiating stage 2 of the deployment and for monitoring the progress of the setup process.
Validate the configuration document	You can optionally verify whether your install spec is valid before starting the setup process.
Configure the vCenter Server instance	You can initiate the setup process by providing an install spec that defines the values for the settings that you want to configure.
Get question	You can retrieve a question raised during the setup process.
Answer question	You can provide an answer to the question raised during the setup process. The available answer values are `YES`, `NO`, `OK`, `CANCEL`, `ABORT`, `RETRY`, and `IGNORE`. The possible answer values depend on the type of the question. **NOTE** Each question has a default answer value. If you set questions to receive automatic answers in the install spec and a question is raised during the setup process, the default answer value is automatically provided as the answer to the question.

Workflows for Install Stage 2

You can use the `vcenter deployment` API to run the install stage 2 process of your vCenter Server instance.

Figure: Install Workflow and Figure: Install Stage 2 Workflow show example install workflows.

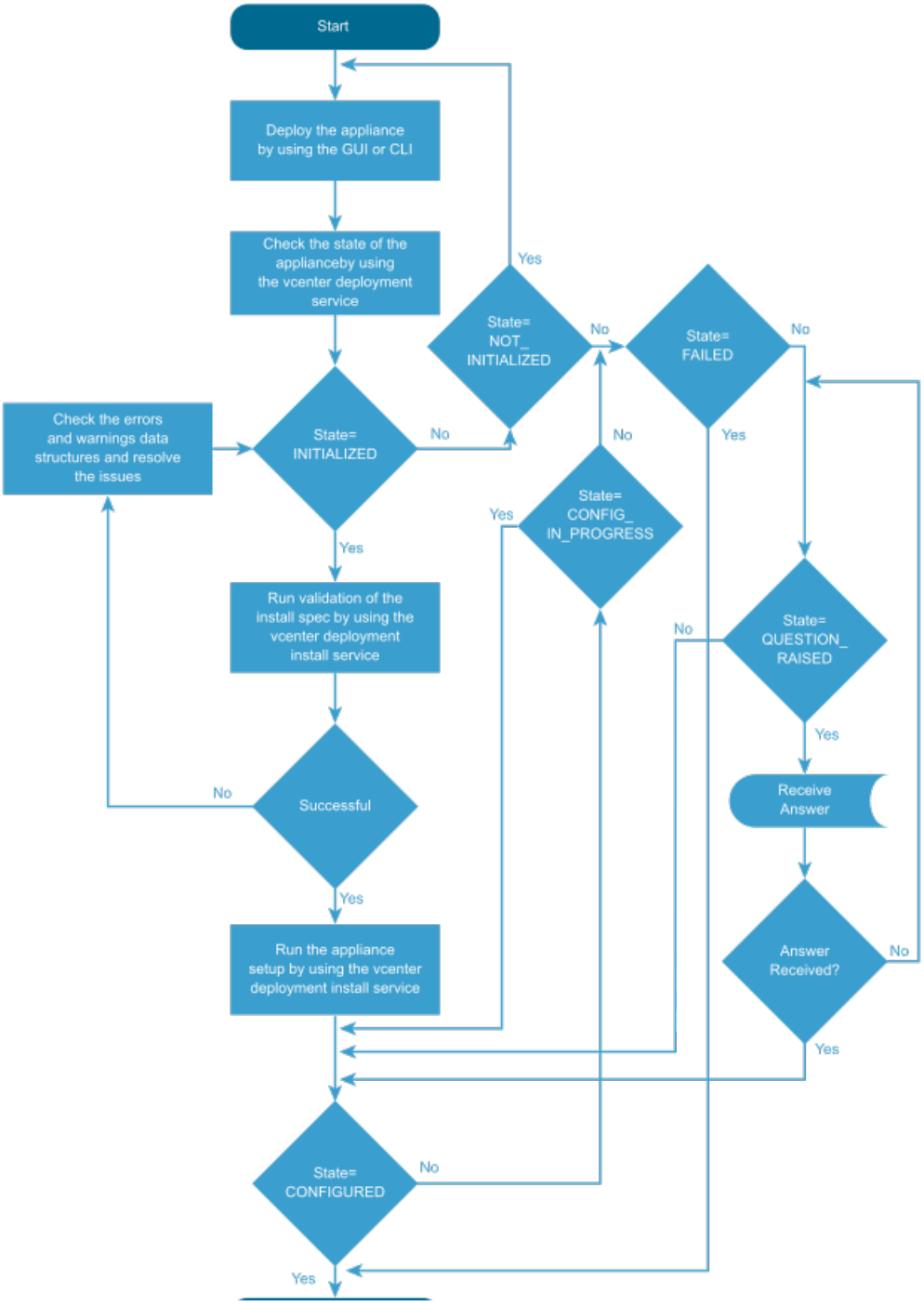
During stage 1, the vCenter Server instance is in a `NOT_INITIALIZED` state. After a successful deployment, the vCenter Server instance enters in an `INITIALIZED` state. If there are errors during stage 1, the vCenter Server instance stays in a `NOT_INITIALIZED` state and you must redeploy it.

You can check the state of the vCenter Server instance before, during, and after the setup process. You can run the install stage 2 process if the vCenter Server instance is initialized. You can check the setup configuration before you initiate stage 2 by running pre-checks. If errors or warnings appear during the validation of the install specification, you must remove the causes and correct the specification.

During the setup process, the regular vCenter Server instance state is `CONFIG_IN_PROGRESS`. The vCenter Server instance can also enter in a `FAILED` or `QUESTION_RASED` state. If a question appears during the setup, the vCenter Server instance enters in a `QUESTION_RAISED` state and stays in it until you provide an answer. You can set questions to receive automatic answers in the install spec and if a question is raised during the setup process, the default answer value is automatically provided as the answer to the question.

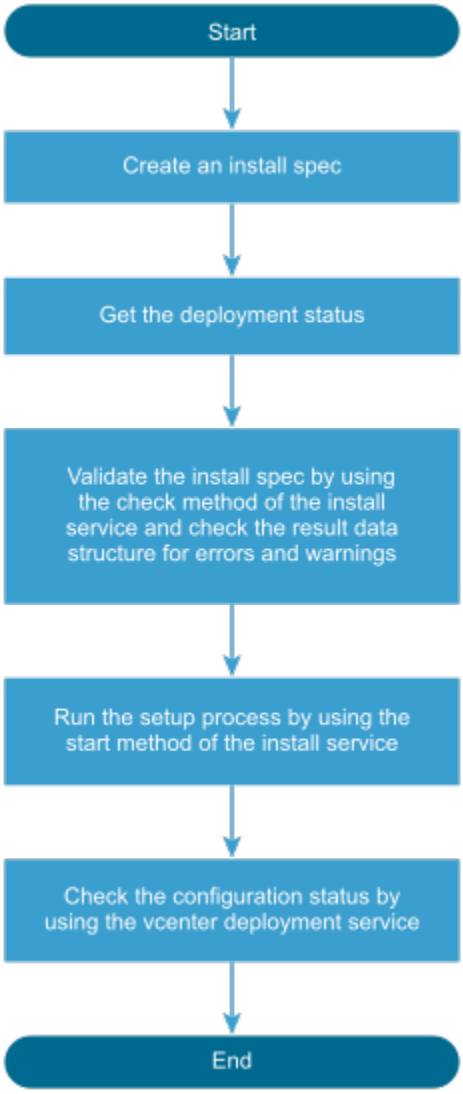
If errors occur during the setup process, the vCenter Server instance enters in a `FAILED` state and you must restart the setup after the causes are removed. If the setup is successful, the vCenter Server instance enters in a `CONFIGURED` state.

Figure 97: Install Workflow



For information about the states of the vCenter Server instance and available operations, see Setting Up a Newly Installed vCenter Server Instance.

Figure 98: Install Stage 2 Workflow



You can run the setup pre-checks and the install stage 2 process by creating and passing an `InstallSpec`. In `InstallSpec`, you define the setup configuration. See Figure: Install Stage 2 Workflow . You can run the setup in silent mode by setting the `InstallSpec.auto_answer` to `true`. The default value of `InstallSpec.auto_answer` is `false` and the setup is in interactive mode, in which you must provide answers to the raised questions.

For information about the classes, variables, and default values, see the *`API reference`* documentation.

File-Based Backup and Restore of vCenter Server

You can back up a vCenter Server instance and later restore the instance from the backup copy.

Backing up vCenter Server

The vCenter ServerKey definition for "mgmt_api" not found in the DITA map. supports backing up key parts of the vCenter Server instance. This allows you to protect vCenter Server data and to minimize the time required to restore data center operations.

The backup process collects key files into a tar bundle and compresses the bundle to reduce network load. To minimize storage impact, the transmission is streamed without caching in the vCenter Server instance. To reduce total time required to complete the backup operation, the backup process handles the different components in parallel.

You have the option to encrypt the compressed file before transmission to the backup storage location. When you choose encryption, you must supply a password which can be used to decrypt the file during restoration.

The backup operation always includes the vCenter Server database and system configuration files, so that a restore operation has all the data needed to re-create an operational vCenter Server instance. Current Alarms are included as well. You also have the option to specify additional data sets, called parts. In this release, you can specify a data set that includes Statistics, Events, and Tasks.

Backup and Restore Protocols for vCenter Server

The vCenter Server backup and restore feature supports a number of plug-in communication protocols.

Choose one of these protocols as the backup location type when you invoke the operation.

- FTP
- FTPS
- SCP
- HTTP
- HTTPS
- NFS
- SMB

The value `PATH` for the location type field indicates a locally mounted volume.

> **NOTE**
> If you specify the `SCP` protocol, you must specify an absolute path as the value of the location type field when you create the backup job.

Calculate the Size Needed To Store the Backup File

When you prepare to do a backup of avCenter Server instance, you can use the API to calculate the storage space needed for the backup file.

- Verify that you have avCenter Server instance running.
- Verify that you are familiar with authentication methods. See Authentication Mechanisms.

You can do this task when you are choosing a backup storage location or whenever your existing storage location might be approaching full capacity.

1. Authenticate to thevSphere Automation API endpoint and establish a session.
2. Request a list of backup parts available.
3. For each available backup part, request the size of the backup file.
   The backup process calculates the compressed size of each backup part.
4. Choose which parts to include in the backup, and sum their sizes.

   The backup storage server must have sufficient space to contain the chosen parts.

After you choose which backup parts you will store, and verify that the backup storage server has sufficient free space, you can launch a backup job. For information, see Back up a vCenter Server Instance by Using the API.

Python

This example shows how to use Python to collect the information you need to calculate the size needed to store a backup image of thevCenter Server instance.

```
from com.vmware.appliance.recovery.backup_client import Parts


# This example assumes you have previously created a session

# and stored the session ID in my_stub_config.


# Issue a request to list the backup image parts.

Parts_stub = Parts( my_stub_config )

parts = Parts_stub.list()


# Extract IDs of backup image parts.

sizes = {}

total = 0

for part in parts :

   size = Parts_stub.get( part.id )

   sizes[part.id] = size

   total += size


# Show the result.

print( 'Backup image parts:' )

for part_id in sizes.keys() :

   print( '  part {0} = {1}KB'.format( part_id, sizes[part_id] ) )

   print( 'Total size: {0}KB'.format( total ) )
```

Back up a vCenter Server Instance by Using the API

You can use the Key definition for "mgmt_api" not found in the DITA map. to create a backup of the vCenter Server database and key components of the vCenter Server instance.

- Verify that the vCenter Server instance is in a ready state. All processes with start-up type automatic must be running.
- Verify that no other backup or restore jobs are running.
- Verify that the destination storage location is accessible to the backup process.
- Verify that the path to the destination directory exists, as far as the parent directory.
- If the destination directory does not exist, the backup process creates it. If the directory does exist, verify that it is empty.

- Verify that the destination storage device has sufficient space for the backup file. For information about how to calculate the space needed for the backup file, see Calculate the Size Needed To Store the Backup File.

This procedure explains the sequence of operations you use to create a backup file of the vCenter Server instance. You can do this as part of a regular maintenance schedule.

1. Authenticate to the vSphere Automation API endpoint and establish a session.
2. to describe the backup operation.

   The specifies several , especially the backup location, the protocol used to communicate with the storage server, the necessary authorization, and which optional parts of the database you want to back up. The core inventory data and Alarms are always backed up, but you can choose whether or not to back up Statistics, Events, and Tasks. Collectively, this optional part of the backup is referred to as `seat`.

3. Issue a request to start the backup operation.
4. From the response, save the unique job identifier of the backup operation.
5. Monitor the progress of the job until it is complete.
6. Report job completion.

The backup file of the vCenter Server instance is created.

Schedule a Backup Job

You can automate the backup process by creating a schedule that runs backup jobs at specific times.

- Verify that you can access the backup server and you have read and write permissions.
- Verify that you have established a connection to the vAPI services.

You can keep existing backups on the backup server. The retention policy defines the maximum number of backups that the server keeps. You can also specify whether the backup job should run once, or on a recurring basis. The recurrence policy defines the days of the week and specific times at which the backup job is scheduled to run.

1. Create a .
2. Specify the retention and recurrence information.
3. Create a schedule by specifying the backup location, user credentials to access the location, retention, and recurrence information.
4. and pass the updated information.
5. Get a backup schedule by passing a schedule ID.

Run the backup job by using the schedule.

Restoring vCenter Server

The vCenter ServerKey definition for "mgmt_api" not found in the DITA map. supports restoring a vCenter Server instance from a backup copy. The API simplifies the process by unifying the handling of various components of vCenter Server in a single operation.

The process of restoring a vCenter Server instance from a backup has two phases.

1. Deploy a new vCenter Server instance. OVF deployment is described in the *vSphere Automation SDKs Programming Guide*.

2. Invoke the `restore` operation from the Key definition for "mgmt_api" not found in the DITA map. to apply configuration settings and load the vCenter Server database from the backup file.

   NOTE
   You cannot specify optional parts for the restore operation. The restore operation includes all optional parts, such as Events and Tasks, that were specified at the time when the backup file was created.

Authentication When Restoring a vCenter Server Instance

During the process of restoring a vCenter Server instance from a backup image, you cannot use token-based authentication. You must use basic authentication until the vCenter Server instance is fully configured.

When you restore your vCenter Server instance from a backup file, it begins in an unconfigured state. During this time, you must use basic authentication to access the Key definition for "mgmt_api" not found in the DITA map.. When you use basic authentication, do not use the vSphere Automation API endpoint. Instead, you must connect your client to port 5480 of the vCenter Server instance.

When you use basic authentication, you must pass user name and password credentials with each . Use credentials that are known to the guest operating system of the vCenter Server instance.

Availability of Services While Restoring a vCenter Server Instance

During the process of restoring the vCenter Server backup file, services in the vCenter Server instance must restart. While they are restarting, your API client receives an error message.

You can write your client to trap the error, but you have no way to know when the vCenter Server services are running again. To determine when the restore process is complete, you must retry the API connection until it succeeds, then request the status of the job.

Restore a vCenter Server Instance by Using the API

You can use the Key definition for "mgmt_api" not found in the DITA map. of to restore a vCenter Server instance from a backup file containing the vCenter Server database and key components of the vCenter Server instance.

- Verify that the backed up vCenter Server instance is powered off.
- A new vCenter Server instance must be deployed in an unconfigured state, except that it must have a fully qualified domain name or IP address that matches the old one.
- Verify that the new vCenter Server instance has the same build number as the one in the backup file.
- Verify that the new vCenter Server instance has a size equal to or greater than the old one. If the old vCenter Server instance was customized to exceed the largest template size, the new one must be customized to the same size.
- Verify that no other backup or restore jobs are running.
- Verify that the destination storage location is accessible to the vCenter Server restore process.

1. to describe the restore operation.
2. Issue a request to start the restore operation.
3. Monitor the progress of the job until it is complete.
4. Report job completion.

After the vCenter Server instance is fully configured by the restore operation, you can resume using the vSphere Automation API endpoint for subsequent operations.

Reconcile a vCenter Server Instance with Nodes in Embedded Linked Mode

You can run the reconciliation process after you successfully restored your vCenter Server instance. By using the API or HTTP requests, you can reconcile vCenter Server nodes that work in an embedded linked mode and are connected in a ring or daisy-chain.

- Verify that you successfully restored your node from an image.
- Verify that the replication partners are available.
- Verify that you restored your nodes in the correct order, if you use a daisy-chain topology.
- Verify that you have administrator's credentials to your Single Sign-On domain.
- Verify that there is no running or failed reconciliation job.

Reconciliation is a post-restore process that checks whether the vCenter Server partners in embedded linked mode are available, synchronizes the vCenter Server data and services with the partners, and runs the vCenter Server services. The processes of restore and reconciliation depend on the topology and if there are changes in the topology between the backup and restore, you cannot restore the embedded linked mode. If the replication partners are not available and you try to restore the first node, you must ignore the warnings. In this case, any changes that are made in the topology or infrastructure after the backup will be lost. If you restore a node different from the first one, you must add it to the domain of the first node. If you use a daisy-chain topology, you must first restore the first node, and after that to restore the second, link it to the first one, and apply the same to the following nodes.
You can use the reconciliation API after a file-based and an image-based restore. After an image-based restore, you can run the reconciliation process by using the API or UI. After a file-based restore, you can monitor the reconciliation process by using the `GET https://<vcenter_ip_address_or_fqdn>/api/appliance/recovery/reconciliation/job` HTTP request. For information about how to restore a vCenter Server instance from an image or a file by using the UI, see the *vCenter Server Installation and Setup* documentation.

1. , specify user name and password of Single Sign-On administrator, and set the `ignore_warnings` field to `true`.

   The default value of `ignore_warnings` is `false`. If you do not set `ignore_warnings` to `true`, the reconciliation fails due to the validation warnings.

2. Run a reconciliation job by using the .

   You can check the result of the operation by reading the . contains information about the job such as description, status, progress, error, start and end time.

3. Get the status of the job by .

   The possible states are `NONE`, `RUNNING`, `FAILED`, and `SUCCEEDED`.

Troubleshooting for vCenter Server Installation or Deployment

You can use the API to perform troubleshooting operations related to the installation and deployment of vCenter Server.

Managing System Logs

You can automate the forwarding of vCenter Server system log messages to remote logging servers by using the vCenter Server Management API.

You can configure the syslog forwarding by using the API or user interface. For information about how to manage the syslog by using the user interface, see the *vSphere Monitoring and Performance* documentation.

Managing Support Bundles

You can retrieve information about support bundles and create support bundles.

The following table lists the operations that you can perform to manage support bundles.

Table 57: User Operations

Operation	Description
Get support bundle components	You can retrieve a list of components and manifests included in the support bundle. Manifests specify the files that must be collected for a component as part of the support bundle.
List support bundles	You can retrieve a list of generated support bundles. The list contains details about each support bundle.
Create a support bundle	You can generate a support bundle. Optionally, you can specify the support bundle components and the partition where you want to save the support bundle.

You can run support bundle management operations by using the vSphere Automation SDK or sending an HTTP request.

REST

You can use HTTP requests to perform support bundle management operations.

The following HTTP requests show the syntax that you can use to perform the available user operations.

- Get support bundle components

  ```
  GET https://<vcenter_ip_address_or_fqdn>:443/api/appliance/support-bundle/
  components
  ```

- List support bundles

  ```
  GET https://<vcenter_ip_address_or_fqdn>:443/api/appliance/support-bundle
  ```

- Create a support bundle

  ```
  POST https://<vcenter_ip_address_or_fqdn>:443/api/appliance/support-bundle
  ```

For information about the content and syntax of the HTTP request body for each operation, see the *API reference* documentation.

The following examples provide sample input and output values for the support bundle management operations.

- *Example: Get Support Bundle Components*
  This operation does not require input.

  ```
  GET https://<vcenter_ip_address_or_fqdn>:443/api/appliance/support-bundle/
  components
  ```

Type	Example

Output	{
	"Group1" : List of manifests under this group,
	"Group2": ...
	}

- *Example: List Support Bundles*
  This operation does not require input, but supports optional query parameters.

  ```
  GET https://<vcenter_ip_address_or_fqdn>:443/api/appliance/support-bundle
  ```

Type	Example

Input	"IterationSpec":{
	"size" : 10
	}
	FilterSpec:{
	"creationStatus" : "SUCCEEDED",
	"available" : True/False
	}

Output	{
	"Description": "Reason for generation",
	"Generation time": 2 Mar 2021,
	"URL" : "Download URL",
	"Downloadable" : True/False
	"Expiration time": "URL expiration time"
	}

- *Example: Create a Support Bundle*
  This operation requires description input and supports optional body parameters. As output, you receive a task ID, which you can use to track the progress of the task. When the operation is completed, you receive the download URL in the task response.

  POST https://*<vcenter_ip_address_or_fqdn>*:443/api/appliance/support-bundle

Type	Example

Input	{
	"Description" : "Reason",
	"Inclusions" : List of components to be included in the bundle
	<Optional> path: partition to be used.
	}

Output	Task ID

vCenter Server Upgrade

You can use the API to perform operations related to stage 2 of the upgrade process. You can also perform historical data transfer operations.

Upgrade Stage 2

You can upgrade your vCenter Server instance by using the API, CLI, or GUI.

For information about how to upgrade the vCenter Server instance by using CLI and GUI, see the *vCenter Server Upgrade* documentation.

Upgrading a vCenter Server Instance

You can use the API during stage 2 of the vCenter Server instance upgrade.

By using the API, you can upgrade your vCenter Server instance. For information about the upgrade process, its stages, supported configurations, upgrade paths, prerequisites for upgrading, and the sequence for upgrading a vSphere environment, see the *vCenter Server Upgrade* documentation.

After you deploy the vCenter Server instance on stage 1 by using the GUI or CLI, the instance enters in an `INITIALIZED` state. If the vCenter Server instance is not initialized, you cannot run stage 2 of the upgrade process. You can get the state of the vCenter Server instance by using the `vcenter deployment` service. There are six states during the upgrade process.

Figure 99: Upgrade Stage 2 State Diagram

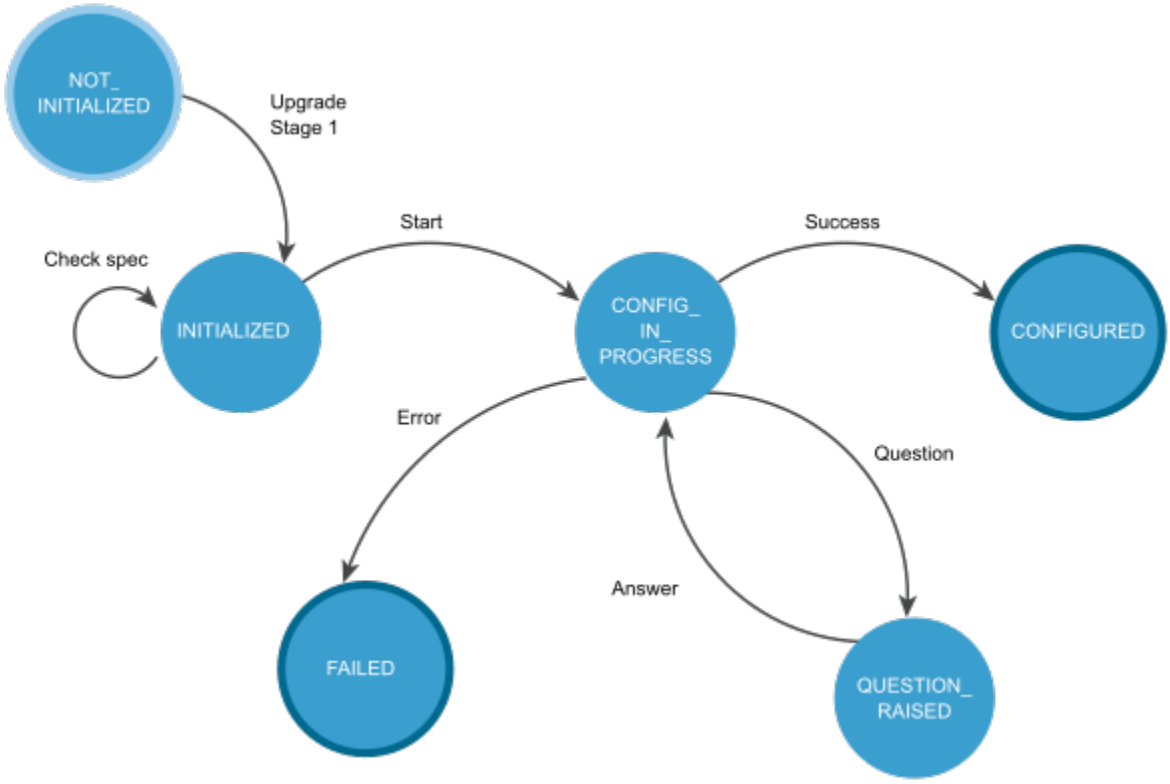


Table 58: vCenter Server Instance States During Upgrade Stage 2

State	Description
`NOT_INITIALIZED`	The upgrade stage 1 phase is in progress, not started, or failed.

Table continued on next page

Continued from previous page

State	Description
INITIALIZED	The vCenter Server instance is deployed and ready for upgrading.
CONFIG_IN_PROGRESS	The upgrade process is in progress.
QUESTION_RAISED	You must answer the question to continue the upgrade process. The vCenter Server instance stays in the QUESTION_RAISED state until it receives the correct answer.
FAILED	Errors appeared during the upgrade process. You can check the errors, warnings, and info data structures.
CONFIGURED	The vCenter Server instance is upgraded or configured successfully.

FAILED and CONFIGURED are final states.

You can roll back a vCenter Server instance upgrade by using the GUI. For information about how to roll back a vCenter Server instance, see the *vCenter Server Upgrade* documentation.

After the upgrade, you can check the vCenter Server instance type, domain registration, services, their state and health status by using the API. For information about how to verify whether the upgrade of your vCenter Server instance is successful, see the *vCenter Server Upgrade* documentation.

Table : User Operations shows operations that you can perform to upgrade your vCenter Server instance.

Table 59: User Operations

Operation	Description
Operations for upgrading	
Get the state of the vCenter Server instance	You can get the state of the vCenter Serverinstance before, during and after the upgrade process.
Check	You can validate the upgrade spec before you run the upgrade process. If the vCenter Server instance is in the INITIALIZED state, you can run the validation. The operation runs upgrade pre-checks. You can check the errors, warnings, and status data structures before you run the upgrade process.
Start	If the vCenter Server instance is in an INITIALIZED state, you can run the upgrade process. If errors appear during the upgrade, you can download the vCenter Server support bundle.
Get	If the vCenter Server instance is in a CONFIGURED state, you can get the spec that is used for upgrading.
Operations for getting and answering a question	
Get	You can get the raised question. If you set the Upgrade.auto_answer to true, the upgrade process will be in a silent mode and the vCenter Serverinstance does not generate questions. It uses default answers and you should not provide an answer.

Table continued on next page

Continued from previous page

Operation	Description
Answer	You can provide an answer to the raised question. The available answers for the upgrading are `OK`, `CANCEL`, `YES`, `NO`, `ABORT`, `RETRY`, and `IGNORE`. The answer depends on the type of the question. If you set the `Upgrade.auto_answer` to `true`, the upgrade process will be in a `silent` mode and the vCenter Serverinstance does not generate questions. It uses default answers and you should not provide an answer.

For information about the available operations in the API, see the `vcenter deployment`, `vcenter deployment upgrade`, `vcenter services`, and `vcenter system-config deployment type` services in the *`API reference`* documentation.

Workflows for Upgrade Stage 2

You can use the `vcenter deployment` API to run the upgrade stage 2 process of your vCenter Server instance.

Figure: Upgrade Workflow and Figure: Upgrade Stage 2 Workflow show example upgrade workflows.

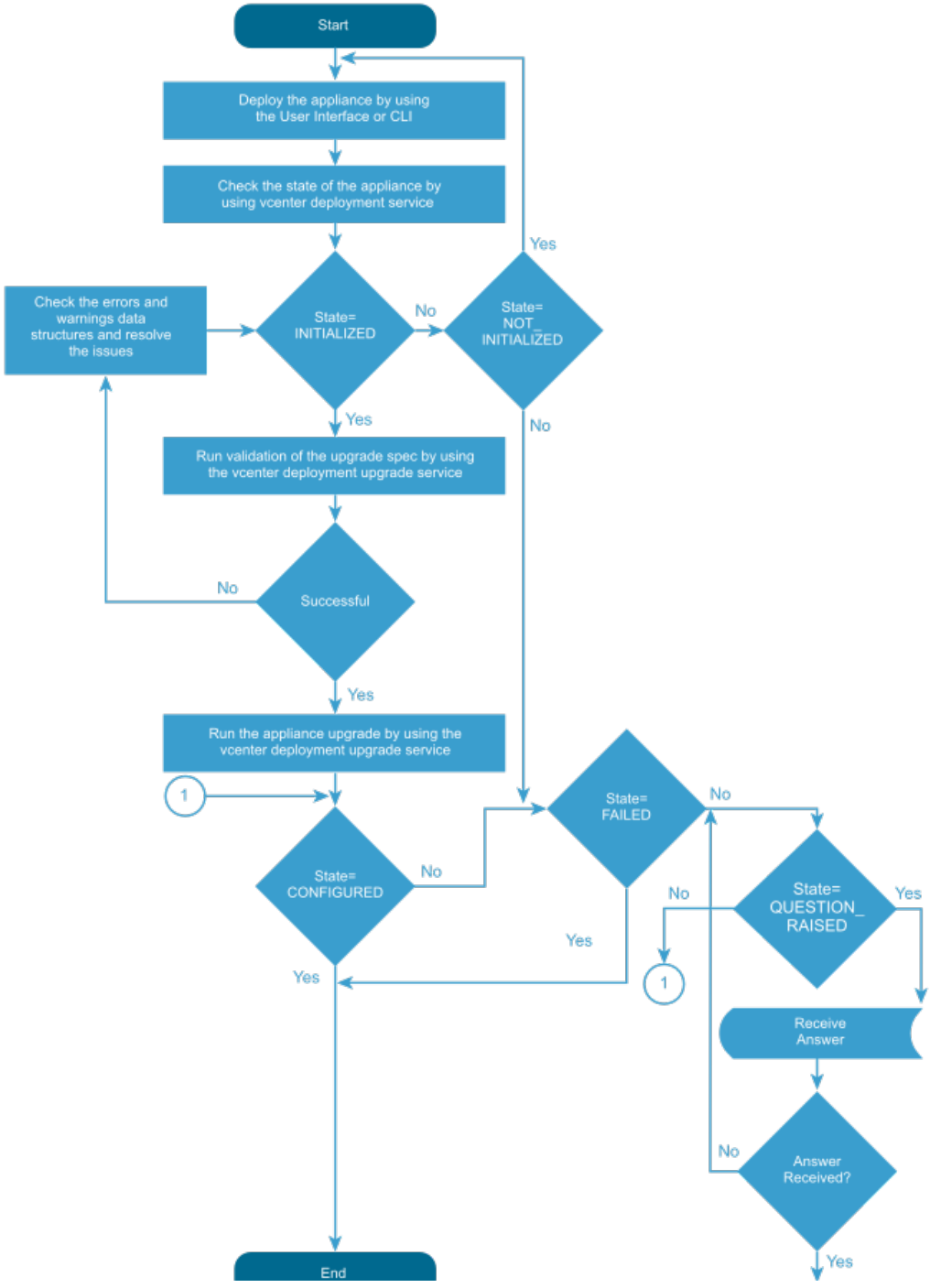
During stage 1, the vCenter Server instance is in a `NOT_INITIALIZED` state. After a successful deployment, the vCenter Server instance enters in an `INITIALIZED` state. If there are errors during stage 1, the vCenter Server instance stays in a `NOT_INITIALIZED` state and you must redeploy it.

You can check the state of the vCenter Server instance before, during, and after the upgrade process. You can run the upgrade stage 2 process if the vCenter Server instance is initialized. You can check the upgrade configuration before you run the upgrade by running pre-checks. If errors or warnings appear during the validation of the upgrade specification, you must remove the causes and correct the specification.

During the upgrade process, the vCenter Server instance can enter in a `FAILED` or `QUESTION_RASED` state. If a question appears during the upgrade, the vCenter Server instance enters in a `QUESTION_RAISED` state and stays in it until you provide an answer. You can run the upgrade in silent mode, in which the vCenter Server instance does not generate questions, and uses default answers.

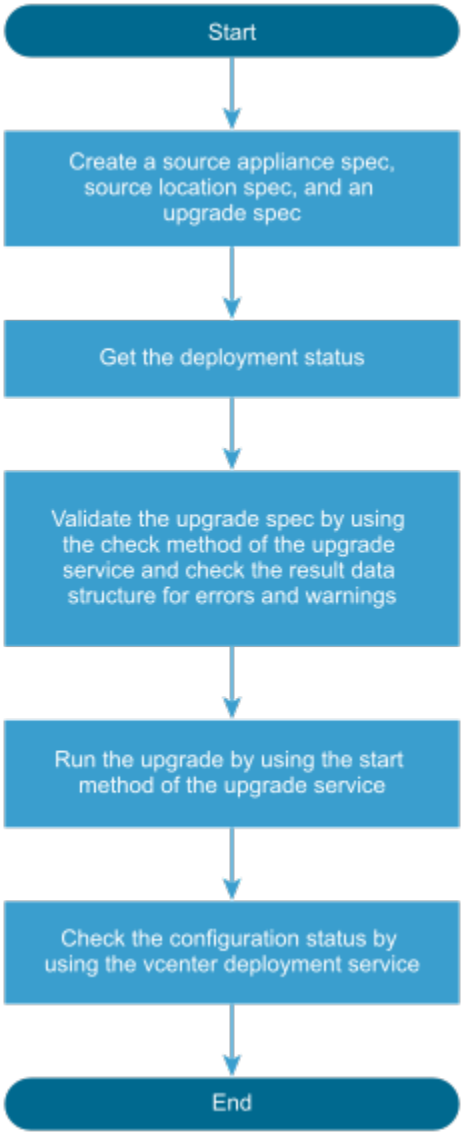
If errors appear during the upgrade, the vCenter Server instance enters in a `FAILED` state and you must remove the causes, redeploy the instance and restart the upgrade. If stage 2 of the upgrade process is successful, the vCenter Server instance enters in a `CONFIGURED` state. If the vCenter Server instance is configured, you can check its services and pause the historical data transfer.

Figure 100: Upgrade Workflow



For information about the states of the vCenter Server instance and available operations, see Upgrading a Instance.

Figure 101: Upgrade Stage 2 Workflow



You can run the upgrade pre-checks and the upgrade stage 2 process by creating and passing an `UpgradeSpec`. In `UpgradeSpec`, you define the upgrade configuration and specify the source vCenter Server instance and the source ESXi host in `SourceApplianceSpec` and `LocationSpec`. See Figure: Upgrade Stage 2 Workflow . You can run the upgrading in silent mode by setting the `UpgradeSpec.auto_answer` to `true`. The default value of `UpgradeSpec.auto_answer` is `false` and the upgrading is in interactive mode, in which you must provide answers to the raised questions.

For information about the classes, variables, and default values, see the *`API reference`* documentation.

Historical Data Transfer

If you migrate vCenter Server for Windows, you can transfer the historical data of your source vCenter Server instance together with the core configuration data.

Deferred Import

The deferred import is a process of historical data transfer after the successful migration of a vCenter Server instance with an external database. The historical data includes statistics, events, and tasks.

By using the deferred import feature, you can postpone the historical data transfer after the migration process completes, so that you manage the downtime of your environment. You can select whether all historical data, or only events and tasks, will be migrated with the core data during the migration. The historical data transfer and deferred import of historical data are deactivated by default. You can enable and configure the historical data transfer by using the API, Key definition for "vami" not found in the DITA map., vCenter Server installer, or CLI installer. A vCenter Server super administrator can run and control the migration and deferred import processes.

If you use the deferred import feature, the historical data is migrated with the core data and the historical data import process starts automatically after a successful migration and when the vCenter Server instance is running. You can pause the historical data import and resume it later.

For information about how to configure and run the migration and deferred import processes by using the Key definition for "vami" not found in the DITA map., see the *vCenter Server Upgrade* documentation.

By using the API, you can configure, control, and monitor the data transfer process. If you use the API to enable the deferred import feature, you must create a history migration spec and set `defer_import` to `true`. For information about how to configure the deferred import by using the API, see the *API reference*.

The data import process has five states that you can check. If the historical data migration and the deferred import are configured, the historical data import starts automatically after a successful migration.

Figure 102: Deferred Import State Diagram

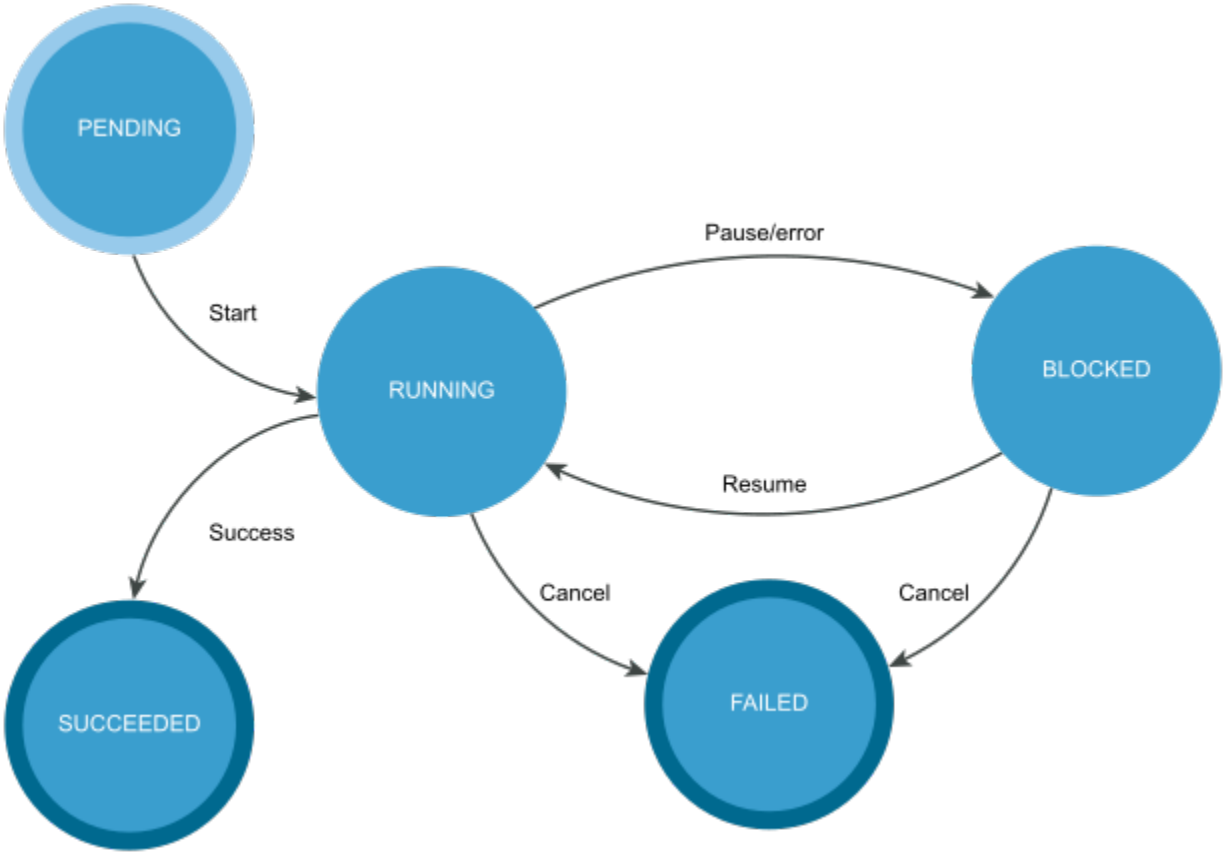


Table 60: User Operations

Operation	Description
Pause	You can pause the running data transfer process. If the data transfer is paused, you can resume or cancel it.
Cancel	You can cancel the data transfer process if it is in a `RUNNING` or `BLOCKED` state. **NOTE** If you cancel the data transfer, the process enters in a final `FAILED` state and you cannot resume the transfer.
Resume	You can resume the stopped data transfer.
Get status	You can retrieve the status of the data transfer process. There are five states. **`PENDING`** The transfer is not started. **`RUNNING`** The transfer is started or resumed. **`BLOCKED`** The transfer is paused or there was a recoverable error, such as not enough disk space, during the import. **`SUCCEEDED`** The transfer is successful. **`FAILED`** The transfer is canceled.

You can run the deferred import operations by using the API or sending an HTTP request.

NOTE
When you send the requests, you must use an authentication.

If you pause the data transfer by using the API or an HTTP request, you can resume or cancel the process by using the API or the Key definition for "vami" not found in the DITA map..

IMPORTANT
If you cancel the transfer process, and want to transfer the historical data later, you must restart the migration process.

Historical Data Import Errors

If an error appears during the data import, the import stops and the process enters in a `BLOCKED` state. You can resume the data import after you eliminate the cause.

You can check the errors, warnings, and info messages by reading the `info`, `status`, and `notifications` data structures.

If the information in the data structures is not enough, you can download the vCenter Server support bundle from `<vcenter_ip_address_or_fqdn>:443/appliance/support-bundle` and check the log files.

Table 61: Log Files

Log File	Path
API log file	`/var/log/vmware/applmgmt/applmgmt.log`
Backend log file	`/var/log/vmware/upgrade/upgrade-post-import.log`
Upgrade Runner log file	`/var/log/vmware/upgrade/deferredimport-upgrade-runner.log`
Deferred import log file	`/var/log/vmware/upgrade/DeferredImport_com.vmware.vcdb_<date_time>.log`

vCenter Server Configuration

You can use the API to perform operations related to health monitoring and capacity monitoring of vCenter Server. You can also manage the global FIPS compliance and perform infrastructure profile management operations.

Health Monitoring of vCenter Server

The vCenter Server API offers health status indicators for several key components of the system. These indicators can be polled periodically to monitor the components for problems.

The health status indicators report graded values from green to red. The general meanings of the grades are as follows.

green
: The component is healthy.

yellow
: The component is healthy, but may have some problems.

orange
: The component is degraded, and may have serious problems.

red
: The component is unavailable, or will stop functioning soon.

gray
: No health data is available.

Check Overall System Health of vCenter Server

vCenter Server provides a composite health indicator that enables you to test a single value that represents the health of all system components. This procedure shows how to test the composite health indicator.

Verify that you have an active authenticated session with vCenter Server. This procedure assumes that the session ID is present in the security context of a stub configuration.

The value of the overall system health indicator reflects the strongest trouble indication among the vCenter Server components. If any component has a `red` indicator, the overall system health indicator is `red`, else if any component has an `orange` indicator, the overall system health indicator is `orange`, and so on.
A `gray` value for any component indicates that data for the subsystem is unknown. If one or more components have a `gray` value, but all other subsystems are `green`, then the overall system health indicator is `gray` rather than `green`. However, if any component has a definite trouble indication, the overall system health indicator reflects the strongest trouble indication among the components.

1. Create an interface stub or REST path that uses the stub configuration.

2. Invoke the `health.system` method.
3. Format and display the resulting value.

Capacity Monitoring ofvCenter Server

vCenter Server keeps a history of statistics that you can use to monitor resources used by thevCenter Server instance.

You can use the statistics to spot peak usage demands or to monitor trends for advance warning of potential resource exhaustion.

Frequency and Retention of Statistics Collection invCenter Server

vCenter Server collects statistics from the guest operating system at regular intervals and stores them in a database. Users can query the statistics in the database by selecting a time period and a roll-up function thatvCenter Server applies to the statistics before returning them to the client.

After the monitoring service starts up, it begins requesting statistics from the guest operating system periodically, at a frequency that depends on the type of statistic. The service requests storage statistics once every 10 minutes, while it requests memory, CPU, and networking statistics once per minute. The collection times are fixed relative to the startup time of the monitoring service, rather than to clock time.

The monitoring service retains statistics approximately 13 months, by default. Older statistics are deleted by the service, creating a 13-month moving window within which you can query statistics. You can choose to delete statistics as needed to conserve storage resources.

Nature of Statistics invCenter Server

vCenter Server supplies statistics of several types.

The guest operating system computes statistics either as rates, such as CPU cycles per second, or as snapshots of size, such as KB used for storage. Statistics stored as size snapshots are collected at the end of their sample periods. Statistics stored as rates are computed as averages of values sampled frequently during each sample period.

When you query the statistics database, the units are not returned with the data, but you can determine the units for any metric by requesting metadata for the metric with the `get()` method.

Requesting Statistics fromvCenter Server

To request statistics, you must construct an appropriate request structure to filter statistics from the database.

To request data or metadata for a metric, you must supply the ID of the metric. You can get a list of metric IDs by using the `list()` method, which returns information on all available metrics.

When you query statistics, you provide a list of IDs to specify the metrics in which you are interested. You also supply a start time, an end time, a roll-up interval, and a roll-up function. These values interact as follows to determine the data returned to you.

- The response contains a list of data points for each metric ID you specified in the request.
- The start time and end time control the limits for the data you want in the response. The response contains data points only for statistics that have timestamps between those limits, inclusive of the endpoints. However, the start time is adjusted to a round number, in some cases.
- The roll-up interval enables you to control the granularity of the data points in the response. Rather than a response with a data point for every statistic between the start time and end time, you get a response with a number of data points equal to the number of intervals between the start and end times. Generally, you should specify a time period that is an even multiple of the interval, so that each data point in the response represents the same number of statistics.

- The roll-up function specifies how the response summarizes the statistics that fall within each interval. The resulting data point can be the maximum statistic value within collection interval, or the mean of the statistics values within the interval, and so on.

Statistics Collection Times

The actual time that a statistic was collected is not readily predictable.

The API does not enable you to determine the exact time that a statistic was collected. Furthermore, some statistics, such as those for storage metrics, might take seconds or minutes to collect, so that they are not available immediately at the time a request is made to the guest operating system.

However, because statistics are collected at regular intervals, and roll-up intervals for a request generally all have the same size, each data point in the response represents the same number of statistics as the others.

Statistics Interval Adjustment invCenter Server

When you make a request for statistics, the monitoring service might adjust the specified roll-up interval times to improve the appearance of statistics graphs in a graphical interface.

The monitoring service adjusts the start time of a data collection request when it is not an exact multiple of the interval length. In these instances, the start time is rounded downward to the previous UTC time that is a multiple of the interval. All subsequent intervals of the data collection are also adjusted to align with the new start time.

For example, if the start time is 10:31 and the interval length is 1 hour, the monitoring service adjusts the start time to 10:00 and the roll-up intervals have the following continuous pattern.

- 10:00 to 10:59:59.999
- 11:00 to 11:59:59.999
- 12:00 to 12:59:59.999

The monitoring service does not adjust the end time of a data collection. Consequently, the response to a statistics query might contain one more data value than expected, or an incomplete final interval might be lengthened.

Empty Data Values

In some instances, you might encounter a response that reports an empty data value, or even a series of empty data values. This might manifest as a list of data values containing some numeric values alternating with empty values.

- Empty data values can happen when the report time period is too short to be certain of containing any statistics. For instance, a time period of 30 seconds is half the length of the sample period for network metrics, so you have only a 50% chance of finding a network statistic during any 30-second reporting period.
- Empty data values can also happen when the interval is shorter than the sample period for a metric you have requested. In this case, some data points are present in the list, while others are empty because no statistic was collected during those intervals. For instance, an interval of 5 minutes is only half the length of the sample period for storage metrics, so every second data value is empty.
- Empty data values can also happen when the monitoring service has not finished collecting and writing the last sample to the database, even if the nominal sample timestamp falls within the report time period. For example, calculation of storage used can delay writing a storage statistic to the database. A request for the statistic during that delay time produces an empty data point in the response.

When a response contains an empty data value, this indicates that no statistics were collected during a collection interval. An appropriate action for the client in such a case depends on how the client is using the data. For example, if you are graphing a resource usage trend, you might choose to interpolate for the missing value to produce a smooth line.

Check Database Usage in vCenter Server

vCenter Server contains a database of all objects managed by the vCenter Server instance. In addition to inventory objects, the database includes vCenter Server statistics, events, alarms, and tasks. You can calculate the database storage consumption by adding the sizes of all data categories.

This task assumes you have previously authenticated and created a client session.

You need to monitor storage consumption in vCenter Server.

1. Prepare a request for database usage statistics.

   Include metric IDs both for `vcdb_core_inventory` and `vcdb_seat`. The name `vcdb_seat` refers to Statistics, Events, and Tasks in the vCenter Server database.

2. Issue the request to the API endpoint.
3. Process the resulting data points as needed.
4. Format and print the results.

The result of this procedure shows the storage used in the vCenter Server database, which includes storage overhead used for indexes and other optimizations beyond the actual size of the data.

List Storage Consumption By Data Type in vCenter Server

vCenter Server provides statistics on several types of storage.

Verify that you have authenticated and created a client session.

For example, you can query statistics about inventory storage, transaction log, and vCenter Server tasks. Many of these statistics are available both for storage consumed and storage available.
This task provides data for system administrators who need to monitor storage consumption in the guest operating system of vCenter Server.

1. Prepare a request for database usage statistics.

   Include metric IDs for each data type you want to monitor.

2. Issue the request to the API endpoint.
3. Process the resulting data points as needed.
4. Format and print the results.

You retrieved information about the storage consumption in your vCenter Server system.

Managing the Global FIPS Compliance

You can retrieve information about the current FIPS (Federal Information Processing Standards) settings of vCenter Server. You can also enable or deactivate the global FIPS compliance.

FIPS 140-2 is a U.S. and Canadian government standard that specifies security requirements for cryptographic modules. vSphere uses FIPS-validated cryptographic modules to match those specified by the FIPS 140-2 standard. The goal of vSphere FIPS support is to ease the compliance and security activities in various regulated environments.

The following table lists the operations that you can perform to manage the FIPS settings of your vCenter Server system.

Table 62: User Operations

Operation	Description
Get FIPS status	You can check whether the global FIPS compliance is currently enabled on the vCenter Server system.
Manage FIPS status	You can enable or deactivate the global FIPS compliance on the vCenter Server system.

NOTE
When you enable FIPS compliance, some components might present functional constraints. For more information, see *vSphere Security*.

You can run FIPS management operations by using the vSphere Automation SDK or sending an HTTP request.

Performing Infrastructure Profile Management Operations

You can export an existing vCenter Server configuration and import it to other vCenter Server instances.

You can export multiple configuration profiles at once. The exported data can contain general configuration settings and user content. You can replicate the same configuration across all vCenter Server instances in your environment by importing the same data package. You can also use the exported data as a backup if you need to revert to the last known good configuration. To avoid configuration issues, you can validate the exported data before importing it to a vCenter Server instance.

The following table lists the operations that you can perform to manage the configuration profiles in your infrastructure.

Table 63: User Operations

Operation	Description
List configuration profiles	You can retrieve a list of all configuration profiles that are registered with vCenter Server.
Export configuration profiles	You can export specific vCenter Server configuration profiles.
Validate configuration profiles	You can validate the exported vCenter Server configuration profiles. The validation process examines the configuration file for possible errors and conflicts and returns output. This operation can help avoid configuration issues or loading the wrong configuration file.
Import configuration profiles	You can import specific vCenter Server configuration profiles into another vCenter Server instance.

You can run infrastructure profile management operations by using the vSphere Automation SDK or sending an HTTP request.

Patching and Updating vCenter Server Deployments

You can use the API to perform operations related to the planning and installing of vCenter Server software updates.

Planning vCenter Server Updates

The vCenter Server API provides operations that can help you plan the life cycle of vCenter Server instances in your environment.

You can use the operations to discover VMware products that can be associated with vCenter Server, list associated products, and manage product associations. You can also get details about available vCenter Server updates, perform pre-checks, and produce reports. The reports can contain interoperability or pre-check information. Interoperability reports

contain information about the interoperability between the associated products and a specific vCenter Server version. Pre-check reports contain information about the compatibility of the current vCenter Server version with a pending update version. You can plan to perform vCenter Server updates based on the information gathered in the produced reports.

Performing Discovery and Planning Operations

You can retrieve information about VMware products associated with vCenter Server, list available vCenter Server updates, and produce reports. The findings can help you plan vCenter Server updates in your environment.

The life cycle management API provides operations that are grouped in the discovery, update, and reports categories. The discovery functionality of the API consists of the operations in the discovery category. The planning functionality of the API consists of the operations in the update category. Both functionalities can produce reports that you can download by using the reports category.

The discovery category provides operations for listing VMware products that can be associated with vCenter Server, managing products associations, and creating interoperability reports. The update category provides operations for listing all available updates and upgrades for vCenter Server and generating a pre-check compatibility report. The reports category provides an operation for downloading reports generated by interoperability and pre-check operations. By using the retrieved information, you can select one of the available patches and can plan an actual patch or upgrade for a specific vCenter Server version.

The following table lists the operations that are available in the discovery category.

Table 64: Discovery User Operations

Operation	Description
Get product catalog	You can retrieve a list of all VMware products that can be associated with vCenter Server.
List associated products	You can retrieve a list of all VMware product deployments in the environment that are associated with vCenter Server. **NOTE** The list contains both product deployments discovered automatically and deployments registered manually through the API.
Get associated product information	You can retrieve detailed information about a product associated with vCenter Server.
Create product association	You can manually associate a VMware product with vCenter Server.
Update product association	You can modify a manually added VMware product that is associated with vCenter Server. **NOTE** You cannot modify VMware products that are discovered automatically.
Delete product association	You can delete or dissociate a manually added VMware product that is associated with vCenter Server. **NOTE** You cannot delete or dissociate VMware products that are discovered automatically.
Create interoperability report	You can create an interoperability report between a vCenter Server release version and all products registered with the vCenter Server instance.

The following table lists the operations that are available in the update category.

Table 65: Update User Operations

Operation	Description
List updates	You can retrieve a list of all available vCenter Server updates. The list can contain minor, in-place, updates and major, migration-based, upgrades.
Get update info	You can retrieve detailed vCenter Server information about a specific update or upgrade.
Create pre-check report	You can create a vCenter Server pre-update compatibility check report for a pending update version. **NOTE** You can export and download the report in CSV format.

The following table lists the operations that are available in the reports category.

Table 66: Reports User Operations

Operation	Description
Get report	You can download the report generated by the interoperability and pre-check operations. For information about downloading the report, see the *API reference* documentation.

You can run life cycle management operations by using the vSphere Automation SDK or sending an HTTP request.

List Available Products and Manage Associated Products

You can automate the management of VMware products associated with vCenter Server by using the API.

- Verify that you have an active authenticated session with vCenter Server.

This procedure includes the operations that you can use to manage the product catalog and associated products.

1. Create a stub configuration.
2. Retrieve the product catalog.
3. Retrieve a list of VMware products associated with vCenter Server.
4. Associate a VMware product with vCenter Server.
5. Update a VMware product associated with vCenter Server.
6. Delete a product form the list of VMware products associated with vCenter Server.

List Available Updates

You can retrieve a list of available vCenter Server updates, details about the updates, and pre-check information by using the API.

- Verify that you have an active authenticated session with vCenter Server.

1. Create a stub configuration.
2. Retrieve a list of available vCenter Server updates.

   If there are available updates, you can retrieve details about the updates.
3. Retrieve pre-check information.

Retrieve a Report

You can retrieve a report generated by the interoperability and pre-check operations by using the API.

- Verify that you have an active authenticated session with vCenter Server.

1. Create a stub configuration.
2. Retrieve the report details.

Updating vCenter Server

vCenter Server provides interfaces to perform software updates.

Before applying updates, you must make sure that your environment is prepared for the vCenter Server software update process.

ApplyingvCenter Server Software Updates

You can automate the installation ofvCenter Server software updates to ensure that your system is stable and protected. Software updates can include security fixes, performance optimizations, and new features.

Security patches usually address vulnerabilities in third-party components and do not affect thevCenter Server functionality.vCenter Server bug fixes can introduce changes to the functionality without affecting the data format or database schema of the system.

Each update contains metadata with information about the updated content, for example, whether high-priority OS updates are included. The update metadata includes a list of components to be updated, the release date of the update, a list of fixed issues, time and disk space requirements, and information whether a reboot is required. The metadata can also contain a newvCenter Server version number, including a build number. In addition to the metadata, an update can contain optional components such as update scripts, new versions ofvCenter Server software components, and new versions of OS components.

vCenter Server can obtain software updates from either a URL or an ISO image. The URL can either point to the VMware Web repository or to a custom repository in which you upload the updates in ZIP format. To perform an update by using an ISO image, attach the image to the CD/DVD drive of thevCenter Server instance.

If you want to prevent issues related to the possibility of update installation failures, you should create a backup or take a snapshot of yourvCenter Server instance before you start the update process. A backup can also be useful when an update is successfully installed. For example, you might decide to revert to the previous version if you encounter any undesired system behavior related to functional changes in the new software version.

Table 67: User Operations

Operation	Description
Get state information	You can retrieve information about the update state.
Check for update	You can check whether a new update is available.
Get update information	You can retrieve information about the available updates.
Get update requirements	You can retrieve information about the update requirements.
Stage	You can initiate the download of the update. **NOTE** The check phase must have completed successfully before you can stage the update.
Get staging status	You can retrieve information about the status of the stage operation.

Table continued on next page

Continued from previous page

Operation	Description
	NOTE You must provide the task ID value that you received as a response when you initiated the stage operation.
Install	You can initiate the installation of the update. **NOTE** The update must be staged before you can install it.
Get installation status	You can retrieve information about the status of the install operation. **NOTE** You must provide the task ID value that you received as a response when you initiated the install operation.
Stage and install	You can initiate the download of the update and the installation starts when the download completes.

You can run software update operations by using thevSphere Automation SDK or sending an HTTP request.

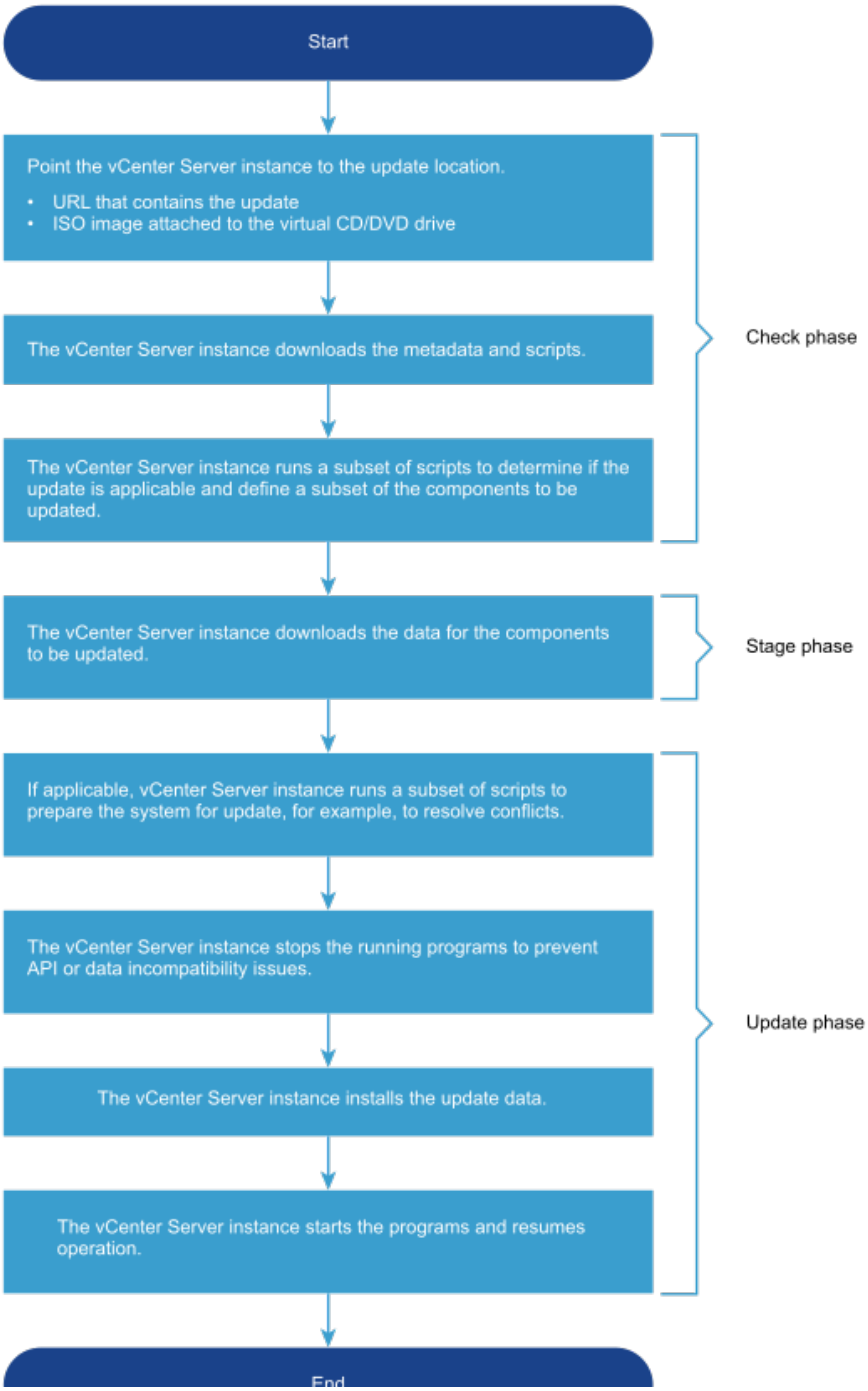
vCenter Server Software Update Workflow

The vCenter Server software update process consists of three major phases. In the first phase, the vCenter Server instance performs various checks, in the second phase it stages the update, and applies the update in the final phase.

To initiate the update process, you must choose whether the vCenter Server instance should obtain software updates from a URL or an ISO image. If you use an ISO image to update the vCenter Server instance, the image must remain attached to the CD/DVD drive of the instance during the stage and install operations.

The workflow in Figure: Update Process Workflow describes the standard steps of the update process.

Figure 103: Update Process Workflow



You can automate checks for new updates and staging of updates by using an update policy. For example, you can set an update policy to make the vCenter Server instance perform automatic checks for new updates at midnight every day. If there are new updates available, the vCenter Server instance can stage them automatically. Using an update policy reduces the waiting time by automating the first two phases and giving you the option to initiate only the update phase manually.

VMware vSphere Automation SDKs Programming Guide

VMware vSphere Automation SDKs Programming Guide provides information about how to use the VMware vSphere®Automation™ SDK to automate different vSphere management tasks.

At VMware, we value inclusion. To foster this principle within our customer, partner, and internal community, we have updated this guide to remove instances of non-inclusive language.

Intended Audience

This manual is intended for anyone who wants to develop applications for accessing and using vSphere features such as virtual machine management, tagging, content libraries, managing the life cycle of clusters with the vSphere Lifecycle Manager, vSphere Supervisor, and so on. The information is written for developers who have understanding of the targeted vSphere features and some experience with Java and Python programming languages.

Introduction to the vSphere Automation SDKs

The vSphere Automation SDKs bundle client libraries for accessing new vSphere features such as using content libraries and existing features such as virtual machine configuration and management, vSphere tags and attributes, and so on.

The vSphere Automation SDKs contain sample applications and API reference documentation for the vSphere services that are accessible via the vSphere Automation API endpoint. The vSphere Automation SDKs also provide sample code that demonstrates how you can establish a secure connection with the vSphere Automation API endpoint and access the available vSphere services for working with vSphere objects.

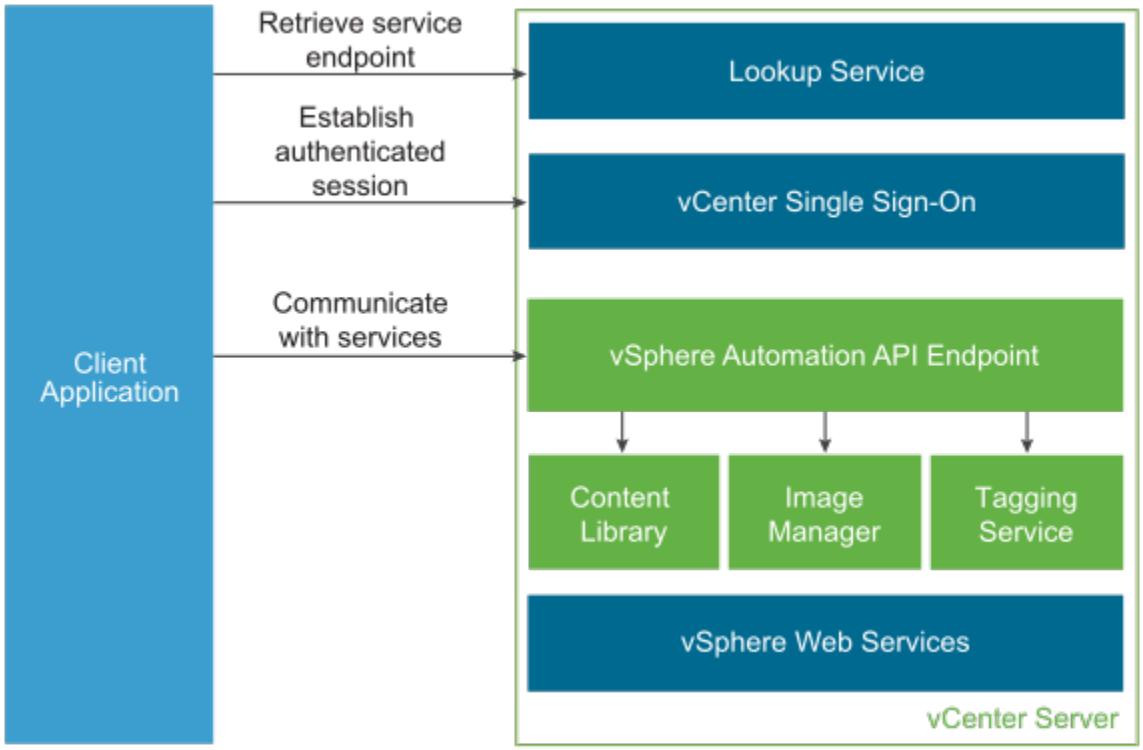
vSphere Automation SDKs support two programming languages for developing client applications for managing components in your virtual environment by accessing the vSphere Automation API services.

vSphere Automation SDK Overview

The vSphere Automation API provides a unified programming interface to vSphere Automation services that you can use through the SDKs provided for two programming languages. The vSphere Automation API provides a service-oriented architecture for accessing resources in the virtual environment by issuing requests to the vSphere Automation API endpoint.

Client applications use the vSphere Automation API to communicate with infrastructure and management vCenter Server services.

Figure 104: Communication Model Between a Client Application and the vSphere Automation API



Client applications use the Lookup Service to retrieve the vCenter Single Sign-On endpoint, the vSphere Automation API endpoint, and the endpoints of services that are exposed through the VMware vSphere® API. To access vSphere Automation services such as those for managing content libraries and vSphere tags, client applications send requests to the vSphere Automation API endpoint. By using the vCenter Single Sign-On service, client applications can either establish an authenticated vSphere Automation session, or authenticate individual requests to the vSphere Automation API endpoint.

Client applications can access services that are exposed through the vSphere Web Services API by using the VMware vSphere® Management SDK.

SDK Developer Setup

To start developing client applications that use the vSphere Automation API, you must download the desired SDK from the VMware GitHub Repository or from the respective vSphere Automation SDK landing page at https://developer.vmware.com and set up your development environment. You can find instructions for setting up your development environment in the README files contained in each vSphere Automation SDK.

API Explorer

The API Explorer is a VMware utility similar to the Managed Object Browser (MOB). Access the API Explorer at `https://<vcenter_ip_address_or_fqdn>/apiexplorer` or from the **API Explorer** tab of the **Developer Center** pane in vSphere Client. Use the utility to connect to the vSphere Automation API endpoint, browse through the available vSphere REST APIs, and make calls to your vCenter Server instance.

SDK Samples

The vSphere Automation SDKs provide code samples that demonstrate how the vSphere Automation API work and how they interoperate with the vSphere Web Services API. The code snippets included in this documentation are based on the samples in the vSphere Automation SDKs.

Supported Programming Languages

The vSphere Automation SDKs provide bindings for two different programming languages that let you build client applications on your preferred programming language.

- VMware vSphere Automation SDK for Java
- VMware vSphere Automation SDK for Python

What Are the Components and Services of the vSphere Environment

At the core of vSphere Automation is vSphere, which provides the virtualization layer of the software-defined data center. vCenter Server is a preconfigured virtual machine optimized for running the vCenter Server service and the vCenter Server components. vCenter Server service acts as a central administrator for ESXi hosts.

Authentication Services Installed with vCenter Server

The vCenter Server group of authentication services includes vCenter Single Sign-On, License Service, Lookup Service, and VMware Certificate Authority. The services installed with the vCenter Server appliance are common to the entire virtual environment. A vCenter Server can be connected to one or more vCenter Server instances. In a deployment that consists of more than one vCenter Server, the data of each service is replicated across all vCenter Server instances.

In the client applications that use the vSphere Automation API, you use the vCenter Single Sign-On and the Lookup Service on the vCenter Server to provide a range of functionality.

Authentication and Session Management

You use the vCenter Single Sign-On service to establish an authenticated session with the vSphere Automation API endpoint. You send credentials to the vCenter Single Sign-On service and receive a SAML token that you use to retrieve a session ID from the vSphere Automation API endpoint. Alternatively, you can access the vSphere Automation APIs in a sessionless manner. You must simply include the SAML token in every request that you issue to the vSphere Automation API endpoint.

Service Discovery

You use the Lookup Service to retrieve the endpoint URL for the vCenter Single Sign-On service on the vCenter Server, the location of the vCenter Server instances, and the vSphere Automation API endpoint.

Components Installed with vCenter Server

vCenter Server is a central administration point for ESXi hosts. The group of components installed when you install vCenter Server include the vCenter Server service, vSphere Client, VMware vSphere®Auto Deploy™, VMware vSphere®ESXi™ Dump Collector, VMware vSphere® Syslog Collector, and vSphere Lifecycle Manager service.

You can use the vSphere Automation API endpoint to access the following services running on vCenter Server.

Content Library

You can use content libraries to share virtual machines, vApps, and other files, such as ISO, OVA, and text files, across the software-defined data center. You can create, share, and subscribe to content libraries on the same vCenter Server instance or on a remote instance. Sharing content libraries promotes consistency, compliance, efficiency, and automation in deploying workloads at scale.

You can also create OVF and VM templates from virtual machines and vApps in hosts, resource pools, and clusters. You can then use the OVF and VM templates to deploy new virtual machines and vApps.

Starting with vSphere 7.0, you can edit the contents of a VM template. You can check out the library item that contains the VM template. After editing the VM template, check in the library item to save the changes to the virtual machine.

Virtual Machine

You can use the vSphere Automation APIs to create, configure, and manage the life cycle of virtual machines in your environment.

Starting with vSphere 7.0, you can also clone, create an instant clone, migrate, register, and unregister a virtual machine.

vSphere Lifecycle Manager

Starting with vSphere 7.0, the life cycle of ESXi hosts and clusters can be managed through the VMware vSphere®Lifecycle Manager™ feature. Based on the current state of the hosts in a cluster, you can easily create a desired software specification by using the contents of a software depot. Then you validate the desired software specification and you apply the specification on all hosts in the cluster.

vSphere Tags

With vSphere tags you can attach metadata to vSphere objects, and as a result, make it easier to filter and sort these objects. You can use the vSphere Automation APIs to automate the management of vSphere tags.

vSphere Supervisor

Starting with vSphere 7.0, you can enable vSphere Supervisor on an existing vSphere cluster in your environment. Create and configure namespaces on the Supervisors to run Kubernetes workloads in dedicated resource pools.

Retrieving Service Endpoints

To access services and resources in the virtual environment, client applications that use the vSphere Automation API must know the endpoints of vSphere Automation and vSphere services. Client applications retrieve service endpoints from the Lookup Service that runs on vCenter Server.

The Lookup Service provides service registration and discovery by using the vSphere Web Services API. By using the Lookup Service, you can retrieve endpoints of services on vCenter Server. The following endpoints are available from the Lookup Service.

- The vCenter Single Sign-On endpoint on vCenter Server. You can use the vCenter Single Sign-On service to get a SAML token and establish an authenticated session with a vSphere Automation API endpoint or a vCenter Server endpoint.
- The vSphere Automation API endpoint on vCenter Server. Through the vSphere Automation endpoint, you can make requests to vSphere Automation API services such as virtual machine management, Content Library, and Tagging.
- The vCenter Server endpoint. In case you want to retrieve service endpoints on a vCenter Server instance that is part of a vCenter Enhanced Linked Mode group, use the vCenter Server endpoint to get the node IDs of all linked instances. You can use the node ID of the specific vCenter Server instance to retrieve service endpoints on that instance.
- The vSphere Web Services API endpoint and endpoints of other vSphere Web services that run on vCenter Server.

Workflow for Retrieving Service Endpoints

The workflow that you use to retrieve service endpoints from the Lookup Service might vary depending on the endpoints that you need and their number. Follow this general workflow for retrieving service endpoints.

1. Connect to the Lookup Service on vCenter Server and service registration object so that you can query for registered services.
2. Create a service registration filter for the endpoints that you want to retrieve.
3. Use the filter to retrieve the registration information for services from the Lookup Service.
4. Extract one or more endpoint URLs from the array of registration information that you receive from the Lookup Service.

Filtering for Predefined Service Endpoints

The Lookup Service maintains a registration list of vSphere services. You can use the Lookup Service to retrieve registration information for any service by setting a registration filter that you pass to the `List()` function on the Lookup Service. The functions and objects that you can use with the Lookup Service are defined in the `lookup.wsdl` file that is part of the SDK.

Lookup Service Registration Filters

You can query for service endpoints through a service registration object that you obtain from the Lookup Service. You invoke the `List()` function on the Lookup Service to list the endpoints that you need by passing `LookupServiceRegistrationFilter`. `LookupServiceRegistrationFilter` identifies the service and the endpoint type that you can retrieve.

Optionally, you can include the node ID parameter in the filter to identify the vCenter Server instance where the endpoint is hosted. When the node ID is omitted, the `List()` function returns the set of endpoint URLs for all instances of the service that are hosted on different vCenter Server instances in the environment.

For example, a `LookupServiceRegistrationFilter` for querying the vSphere Automation service has these service endpoint elements.

Table 68: Service Registration Filter Parameters

Filter Types	Value	Description
`LookupServiceRegistrationServiceType`	`product= "com.vmware.cis"`	vSphere Automation namespace.
	`type="cs.vapi"`	Identifies the vSphere Automation service.
`LookupServiceRegistrationEndpointType`	`type="com.vmware.vapi.endpoint"`	Specifies the endpoint path for the service.
	`protocol= "vapi.json.https.public"`	Identifies the protocol that will be used for communication with the endpoint .

For information about the filter parameter of the available predefined service endpoints, see Filter Parameters for Predefined Service Endpoints.

Filter Parameters for Predefined Service Endpoints

Depending on the service endpoint that you want to retrieve, you provide different parameters to the `LookupServiceRegistrationFilter` that you pass to the `List()` function on the Lookup Service. To search for services on a particular vCenter Server instance, set the node ID parameter to the filter.

Table 69: Input Data for URL Retrieval for the Lookup Service Registration Filter

Service	Input Data	Value
vCenter Single Sign-On	product namespace	`com.vmware.cis`
	service type	`cs.identity`
	protocol	`wsTrust`
	endpoint type	`com.vmware.cis.cs.identity.sso`
vSphere Automation Endpoint	product namespace	`com.vmware.cis`
	service type	`cs.vapi`
	protocol	`vapi.json.https.public`
	endpoint type	`com.vmware.vapi.endpoint`

Table continued on next page

Continued from previous page

Service	Input Data	Value
vCenter Server	product namespace	`com.vmware.cis`
	service type	`vcenterserver`
	protocol	`vmomi`
	endpoint type	`com.vmware.vim`
vCenter Storage Monitoring Service	product namespace	`com.vmware.vim.sms`
	service type	`sms`
	protocol	`https`
	endpoint type	`com.vmware.vim.sms`
vCenter Storage Policy-Based Management	product namespace	`com.vmware.vim.sms`
	service type	`sms`
	protocol	`https`
	endpoint type	`com.vmware.vim.pbm`
vSphere ESX Agent Manager	product namespace	`com.vmware.vim.sms`
	service type	`cs.eam`
	protocol	`vmomi`
	endpoint type	`com.vmware.cis.cs.eam.sdk`

Connect to the Lookup Service and Retrieve the Service Registration Object

You must connect to the Lookup Service to gain access to its operations. After you connect to the Lookup Service, you must retrieve the service registration object to make registration queries.

1. Connect to the Lookup Service.
   a) Configure a connection stub for the Lookup Service endpoint, which uses SOAP bindings, by using the HTTPS protocol.
   b) Create a connection object to communicate with the Lookup Service.
2. Retrieve the Service Registration Object.
   a) Create a managed object reference to the Service Instance.
   b) Invoke the `RetrieveServiceContent()` method to retrieve the `ServiceContent` data object.
   c) Save the managed object reference to the service registration object.

   With the service registration object, you can make registration queries.

Retrieve Service Endpoints on vCenter Server Instances

You can create a function that obtains the endpoint URLs of a service on all vCenter Server instances in the environment. You can modify that function to obtain the endpoint URL of a service on a particular vCenter Server instance.

- Establish a connection with the Lookup Service.
- Retrieve a service registration object.

1. Create a registration filter object, which contains the following parts:
   - A filter criterion for service information
   - A filter criterion for endpoint information

Option	Description
Omit the node ID parameter	Retrieves the endpoint URLs of the service on all vCenter Server instances.
Include the node ID parameter	Retrieves the endpoint URL of the service on a particular vCenter Server instance.

2. Retrieve the specified service information by using the `List()` function.

Depending on whether you included the node ID parameter, the `List()` function returns one of the following results:

- A list of endpoint URLs for a service that is hosted on all vCenter Server instances in the environment.
- An endpoint URL of a service that runs on a particular vCenter Server instance.

Call the function that you implemented to retrieve service endpoints. You can pass different filter parameters depending on the service endpoints that you need. For more information, see Filter Parameters for Predefined Service Endpoints.

To retrieve a service endpoint on a particular vCenter Server instance, you must retrieve the node ID of that instance and pass it to the function. For information about how to retrieve the ID of a vCenter Server instance, see Retrieve a ID by Using the Lookup Service.

Retrieve a vCenter Server ID by Using the Lookup Service

You use the node ID of a vCenter Server instance to retrieve the endpoint URL of a service on that vCenter Server instance. You specify the node ID in the service registration filter that you pass to the `List()` function on the Lookup Service.

- Establish a connection with the Lookup Service.
- Retrieve a service registration object.

Managed services are registered with the instance name of the vCenter Server instance where they run. The instance name maps to a unique vCenter Server ID. The instance name of a vCenter Server system is specified during installation and might be an FQDN or an IP address.

1. List the vCenter Server instances.
2. Find the matching node name of the vCenter Server instance and save the ID.

Use the node ID of the vCenter Server instance to filter subsequent endpoint requests. You can use the node ID in a function that retrieves the endpoint URL of a service on a vCenter Server instance. For information about implementing such a function, see Retrieve Service Endpoints on Instances.

Retrieve a vSphere Automation API Endpoint on a vCenter Server Instance

Through the vSphere Automation API endpoint, you can access other vSphere Automation services that run on vCenter Server, such as Content Library and Tagging. To use a vSphere Automation service, you must retrieve the vSphere Automation API endpoint.

- Establish a connection with the Lookup Service.
- Retrieve a service registration object.
- Determine the node ID of the vCenter Server instance where the vSphere Automation service runs.
- Implement a function that retrieves the endpoint URL of a service on a vCenter Server instance.

1. Invoke the function for retrieving the endpoint URL of a service on a vCenter Server instance by passing filter strings that are specific to the vSphere Automation API endpoint.
2. Save the URL from the resulting single-element list.

Authentication Mechanisms

To perform operations in the vSphere environment, principals must authenticate to the vSphere Automation services. You can use the default vCenter Single Sign-On identity source or federate your vCenter Server to an external identity provider such as Microsoft Active Directory Federation Services, Okta, or Azure AD.

As a result from authenticating to the vSphere Automation services, client applications and users receive a session identifier. You can choose from several supported authentication mechanisms to obtain a session identifier that grants you access to the vSphere Automation services in the virtual environment.

Choosing Your Identity Source

First, you must decide whether to authenticate to vCenter Server by using the default identity provider, vCenter Single Sign-On, or leverage your enterprise configuration to an external identity provider.

Default or External Identity Provider

vCenter Server acts as the default identity provider, by using its built-in vCenter Single Sign-On service to authenticate users and service accounts.

In vSphere 7.0 and later, you can delegate vSphere identity management to an external identity provider and take advantage of increased security and multifactor authentication. In this scenario, vCenter Single Sign-On is replaced by an external identity provider as the entity that performs principals' authentication.

Identity provider federation enhances the security and compliance of your applications and provides access to flexible benefits such as multifactor authentication (MFA) and automatic account synchronization.

You can federate vCenter Server authentication to:

- AD FS (starting in vSphere 7.0)
- Okta (starting in vSphere 8.0 Update 1)
- Azure AD (starting in vSphere 8.0 Update 2)

For more information, see Identity Provider Federation for vCenter Server.

Basic or Token-Based Authentication

You can use basic and token-based authentication to authenticate principals to the vSphere Automation services. VMware encourages you to use token-based authentication as it provides more security and flexibility for your applications and users.

Basic Authentication

Basic authentication passes a user name and password to vCenter Server.

By default, principals can use the vCenter Single Sign-On user name and password to connect to the vSphere Automation endpoint. The vSphere Automation endpoint checks whether the user name and password are present in the VMware Directory Service. On success, the system returns a session identifier for the vSphere Automation endpoint.

If your vCenter Server is federated to an external identity provider, you can use basic authentication through the OAuth 2.0 Password grant type.

IMPORTANT
VMware encourages you to refrain from using basic authentication and adopt token-based authentication, as it is more secure and provides more flexibility.

Token-Based Authentication

Token-based authentication involves the use of an encrypted token that provides authentication and authorization data to the vSphere Automation endpoint. The vCenter Server token complies with the Security Assertion Markup Language (SAML) specification, an XML-based schema for communicating authentication data.

To acquire a SAML token, client applications issue a token request to vCenter Single Sign-On. Client applications then send the SAML token to the vSphere Automation API endpoint in exchange for a session identifier.

Starting with vSphere 7.0, you can use token-based authentication for your federated vCenter Server through a 3-step process:

1. Principals use the various OAuth 2.0 grant types with the external identity provider to obtain identity and access tokens in JSON Web Token (JWT) format.
2. Principals exchange the JWT tokens for a vCenter Single Sign-On SAML token.
3. Principals use the SAML token to obtain a session identifier from the vSphere Automation endpoint.

Authentication Terminolgy

To use the vSphere programming features effectively, you must understand a set of specific authentication terms and concepts.

Table 70: vCenter Single Sign-On Glossary

Term	Definition
Principal	An entity that can be authenticated, such as a user.
Identity Provider	A service that manages identity sources and authenticates principals. Examples: Microsoft Active Directory Federation Services (AD FS) and vCenter Single Sign-On.
Identity Source (Directory Service)	Stores and manages principals. Principals consist of a collection of attributes about a user or a service account such as name, address, email, and group membership. Examples: Microsoft Active Directory and VMware Directory Service (vmdir).
Authentication	The means of determining whether someone or something is, in fact, who or what it declares itself to be. For example, users are authenticated when they provide their credentials, such as smart cards, user name and correct password, and so on.
Authorization	The process of verifying what objects principals have access to.
Token	A signed collection of data comprising the identity information for a given principal. A token might include not only basic information about the principal such as email address and full name, but also, depending on the token type, the principal's groups and roles.

Table continued on next page

Continued from previous page

Term	Definition
vmdir	VMware Directory Service. The internal (local) LDAP repository in vCenter Server that contains user identities, groups, and configuration data.
OAuth 2.0	An open authorization standard that enables the exchange of information among principals and web services without exposing principals' credentials.
OpenID Connect (OIDC)	Authentication protocol based on OAuth 2.0 that augments OAuth with user-identifying information. It is represented by the ID token that the authorization server returns together with the access token during OAuth authentication. vCenter Server uses OIDC capabilities when interacting with Active Directory Federation Services (AD FS), Okta, Microsoft Entra ID, and PingFederate.
System for Cross-domain Identity Management (SCIM)	The standard for automating the exchange of user identity information between identity domains or IT systems.
VMware Identity Services	Starting in version 8.0 Update 1, VMware Identity Services is a built-in container within vCenter Server that you can use for identity federation to external identity providers. It serves as an independent identity broker within vCenter Server and comes with its own set of APIs. Currently, VMware Identity Services support Okta, Microsoft Entra ID, and PingFederate as external identity providers.
Tenant	A VMware Identity Services concept. A tenant provides a logical separation of data from other tenants' data in one and the same virtual environment.
JSON Web Token (JWT)	A token format defined by the OAuth 2.0 specification. A JWT token carries authentication and authorization information about a principal.
Relying party	A relying party "relies" on the authorization server, VMware Identity Services or AD FS, for identity management. For example, through federation, vCenter Server establishes relying party trust to VMware Identity Services or AD FS.
Security Assertion Markup Language (SAML)	An XML-based open standard for exchanging authentication and authorization data between parties that is used by vCenter Server. Principals obtain a SAML token from vCenter Single Sign-On and then send it to the vSphere Automation API endpoint for a session identifier.

Create a vSphere Automation Session with User Credentials

You can use your vCenter Single Sign-On credentials with basic authentication to establish a session with a vCenter Server system.

- Retrieve the vSphere Automation endpoint URL from the Lookup Service.
- Verify that you have valid user credentials for the vCenter Single Sign-On identity store.

The single-factor basic authentication is the default method to authenticate to vCenter Server. You connect to the vSphere Automation SDKs with your vCenter Single Sign-On user name and password. The vSphere Automation endpoint checks whether the user credentials are present in vmdir, a vCenter Single Sign-On component. On success, the system returns a session identifier that you can use for subsequent API calls.

1. Create a connection stub by specifying the vSphere Automation endpoint URL and the message protocol to be used for the connection.
2. Create a stub configuration instance and set the specific security context to be used.

   The security context object uses the user name and password that are used for authenticating to the vCenter Single Sign-On service.
3. Create a `Session` stub that uses the stub configuration instance.
4. Call the create operation on the `Session` stub to create an authenticated session to the vSphere Automation endpoint.
   The operation returns a session identifier.
5. Create a security context instance and add the session ID to it.
6. Update the stub configuration instance with the session security context.

   IMPORTANT
   VMware encourages you to refrain from using basic authentication and adopt token-based authentication, as it is more secure and provides more flexibility.

You can use the authenticated session to access vSphere Automation services. For more information about creating stubs to the vSphere Automation services, see Accessing Services.

Retrieve a SAML Token

The vCenter Single Sign-On service provides authentication mechanisms for securing the operations that your client application performs in the virtual environment. Client applications use SAML security tokens for authentication.

Verify that you have the vCenter Single Sign-On URL. You can use the Lookup Service on vCenter Server to obtain the endpoint URL. For information about retrieving service endpoints, see Retrieving Service Endpoints.

Client applications use the vCenter Single Sign-On service to retrieve SAML tokens. For more information about how to acquire a SAML security token, see the *`vCenter Single Sign-On Programming Guide`* documentation.

1. Create a connection object to communicate with the vCenter Single Sign-On service.

   Pass the vCenter Single Sign-On endpoint URL, which you can get from the Lookup Service.
2. Issue a security token request by sending valid user credentials to the vCenter Single Sign-On service on vCenter Server.

The vCenter Single Sign-On service returns a SAML token.

You can present the SAML token to the vSphere Automation API endpoint or other endpoints, such as the vSphere Web Services endpoint. The endpoint returns a session ID and establishes a persistent session with that endpoint. Each endpoint that you connect to uses your SAML token to create its own session.

Create a vSphere Automation Session with a SAML Token

To establish a vSphere Automation session, you create a connection to the vSphere Automation API endpoint and then you authenticate with a SAML token to create a session for the connection.

- Retrieve the vSphere Automation endpoint URL from the Lookup Service.
- Obtain a SAML token from the vCenter Single Sign-On service.

1. Create a connection by specifying the vSphere Automation API endpoint URL and the message protocol to be used for the connection.

   NOTE
   To transmit your requests securely, use `https` for the vSphere Automation API endpoint URL.

2. Create the request options or stub configuration and set the security context to be used.

   The security context object contains the SAML token retrieved from the vCenter Single Sign-On service. Optionally, the security context might contain the private key of the client application.

3. Create an interface stub or a REST path that uses the stub configuration instance.

   The interface stub corresponds to the interface containing the method to be invoked.

4. Invoke the session `create` method.
   The service creates an authenticated session and returns a session identification cookie to the client.
5. Create a security context instance and add the session ID to it.
6. Update the stub configuration instance with the session security context.

Use the updated stub configuration with the session ID to create a stub for the interface that you want to use. Method calls on the new stub use the session ID to authenticate.

Create a Web Services Session

To develop a complex workflow, you might need to send requests to vSphere Web Services running in your virtual environment. To achieve this, you access the vSphere Web Services API by using the Web Services endpoint.

- Retrieve the vSphere Web Services endpoint URL from the Lookup Service.
- Obtain a SAML token from the vCenter Single Sign-On service.

The vSphere Web Services API also supports session-based access. To establish an authenticated session, you can send the SAML token retrieved from the vCenter Single Sign-On service to a vSphere Web Service. In return, you receive a session identifier that you can use to access the service. For more information about accessing Web Services and additional examples, see the *`vSphere Web Services SDK Programming Guide`* documentation.

1. Connect to the vSphere Web Services endpoint.
2. Send the SAML token to a specific Web service to create an authenticated session.
3. Add the retrieved session ID to the service content object.

   The Service Content object gives you access to several server-side managed objects that represent vSphere services and components.

Identity Provider Federation for vCenter Server

Starting with vSphere 7.0, you can replace through federation your built-in vCenter Server identity source with an external identity source. Depending on your vCenter Server version, you can use external identity providers such as AD FS, Azure AD, or Okta.

With identity federation, you can use the same corporate identity source for your vCenter Server that you use for your other federated desktop and cloud applications.

vCenter Server Identity Provider Federation Basics

In vSphere 7.0 and later, vCenter Server supports federated authentication. In this scenario, when a user logs in to vCenter Server, vCenter Server redirects the user login to the external identity provider. The user credentials are no

longer provided to vCenter Server directly. Instead, the user provides credentials to the external identity provider. vCenter Server trusts the external identity provider to perform the authentication. In the federation model, users never provide credentials directly to any service or application but only to the identity provider. As a result, you "federate" your applications and services, such as vCenter Server, with your identity provider.

Why Is Identity Provider Federation Useful

Federating vCenter Server to an enterprise identity provider alleviates the burden of identity management and provides flexible options such as multifactor authentication (MFA), automatic registration and termination of users across services, and many more. Identity provider federation uses token-based authentication and minimizes the risk of bad actors acquiring protected credentials such as user names and passwords. Identity provider federation also helps your organization with compliance as various international standards require MFA to guarantee data security. In addition, with identity provider federation, you can automate vCenter Server user management because you utilize the users and groups from your enterprise identity source.

vCenter Server External Identity Provider Support

vCenter Server supports the following external identity providers:

- AD FS (vSphere 7.0 and later)
- Okta (vSphere 8.0 Update 1 and later)
- Microsoft Entra ID, formerly called Azure AD (vSphere 8.0 Update 2 and later)
- PingFederate (starting in vSphere 8.0 Update 3)

Federation to Microsoft Active Directory Federation Services (AD FS)

In vSphere 7.0 and later, you can activate identity federation to Microsoft Active Directory Federation Services (AD FS). In this scenario, vCenter Server federates directly to the enterprise identity provider, AD FS, without the use of an authentication intermediary service.

You can configure federation to AD FS with the help of the vSphere Automation API. For more information, see Federate vCenter Server to Microsoft Active Directory Federation Services (AD FS).

Federation to Okta and Azure AD Through VMware Identity Broker - vCenter Server

Starting from vSphere 8.0 Update 1, you can activate federation to Okta as the identity provider. Starting from vSphere 8.0 Update 2, you can activate federation to Azure AD as the identity provider. Both configurations use VMware Identity Broker - vCenter Server, an authentication intermediary that functions as a built-in container within vCenter Server. With VMware Identity Broker - vCenter Server, you can configure principals to authenticate to vCenter Server by using an external identity provider.

vCenter Server can support the default `vsphere.local` identity source and only one configured external identity provider. You cannot use multiple external identity providers.

Once you set up your federated authentication, you can use OAuth for login to vCenter Server.

> **NOTE**
> You can federate vCenter Server to Okta or Azure AD by using the vSphere Client. For more information, see Configure vCenter Server Identity Provider Federation for Okta and Configure vCenter Server Identity Provider Federation for Microsoft Entra ID in the *vSphere Authentication Guide*.

What is OAuth 2.0

OAuth 2.0 is an open authorization framework that allows information exchanges among web services without exposing sensitive data such as user names and passwords. OAuth 2.0 boosts security and prevents malicious actors from stealing users' credentials over the Internet. OAuth 2.0 is an open framework and can be customized according to the organizational requirements.

You can use different OAuth 2.0 grant types to authenticate your applications to vCenter Server. The OAuth grant types represent different methods by which a principal can obtain an access token from an authorization server. In the vSphere implementation, the access token is used to obtain a SAML token and authenticate principals to vCenter Server.

OAuth 2.0 Grant Types

Password

In the Password grant type, the client application exchanges the user's credentials, user name and password, for an access token, and optionally an ID token (JWT tokens), from the authorization server. OAuth 2.0 has been designed to prevent the exchange of user credentials, so the Password grant type must be avoided whenever possible, especially for third-party application authentication. The latest *`OAuth 2.0 Security Best Current Practice`* recommends avoiding the Password grant type altogether. As a single-factor method, the Password grant type is incompatible with the requirements of multifactor authentication (MFA). For more information, see Authenticate to an AD FS-Federated vCenter Server by Using the Password Grant Type.

> **IMPORTANT**
> The Password grant type is disallowed by the latest *`OAuth 2.0 Security Best Current Practice`*. The Password grant is excluded entirely from OAuth 2.1.

Authorization Code

In the Authorization Code grant type, the client application uses an authorization code to obtain JWT tokens from the authorization server. This grant type requires a user to verify an authentication request through a web browser. The Authorization Code grant type is compatible with multi-factor authentication and guarantees a substantial degree of security. This grant type is appropriate for regular web applications running on a server. For more information, see Authenticate to an AD FS - Federated vCenter Server by Using the Authorization Code Grant Type.

Client Credentials

The Client Credentials grant type permits web services to authenticate to other web services without the involvement of a user. This grant type is useful for applications that are resource owners and can take advantage of headless, or backend automation, when a physical user is not present. For example, a client application might use this grant type to authenticate and fetch details about a virtual environment.

Refresh Token

The Refresh Token grant type is used by clients to exchange an expiring access token for a refresh token that can be used to receive a new access token. With this grant type, a client application can fetch a valid access token without asking the user to log in again. It allows client applications to provide a seamless user experience without further interaction with the user after timeouts.

What Is OpenID Connect (OIDC)

The vSphere implementation of OAuth 2.0 uses the OpenID Connect (OIDC) authentication protocol. With OIDC, the authorization server returns an ID token containing data about who the users are and what permissions they have on the resource server. As a result, vCenter Server can identify users and grant them the respective permissions that are defined on the authorization server.

AD FS Federation Workflows

You can federate your vCenter Server to Microsoft Active Directory Federation Services (AD FS) and use the OAuth 2.0 grant types to authenticate.

Federate vCenter Server to Microsoft Active Directory Federation Services (AD FS)

You can federate vCenter Server to Microsoft Active Directory Federation Services (AD FS) as an external identity provider by using the vCenter Server `Identity Providers` service.

NOTE
This process of configuring an AD FS identity provider requires that you have administrative access to both your vCenter Server and your AD FS server. During the configuration process, you enter information first in your vCenter Server, then in your AD FS server, then in your vCenter Server.

Active Directory Federation Services requirements:

- AD FS for Windows Server 2016 or later must already be deployed.
- AD FS must be connected to Active Directory.
- An Application Group for vCenter Server must be created in AD FS as part of the configuration process. See the VMware knowledge base article at https://kb.vmware.com/s/article/78029.
- An AD FS server certificate (or a CA or intermediate certificate that signed the AD FS server certificate) that you add to the Trusted Root Certificates Store.
- You have created a vCenter Server administrators group in AD FS that contains the users you want to grant vCenter Server administrator privileges to.

For more information about configuring AD FS, see the Microsoft documentation.

vCenter Server and other requirements:

- vSphere 7.0 or later
- vCenter Server must be able to connect to the AD FS discovery endpoint, and the authorization, token, logout, JWKS, and any other endpoints advertised in the discovery endpoint metadata.
- You need the **VcIdentityProviders › Manage** privilege to create, update, or delete a vCenter Server Identity Provider that is required for federated authentication. To limit a user to view the Identity Provider configuration information only, assign the **VcIdentityProviders › Read** privilege.

1. Authenticate to the vSphere Automation API endpoint and establish a session.
2. Add your AD FS root CA certificate to the Trusted Root Certificates Store.

   See Managing Certificates.

3. Add the identity provider by using the service.
4. Configure vCenter Server permissions for Active Directory users or groups in your AD FS environment.

   You can do this in two ways:

   a)
   - Add a user from your AD FS environment to a group in vCenter Server.
   - Configure Global Permissions for an AD FS user.

   NOTE
   In vSphere 8.0 and later, you cannot configure permissions through the vSphere Automation API. Instead, you use either the vSphere Client or the vSphere Web Services API. For more information, see the *vSphere Authentication Guide* or the *vSphere Web Services SDK Programming Guide*.

5. Copy the two redirect URIs from the Identity Provider Configuration page in the vSphere Client and add them to your AD FS Application Group.

   NOTE
   You must do this step to enable logging in to vCenter Server through AD FS by using the vSphere Client.

You configured vCenter Server to use AD FS as the identity provider.

Authenticate to an AD FS-Federated vCenter Server by Using the Password Grant Type

Authenticate by Using the Password Grant Type
If your vCenter Server is federated to AD FS, you can authenticate with the OAuth 2.0 Password grant type.

- Verify that your vCenter Server is federated to AD FS. For more information, see *Federate vCenter Server to Active Directory Federation Services (AD FS).*
- You must have an AD FS account and user credentials with the necessary permissions to view and manage vCenter Server.
- You must register an OAuth client for your application on the authorization server (AD FS).

You can use the Password grant type to exchange user credentials for an access token and an ID token from the authorization server. From the user's perspective, the password grant type functions exactly as basic authentication with the only difference that you use your AD FS, and not your local vCenter Single Sign-On credentials.
The Password grant type is possible among native apps but is not recommended for authentication to third-party apps. Modern security best practices require that primary user credentials such as passwords do not leave the native API environment. Therefore, it is more secure to use other OAuth 2.0 grant types for authentication to third-party apps.
To use the Password grant type with AD FS, you provide your AD FS user name and password to the `Common Infrastructure Services (CIS) Session` endpoint.
In the background, the client application calls the authorization server (AD FS) and obtains an access token and an ID token in JWT format. vCenter Single Sign-On converts the JWT tokens into a SAML token which is used to obtain a session identifier and authenticate your application to vCenter Server.

> **IMPORTANT**
> The Password grant type is disallowed by the latest *OAuth 2.0 Security Best Current Practice*.
> The Password grant is excluded entirely from OAuth 2.1.

Java Example of Creating a vSphere Automation API Session with User Credentials

This example is based on the code in the `VapiAuthenticationHelper.java` sample.

> **NOTE**
> For a complete and up-to-date version of the Java sample code, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...


this.stubFactory = createApiStubFactory(server, httpConfig);


// Create a security context for username/password authentication

SecurityContext securityContext =

        SecurityContextFactory.createUserPassSecurityContext(

            username, password.toCharArray());


// Create a stub configuration with username/password security context

StubConfiguration stubConfig = new StubConfiguration(securityContext);
```

```
// Create a session stub using the stub configuration.
Session session =
        this.stubFactory.createStub(Session.class, stubConfig);

// Login and create a session
char[] sessionId = session.create();

// Initialize a session security context from the generated session id
SessionSecurityContext sessionSecurityContext =
        new SessionSecurityContext(sessionId);

// Update the stub configuration to use the session id
stubConfig.setSecurityContext(sessionSecurityContext);

/*
 * Create a stub for the session service using the authenticated
 * session
 */
this.sessionSvc =
        this.stubFactory.createStub(Session.class, stubConfig);

VM vmService = this.stubFactory.createStub(VM.class, stubConfig);
```

Python Example of Creating a vSphere Automation Session with Username and Password

This example illustrates how to create an authenticated session to the vSphere Automation API by using basic authentication in Python.

```
import requests
import urllib3
from vmware.vapi.vsphere.client import create_vsphere_client
session = requests.session()
session.verify = False
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
```

```
vsphere_client = create_vsphere_client(hostname, username, password, session=session)

# Interface factory object:

stub_config = vsphere_client._stub_config
```

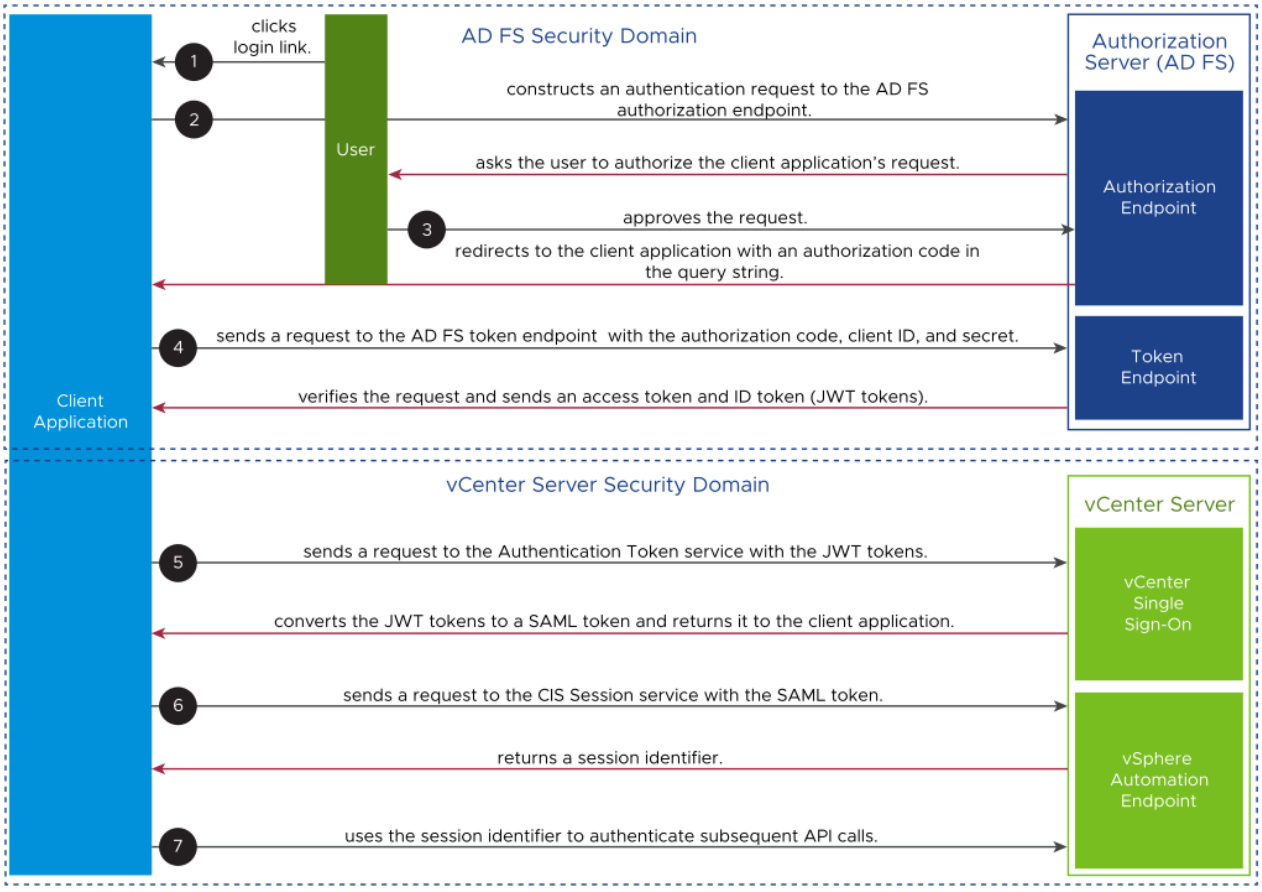
Authenticate to an AD FS-Federated vCenter Server by Using the Authorization Code Grant Type

Authenticate by Using the Authorization Code Grant Type
If your vCenter Server is federated to an enterprise identity provider, such as Active Directory Federation Services (AD FS), you can authenticate your client applications by using the OAuth 2.0 Authorization Code grant type.

- Verify that your vCenter Server is federated to AD FS. For more information, see *Federate vCenter Server to Active Directory Federation Services (AD FS).*
- You must have an AD FS account and user credentials with the necessary permissions to view and manage vCenter Server.
- You must register an OAuth client for your application on the authorization server (AD FS).

You can use the Authorization Code grant type if you want to authenticate client apps to vCenter Server through multifactor authentication (MFA). This grant type is applicable for regular web applications that involve a human user interacting with a server.

Figure 105: Authentication to an AD FS-Federated vCenter Server Through the Authorization Code Grant Type



A principal authenticates to vCenter Server through the Authorization Code grant type as follows.

1. The user clicks **Login** within the client application.
2. The client application constructs a request to the AD FS authorization endpoint.

   To retrieve the authorization endpoint, you must query the AD FS discovery endpoint at, for example, `https://<your_adfs_server.com>/adfs/.well-known/openid-configuration`.

   The client application's request must include the application's client ID and a redirect URI that tells the authorization server where to redirect the user after approving the request.

   As a result, the authorization server opens a prompt page asking the user to authorize the client application's request.
3. The user confirms the request.
   The browser is redirected back to the client application with an authorization code in the query string.
4. The client application sends a request to the AD FS token endpoint with the authorization code, client ID, and secret.
   The authorization server verifies the request and sends an access token and an ID token (JWT tokens).
5. The client application sends the JWT tokens to vCenter Single Sign-On by using the .
   vCenter Single Sign-On returns a SAML token.
6. The client application calls the `Common Infrastructure Services (CIS) Session` service.

   By using the `CIS Session` service, principals can manage session tokens - including creation and deletion of session identifiers, and obtaining information about sessions.

   a) Base-64 decode the SAML token, compress it, and then Base-64 encode it again.
   b) Call the with the SAML token.

   On success (status code 201), the vSphere Automation endpoint returns a session identifier in the response.
7. To authenticate, the client application uses the session identifier in subsequent API calls to the vSphere Automation API endpoints.

VMware Identity Broker - vCenter Server Workflows

You can federate your vCenter Server to an external identity provider through VMware Identity Broker - vCenter Server. It is a built-in container within vCenter Server that enables federation to an external identity provider such as Okta or Azure AD.

What Is VMware Identity Broker - vCenter Server

VMware Identity Broker - vCenter Server is VMware's specialized authentication solution that allows you to federate your apps to external identity providers such as Okta or Azure AD. VMware Identity Broker - vCenter Server functions as a container built-in within your vCenter Server. VMware Identity Broker - vCenter Server comes with its own API Interface that is separate from the vSphere Automation API and the vSphere Web Services API.

You can use VMware Identity Broker - vCenter Server to federate to:
- Okta (starting in vSphere 8.0 Update 1)
- Azure AD (starting in vSphere 8.0 Update 2)

Configure vCenter Server Identity Provider Federation to Okta or Azure AD

To configure your vCenter Server to point to Okta or Azure AD as the identity provider, you must use the Okta or Azure AD interface and the vSphere Client. This integration uses VMware Identity Broker - vCenter Server. For more information, see Configure vCenter Server Identity Provider Federation for Okta and Configure vCenter Server Identity Provider Federation for Microsoft Entra ID from the *vSphere Authentication Guide*.

OAuth Authentication to Your Federated vCenter Server, App, or Script

Once configured, you can use the vSphere Automation and the VMware Identity Broker - vCenter Server APIs to authenticate to your federated vCenter Server, application, or script.

You can use the following OAuth 2.0 grant types:

- Password (not recommended)
- Authorization Code
- Client Credentials
- Refresh Token

Federate Apps or Scripts to an External Identity Provider by Using Your VMware Identity Broker - vCenter Server Configuration

Configure a client application or script for authentication by using an already existing vCenter Server identity provider configuration on VMware Identity Broker - vCenter Server .

Okta and Azure AD requirements:

- You are a customer of Okta or Microsoft and have a dedicated Okta domain space or an Azure AD account.
- You have identified the Okta or Azure AD users and groups that you want to share with your app.

vCenter Server and other requirements:

- vSphere 8.0 Update 1 or later, with the VMware Identity Broker - vCenter Server activated (they are activated by default).
- A vCenter Server that is federated to Okta or Azure AD. (You must know the IP address or the FQDN of the vCenter Server.)
- The users and groups from the identity provider are provisioned in your vCenter Server.
- You must know the tenant that you want to use for your client app or script. The default tenant that is created during vCenter Server federation is called `CUSTOMER`.
- You must have the **VcIdentityProviders › Manage** privilege.

You can leverage the identity provider configuration you created for vCenter Server for other apps or scripts that you want to federate to Okta or Azure AD.
For this automation scenario you must use services from the vSphere Automation API and the VMware Identity Broker - vCenter Server API. The VMware Identity Broker - vCenter Server API endpoints are located also on your vCenter Server but are taxonomically separated from the vSphere Automation API endpoints.

1. Get a session identifier for the vSphere Automation API.

   For more information see Authentication Mechanisms.

2. Get a tenant admin client token.

   The default `tenant` that is created during the vCenter Server federation is called `CUSTOMER`. You can use this tenant for your app or script.

   In the response, you get an access token in JWT format that you use to authenticate to VMware Identity Broker - vCenter Server.

3. Create an OAuth 2.0 client for your app.

   To authenticate to the external identity provider, you must create an OAuth 2.0 client for your app on VMware Identity Broker - vCenter Server. You can do this by using the Create Broker OAuth 2 Client service of the VMware Identity Broker - vCenter Server API.

   For authentication, you must use the tenant admin client token obtained in the previous step.

Example request:

```
curl --location --request POST 'https://<vcenter_server_fqdn>/acs/t/<tenant>

/broker/oauth2-clients' \

--header 'Content-Type: application/
vnd.vmware.horizon.manager.accesscontrol.broker.oauth2client.with.rule.sets+json' \

--header 'Accept: application/
vnd.vmware.horizon.manager.accesscontrol.broker.oauth2client.with.rule.sets+json' \

--header 'Authorization: HZN <my_admin_client_token>' \

--data-raw '{

    "client_id": "<my_app_client_id>",

    "grant_types": [

        "refresh_token",

        "client_credentials",

        "password",

        "authorization_code"

    ],

    "scope": [

        "email",

        "profile",

        "user",

        "openid",

        "group"

    ],

    "secret": "<my_app_client_secret>",

    "redirect_uris": [

        "https://<my_app_redirect_uri1>",

        "https://<my_app_redirect_uri2>"

    ]

}'
```

On success (status code 201), the operation returns the data about the new OAuth 2.0 client.

Authenticate your app or script to the external identity provider by using the OAuth 2.0 grant types.

Obtain JWT Tokens from VMware Identity Broker - vCenter Server by Using the Password Grant Type

Obtain JWT Tokens by Using the Password Grant Type

If your vCenter Server or other application is federated to an external identity provider, such as Okta or Azure AD, through VMware Identity Broker - vCenter Server, you can obtain an access and ID token in JWT format by using the OAuth Password grant type.

- Verify that your vCenter Server or client app is federated to an external identity provider through VMware Identity Broker - vCenter Server.
- You must create an OAuth client for your vCenter Server or client app on VMware Identity Broker - vCenter Server. You must know the client id, client secret, and the redirect URIs for the OAuth client.
- You must know the tenant name for the configuration you want to use. The default tenant that is created during vCenter Server federation is called *CUSTOMER*.
- You must have an Okta or Azure AD account and user credentials with the necessary permissions to view and manage vCenter Server.

You can use the Password grant type to exchange user credentials for an access token and an ID token from the authorization server. From the user's perspective, the password grant type functions similarly to local domain authentication with the difference that principals use their external identity provider, and not their local vCenter Single Sign-On credentials, to authenticate.
The Password grant type involves the client collecting the principal's user name and password directly and exchanging them for an access token from the authorization server. From a security perspective, handling user credentials directly increases the risk of exposure and compromise. From a user experience perspective, users might be reluctant to expose their credentials to third-party apps, as it goes against the principle of separating authentication from authorization. Therefore, using the OAuth 2.0 Password grant type is not recommended.

IMPORTANT
The Password grant type is disallowed by the latest *OAuth 2.0 Security Best Current Practice*. The Password grant is excluded entirely from OAuth 2.1.

1. Obtain an access and ID token.

   Send a request to the VMware Identity Broker - vCenter ServerGet Access Token endpoint with the required grant type, your app's client ID and secret, and the user name and password of a registered user.

   In the authorization header, use the base-64 encoded value of your app's client ID and secret separated by a colon (`client_id:secret`).

   The content type must be *x-www-form-urlencoded*.

   Example request:

```
curl --location --request POST 'https://<vcenter_server_fqdn>/acs/t/<tenant>/token' \
--header 'Authorization: Basic <my_app_client_id>:<my_app_client_secret>' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=password' \
--data-urlencode 'username=<my_user_name>' \
--data-urlencode 'password=<my_password>'
```

   The authorization server returns the JWT tokens corresponding to the grant type: access token , ID token, and refresh token.

Exchange the JWT tokens for a vCenter Server SAML token that you can use to establish an authentication session. For more information, see Exchange JWT Tokens for a SAML Token and Obtain a Session Identifier.

Obtain JWT Tokens from VMware Identity Broker - vCenter Server by Using the Authorization Code Grant Type

Obtain JWT Tokens by Using the Authorization Code Grant Type
If your vCenter Server or other application is federated to an external identity provider, such as Okta or Azure AD, through VMware Identity Broker - vCenter Server, you can obtain an access and ID token in JWT format by using the OAuth 2.0 Authorization Code grant type.

- Verify that your vCenter Server or client app is federated to an external identity provider through VMware Identity Broker - vCenter Server.
- You must create an OAuth client for your vCenter Server or client app on VMware Identity Broker - vCenter Server. You must know the client id, client secret, and the redirect URIs for the OAuth client.
- You must know the tenant name for the configuration you want to use. The default tenant that is created during vCenter Server federation is called *CUSTOMER*.
- You must have an Okta or Azure AD account and user credentials with the necessary permissions to view and manage vCenter Server.

The Authorization Code grant type is appropriate for regular web applications that involve a human user interacting with a server, especially if you want to activate multifactor authentication (MFA). The Authorization Code grant type protects user credentials as they do not leave the API environment.

1. Obtain an authorization code.
   a) Construct a query request in the browser with the following parameters.

client_id	The client identifier you obtained when you registered the OAuth 2.0 client on VMware Identity Services.
redirect_uri	The callback endpoint in your application that must receive the authorization code. It must be one of the `redirect_uris` defined in your OAuth 2.0 client registration. When sending the `redirect_uri` as a URL parameter it must be URL-encoded.
state	A random string that your application generates and that is sent back as a parameter during the URI redirect. Serves as a security mechanism as per the OAuth 2.0 specification.
response_type	Specifies what must be included in the authorization response. For the authorization code grant type, use *code*.

   b) Send the query request to the VMware Identity Broker - vCenter ServerAuthorize endpoint.

   For example:

   ```
   https://<vcenter_server_fqdn>/acs/t/<tenant>/authorize?client_id=<my_app_client
   _id>&redirect_uri=https://<my_app_redirect_uri1>&state=<generated_random_string>
   &response_type=code
   ```

   VMware Identity Broker - vCenter Server opens a dialog box asking the user to authenticate.

   c) The user authenticates to the identity provider.

   The authentication mechanism depends on the specific user configuration on the external identity provider website. It could be basic authentication, biometrics, multifactor authentication, and so on.

   The authorization server directs the user to the client app's redirect URI with the authorization code in the query string.

2. Obtain an access and ID token.

   Send a request to the VMware Identity Broker - vCenter ServerGet Access Token endpoint containing the required grant type, the authorization code, the redirect URI, and your app's client ID and secret.

   In the authorization header, use the base-64 encoded value of your app's client ID and secret separated by a colon (*my_app_client_id*:*my_app_client_secret*).

   The content type must be *x-www-form-urlencoded*.

   Example request:

   ```
   curl -- location --request POST 'https://<vcenter_server_fqdn>/acs/t/<tenant>/token'
   \
   --header 'Authorization: Basic <my_app_client_id>:<my_app_client_secret>' \
   --header 'Content-Type: application/x-www-form-urlencoded' \
   --data-urlencode 'grant_type=authorization_code' \
   --data-urlencode 'code=<my_app_authorization_code>' \
   --data-urlencode 'redirect_uri=https://<my_app_redirect_uri1>'
   ```

   The authorization server returns the JWT tokens corresponding to the grant type: access token , ID token, and refresh token.

Exchange the JWT tokens for a vCenter Server SAML token that you can use to establish an authentication session. For more information, see Exchange JWT Tokens for a SAML Token and Obtain a Session Identifier.

Obtain JWT Token from VMware Identity Broker - vCenter Server by Using the Client Credentials Grant Type

Obtain JWT Token by Using the Client Credentials Grant Type
If your vCenter Server, app, or script is federated to an external identity provider, such as Okta or Azure AD, through VMware Identity Broker - vCenter Server, you can obtain an access token in JWT format by using the OAuth Client Credentials grant type.

- Verify that your vCenter Server or client app is federated to an external identity provider through VMware Identity Broker - vCenter Server.
- You must create an OAuth client for your vCenter Server or client app on VMware Identity Broker - vCenter Server. You must know the client id and secret for the OAuth client.
- You must know the tenant name for the configuration you want to use. The default tenant that is created during vCenter Server federation is called *CUSTOMER*.

You can use the Client Credentials grant type if you want to authenticate vCenter Server or another application to your external identity provider without the presence of a human user. The Client Credentials grant type is appropriate for headless automation when you want a web service to authenticate automatically against an identity provider.

1. Obtain an access token for your app.

   Send a request to the VMware Identity Broker - vCenter ServerGet Access Token endpoint with the required grant type.

In the authorization header, use the base-64 encoded value of your app's client ID and secret separated by a colon (`client_id:secret`).

The content type must be *`x-www-form-urlencoded`*.

Example request:

```
curl --location --request POST 'https://<vcenter_server_fqdn>/acs/t/<tenant>/token' \
--header 'Authorization: Basic <my_app_client_id>:<my_app_client_secret>' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=client_credentials'
```

The authorization server returns the JWT token corresponding to the grant type: access token.

Exchange the JWT token for a vCenter Server SAML token that you can use to establish an authentication session. For more information, see Exchange JWT Tokens for a SAML Token and Obtain a Session Identifier.

Exchange JWT Tokens for a SAML Token and Obtain a Session Identifier

You can exchange the JWT tokens that you obtained from the authorization server for a vCenter Server SAML token and create an authenticated session.

- You must have a valid access and ID token (in JWT format) from the authorization server (AD FS or VMware Identity Broker - vCenter Server) for the user or application you want to authenticate.

1. Prepare the SAML token for the API call.

   Base-64 decode the SAML token, compress it, and then Base-64 encode it again.

2. To obtain a session identifier, call the with the SAML token.
3.

You created an authenticated session and now you can access and use the vSphere Automation services.

Bash Shell

This Bash script illustrates the use case where you get JWT tokens from the authorization server (AD FS or VMware Identity Broker - vCenter Server), exchange them for a vCenter Server SAML token, and obtain an authentication session identifier from the vSphere Automation endpoint.

This script consists of three parts:

1. Obtain JWT tokens from the authorization server (AD FS or VMware Identity Broker - vCenter Server) by using the OAuth 2.0 Password grant type.
2. Exchange the JWT tokens for a vCenter Server SAML token.
3. Use the SAML token to obtain a session identifier for the vSphere Automation API.

```
#!/bin/bash


: '
```

```
Variable definitions:

$vcip = The IP address or FQDN of your vCenter Server.

$ACCESS_TOKEN = The access token in JWT format that you received from the
authentication server.

$ID_TOKEN = The ID token in JWT format that you received from the authentication
server.

'

if [ -z "$1" ] || [ -z "$2" ] || [ -z "$3" ] || [ -z "$4" ] || [ -z "$5" ] || [ -z
"$6" ]; then

  echo "Usage: <vc-ip> <pwgrant-userid> <pwgrant-password> <client-id> <client-
secret> <token-endpoint>"

  exit 0

fi



vcip="$1"

userid="$2"

password="$3"

clientid="$4"

clientsecret="$5"

tokenendpoint="$6"


echo "Obtaining JWT access and ID tokens for user $userid ..."



PWGRANT_OUTPUT=$(curl -k --silent --location -u "$clientid:$clientsecret" --request
POST "$tokenendpoint" \

  --header "Content-Type: application/x-www-form-urlencoded" \

  --data-urlencode "grant_type=password" \

  --data-urlencode "username=$userid" \

  --data-urlencode "password=$password")

ACCESS_TOKEN=$(echo $PWGRANT_OUTPUT | jq -r '.access_token')

ID_TOKEN=$(echo $PWGRANT_OUTPUT | jq -r '.id_token')



echo

echo "Access token: $ACCESS_TOKEN"
```

```
echo

echo "ID token: $ID_TOKEN"

echo


echo "Exchanging JWT tokens for vCenter SAML token ..."


TOKEN_EXCHANGE_OUTPUT=$(curl -k --silent --location --request POST "https://$vcip/
api/vcenter/authentication/token" \

  --header "Content-Type: application/x-www-form-urlencoded" \

  --header "Authorization: Bearer $ACCESS_TOKEN" \

  --data-urlencode "subject_token_type=urn:ietf:params:oauth:token-type:access_token"
\

  --data-urlencode "subject_token=$ACCESS_TOKEN" \

  --data-urlencode "actor_token_type=urn:ietf:params:oauth:token-type:id_token" \

  --data-urlencode "actor_token=$ID_TOKEN" \

  --data-urlencode "grant_type=urn:ietf:params:oauth:grant-type:token-exchange" \

  --data-urlencode "requested_token_type=urn:ietf:params:oauth:token-type:saml2")


echo

echo "$TOKEN_EXCHANGE_OUTPUT"

echo


SAML_TOKEN=$(echo $TOKEN_EXCHANGE_OUTPUT | jq -r '.access_token')


echo "vCenter SAML token: $SAML_TOKEN"

echo


echo "Establishing vCenter session with SAML token ${SAML_TOKEN::7}..."

echo

COMP_TOKEN=$(echo $SAML_TOKEN | base64 -d | gzip | base64 -w0)


SESSION_OUTPUT=$(curl -k --silent --location --request POST "https://$vcip/api/
session" \
```

```
  --header "Authorization: SIGN token=\"$COMP_TOKEN\"")


echo "Create Session Response: $SESSION_OUTPUT"


SESSION_ID=$(echo "$SESSION_OUTPUT" | tr -d '"')


echo "Tagging categories:"

curl -k "https://$vcip/api/cis/tagging/category" --header "vmware-api-session-id:
$SESSION_ID"


echo

echo "Done!"
```

Accessing vSphere Automation Services

vSphere Automation SDK provides mechanisms for creating remote stubs to give clients access to vSphere Automation services.

The sequence of tasks you must follow to create a remote stub starts with creating a `ProtocolFactory`. You use the protocol factory object to create a `ProtocolConnection`. Connection objects provide the basis for creating stub interfaces tovSphere Automation services.

When you establish a connection to the vSphere Automation endpoint, you can create a `StubFactory` object and a `StubConfiguration` object. With these objects, you can create the remote stub for the vSphere Automation service that you want to access.

The complete connection sequence also includes SSL truststore support and a temporary `StubConfiguration` that you use for SAML token authentication and session creation.

For greater security, use an external utility to create a certificate store:

```
keytool -import -noprompt -trustcacerts \

-alias <alias name> \

-file <certificate file> \

-keystore <truststore filename> \

-storepass <truststore password>
```

Access a vSphere Automation Service

To access a vSphere Automation service, you must have a valid session connection. The sequence for accessing a vSphere Automation service includes creating a protocol connection object and using it to create the service stub.

Establish a connection to the vSphere Automation endpoint URL. For more information about the authentication mechanisms that you can use, see Authentication Mechanisms.

1. Create a protocol factory object.
2. Create a protocol connection object to access an API provider.

   The vSphere Automation API clients use `ApiProvider` instances to invoke operations on services running in the virtual environment. To invoke an operation, you must specify the target service and operation, input parameters, and execution context.
3. Create a `StubFactory` object by using the `ApiProvider` instance.
4. Create a `StubConfiguration` instance and set the security context to be used for the service stub.
5. Create the stub for the vSphere Automation service interface by calling the create method of the `StubFactory` instance. Pass the service class and the `StubConfiguration` instance as arguments.

A connection to the vSphere Automation endpoint is established and you can access all avaiable vSphere Automation services.

Java

The example is based on the code in the `LibraryCrud.java` sample.

This example shows the steps for creating an authenticated session to the vSphere Automation endpoint and creating the service stub for the Content Library API provider.

NOTE
For a complete and up-to-date version of the Java sample code, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...


// Log in by using username/password

      this.stubFactory = createApiStubFactory(server, httpConfig);


// Create a security context for username/password authentication

SecurityContext securityContext =

        SecurityContextFactory.createUserPassSecurityContext(

            username, password.toCharArray());


// Create a stub configuration with username/password security context

StubConfiguration stubConfig = new StubConfiguration(securityContext);
```

```
// Create a session stub by using the stub configuration.
Session session =
        this.stubFactory.createStub(Session.class, stubConfig);

// Log in and create a session
char[] sessionId = session.create();

// Initialize a session security context from the generated session id
SessionSecurityContext sessionSecurityContext =
        new SessionSecurityContext(sessionId);

// Update the stub configuration to use the session id
stubConfig.setSecurityContext(sessionSecurityContext);

/*
 * Create a stub for the session service using the authenticated
 * session
 */
this.sessionSvc =
        this.stubFactory.createStub(Session.class, stubConfig);

// Create service stubs for the Content Library service.
Library libraryService = stubFactory.createStub(Library.class, stubConfig);

// Invoke an operation of the Content Library service.
List<String> listContentLibraries = libraryService.list();
```

Managing the Life Cycle of Hosts and Clusters

You can manage the life cycle of ESXi hosts collectively by using the vSphere Lifecycle Manager feature through the vSphere Automation API. Starting with vSphere 8.0, you can manage the lifecycle of a standalone ESXi host by using an image only through the vSphere Lifecycle Manager automation API.

You can automate the life cycle management of a cluster or standalone host by performing the following operations:

- Retrieve information about the current state of the cluster or the standalone host.

- Create a desired state that includes a specific version of the ESXi host. You can also add some compatible partner software and firmware components and add-ons.
- Validate the desired state to detect any discrepancies between the desired state and the host hardware.
- Check the compliance of a cluster or host against the desired state and determine whether some additional steps must be taken to ensure the success of the cluster or host remediation.
- Apply the desired state on a cluster or a standalone host.

You can use the vSphere Lifecycle Manager to manage the life cycle of hosts in a cluster that meet the following requirements:

- Hosts must be of version 7.0 and later.
- Hosts must be stateful.
- All hosts in the cluster must be from the same vendor and with identical hardware.
- The hosts must include only integrated solutions, such as VMware vSAN™, vSphere Supervisor, NSX and VMware vSphere® High Availability.

A standalone host is a host that is added to a vCenter Server instance but is not part of any cluster. For more information about how to add, connect, and disconnect standalone host, see Hosts. You can manage the life cycle of a standalone host by performing almost all vSphere Lifecycle Manager operations that you can perform on a cluster level. The only limitation for managing the life cycle of a standalone host through the vSphere Automation API, is that you can't update the firmware of the host.

vSphere Lifecycle Manager Terms

You must understand the basic terminology that is used within this chapter to be able to use the vSphere Lifecycle Manager functionality efficiently.

vSphere Lifecycle Manager Terminology

Term	Definition
Upgrade, update, and patch	You can upgrade to another major version of the software running on an ESXi host, and install patches and updates that include smaller changes, bug fixes, or other small improvements.
Depot	A depot is a well-defined folder structure that is used for distributing payloads and their metadata. Depots are consumed by different products and features such as the vSphere Lifecycle Manager and ESXCLI. The vSphere Lifecycle Manager works with three types of depots: online, offline, and UMDS. See Software Depots.
Component	A component is the smallest unit that the vSphere Lifecycle Manager uses during the installation and update processes. Software vendors use components to encapsulate a group of payloads that can be managed together.
Base image	A base image is a collection of components that shape the bootable ESXi used for the installation or upgrade process. Base images are currently distributed only by VMware and support x86 servers. VMware provides new versions of the base image for each upgrade, update, and patch release of the ESXi. Base images are hosted at the VMware online depot that is available by default to the vSphere Lifecycle Manager.

Table continued on next page

Continued from previous page

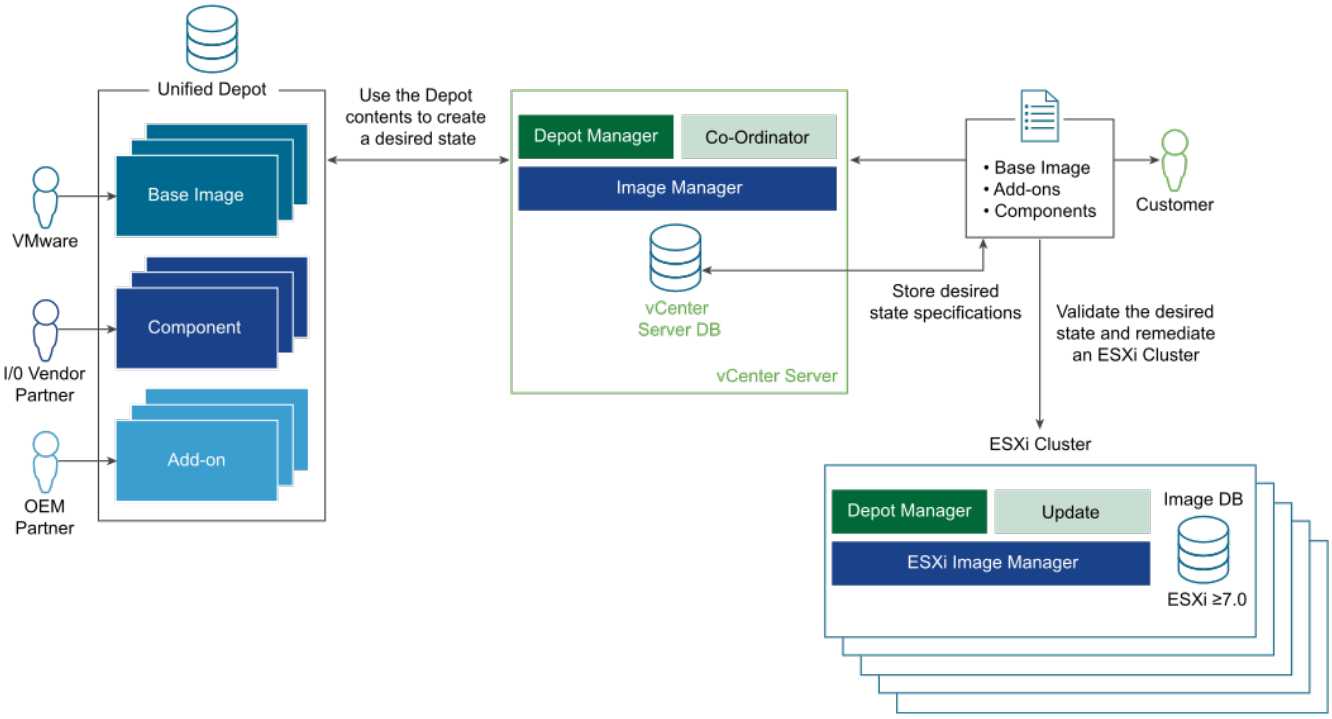
Term	Definition
	Furthermore, you can download a different base image version, in the form of an offline ZIP bundle, from https://my.vmware.com/web/vmware/downloads.
Vendor add-on	An add-on is a collection of components that different OEMs provide on top of a base image. Vendors use an add-on to group some customizations for a family of servers. Unlike base images, add-ons are not complete and are not sufficient to boot an ESXi. Each add-on must have a unique name and version. An add-on allows vendors to add, remove, or update components that are part of the VMware base image, if there are no unresolved dependencies and conflicts between the components.
Solution	A solution contains one or more components, and provides information about its constraints and compatibility with the different ESXi versions. For example, from the perspective of the vSphere Lifecycle Manager solutions are VMware NSX, VMware vSphere® High Availability, vSAN.
Desired state	A desired state of a cluster is represented with a software specification. The desired state defines a set of components that a user wants to install on a single ESXi host or on a cluster of hosts.
OEMs	Original Equipment Manufacturers. VMware partners enrolled in the VMware Partner Connect application, such as Dell, Inc., HP Inc., Lenovo Group Ltd., and so on.
IOVP	I/O Vendor Partner. Qualified VMware partners providing certified I/O device drivers for network and storage host bus adapters.
Third-party software providers	Providers of I/O filters, device drivers, CIM modules, and so on, that are not part of VMware partner programs.
vSphere Configuration Profiles	With vSphere Configuration Profiles, you can manage the configuration of all hosts in a cluster collectively. This ensures consistency in host configuration at a cluster level. You can enable vSphere Configuration Profiles only on clusters that have the vSphere Lifecycle Manager enabled.

vSphere Lifecycle Manager Overview

The vSphere Lifecycle Manager feature provides means for managing the life cycle of hosts on a cluster level. The functionality of this feature is achieved through several major services that are running on both vCenter Server and ESXi.

After you install vSphere 8.0 Update 1, you can access the feature through several major services available on vCenter Server and on each ESXi host.

Figure 106: vSphere Lifecycle Manager System Architecture



Depot Manager

For each upgrade and patch release, VMware, OEMs, and other third-party companies make the software updates of their products available to the customers. Software updates are distributed to different locations and in different formats depending on the way they are accessed, downloaded, and used. Depot Manager allows these different software resources to be presented to the vSphere customers in a unified format and as a result, makes them easier to use.

Depending on your environment and specific use case, Depot Manager gives you access to the software updates within three different types of depots: online, offline, and UMDS. See Software Depots.

Depot Manager that runs on the vCenter Server instance achieves the following goals:

- Represents the contents of all depots in a unified way.
- Caches the payloads and their metadata locally prior to their use.
- Enables depot overrides on a cluster or host level for Remote Office/Branch Office (ROBO) environments, or Edge computing environments.

Depot Manager that runs on the hosts serves as a proxy to the vCenter Server Depot Manager. If ROBO cluster or ROBO host, or Edge computing environments are enabled, Depot Manager that runs on the ESXi hosts serves as a proxy to the nearby vSphere Lifecycle Manager compatible depot.

Image Manager

Image Manager allows you to create a desired state that you can apply on a cluster or a standalone host. The specification describes all components, add-ons, and the base image that you can use to update or upgrade the hosts in your environment. Image Manager supports validation of the desired state and the detecting drifts from the desired state.

Coordinator and ESXi Updater Managers

Coordinator Manager runs on the vCenter Server instance and makes sure that the desired state is applied to all hosts in the cluster. This module also runs pre-checks to evaluate how each host in the cluster is affected by the remediation and

whether you must take some additional steps to ensure the success of the procedure. Coordinator Manager also allows you to query the status of the remediation operation for the cluster and for each host part of the cluster.

The ESXi Updater Manager takes care of the actual remediation happening on each host.

Options for Managing the ESXi Life Cycle

Based on your needs and environment setup, you can choose from several methods for managing the life cycle of the ESXi hosts. The vSphere Lifecycle Manager provides means for updating all hosts in a cluster or a standalone host with a desired software state.

To manage the life cycle of the hosts in your environment, you can use the vSphere Lifecycle Manager through the vSphere Client. See Managing Host and Cluster Lifecycle.

This chapter of the vSphere Automation Programming Guide discusses how you can access and use the functionality provided by the vSphere Lifecycle Manager Automation API.

vSphere Lifecycle Manager Features

You can use the vSphere Automation API to manage the life cycle of all hosts in a cluster or of standalone hosts by using a vSphere Lifecycle Manager image. You can access and use the following vSphere Lifecycle Manager functionality:

- Depot management. You can add, remove, explore the contents of different types of depots. See Software Depots. The content of the depots is provided by VMware and VMware partners. Partners can use the ESXi Packaging Kit (EPK) to assemble a custom bootable ESXi image. The custom image can then be shared to other third-party customers and used through the Depot Manager service. For more information about how to create custom ESXi images, see *`ESXi Packaging Kit (EPK) Development Guide`*.
- Desired software state. You can create, edit, and delete a desired software state for a cluster or a standalone host on which the vSphere Lifecycle Manager is enabled. A desired software state must contain at least a single ESXi image provided by VMware. You can also set an add-on provided by OEMs, and one or more components by different software vendors. Furthermore, during the process of creating the desired software state, you can check the validity of the specification and compare the current state of the hosts in the cluster or the standalone host with the desired software state.
- Cluster remediation. You can apply the desired state on each of the hosts in a cluster which current state is different from the desired specification. Applying a desired state on a cluster level has the following prerequisites:
  - The cluster must have the vSphere Lifecycle Manager enabled.
  - All hosts in the cluster must store their data on a local or remote disk, or on a USB drive.
  - All hosts in the cluster must be of version 7.0 or higher.
  - All hosts must contain only components that the vSphere Lifecycle Manager can recognize and maintain. If a host contains some old content that the vSphere Lifecycle Manager does not recognize, the content is removed from the host during remediation.
- Standalone host remediation. You can apply the desired software state on a standalone host that is managed with baselines or on a standalone host that is managed with images but its current state differs from the desired state.

Enabling a Cluster to Use a Software Specification

If you want to use the vSphere Lifecycle Manager to manage the life cycle of clusters in your environment, you have two options for enabling this feature. You can enable the vSphere Lifecycle Manager when you create the cluster. You can also turn an already created cluster into one managed by the vSphere Lifecycle Manager.

Creating a Cluster with Enabled vSphere Lifecycle Manager

You can use only the vSphere Web Services API or the Virtual Infrastructure Management API to create a cluster in your virtual environment and specify the initial desired state of the cluster. Use the vSphere AutomationAPI to specify the detailed desired state after the cluster creation.

Starting with vSphere 8.0 Update 1, you can use the Virtual Infrastructure Management API which introduced a new way of accessing and using the vSphere Web Services. That is, you can now use the JSON data format over HTTP and create REST-like requests to the vSphere Web Services. Use the Virtual Infrastructure API or the vSphere Web Services API to create a cluster. For more information about how to authenticate and use the Virtual Infrastructure Management API, see the *vSphere Web Services SDK Programming Guide*, *An Introduction to vSphere APIs*, and *Getting Started with vSphere APIs* documentation. You can also check the *Virtual Infrastructure JSON API* reference documentation at https://developer.vmware.com/apis/vi-json/latest/.

You can use the `POST https://<vcenter_ip_address_or_fqdn> /sdk/vim25/<release_number>/Folder/<moID>/CreateClusterEx` request and submit the name of the cluster and a `ClusterConfigSpecEx` data object in the request body. In the data object, among other properties, you can specify the desired state for the cluster. The `desiredSoftwareSpec` property in the `ComputeResourceConfigSpec` data object contains the desired software specification for the cluster. This property is available for applications using the vSphere Web Services API of version 7.0 and later. You can create a `DesiredSoftwareSpec` data object and specify the base image that must be applied on the cluster with the `baseImageSpec` property. Optionally, you can specify a vendor add-on to be added to the software specification with the `vendorAddOnSpec` property.

Starting with vSphere 8.0, you can also enable the cluster to use vSphere Configuration Profiles by setting the `enableConfigManager` property of the `ComputeResourceConfigSpec` data object to `true`.

Enabling an Existing Cluster to Use vSphere Lifecycle Manager

If you want to manage the life cycle of a cluster by using a single software specification, you must first enable vSphere Lifecycle Manager on that cluster. You can use the vSphere AutomationAPI to enable a cluster to use the vSphere Lifecycle Manager feature.

Before you enable vSphere Lifecycle Manager on a cluster, you can check whether the cluster meets all prerequisites. vSphere Lifecycle Manager can be enabled for a cluster only if the following requirements are met:

- All hosts in the cluster are of version 7.0 or later.
- All hosts in the cluster are stateful.
- All hosts in the cluster include only components that belong to integrated solutions, such as VMware vSAN™ and VMware vSphere® High Availability.
- None of the hosts in the cluster are in the process of active remediation through the VMware vSphere®Update Manager™.
- The cluster has a desired state already created for it.

If you want to run a preliminary check about whether all hosts in the cluster meet these requirements, .

The cluster ID represents the unique identifier for a cluster resource.

to specify which checks can be skipped during the cluster preliminary check. Though you can skip some checks with this operation, the Image Manager runs all checks during the enablement operation. If you leave the check specification empty, all checks are run for each host in the cluster. You can select among the following checks to be skipped when running a pre-check operation:

- `SOFTWARE`. Checks whether there are any orphaned vSphere Installation Bundles (VIBs) and any software that cannot co-exist with vSphere Lifecycle Manager.
- `VERSION`. Checks whether all hosts in the cluster are of version greater than a predefined one.
- `STATELESSNESS`. Checks whether there are any stateless hosts in the cluster. vSphere Lifecycle Manager can be enabled only if the cluster does not contain stateless hosts.
- `VUM_REMEDIATION`. Checks whether any of the hosts in the cluster are currently remediated through the VMware vSphere®Update Manager™.
- `SOFTWARE_SPECIFICATION_EXISTENCE`. Checks whether there is a software specification already associated with this cluster. In case, this check reports that the cluster does not have a software specification, you must first create a draft software specification for this cluster and then commit the draft.

- `VSAN_WITNESS_ELIGIBILITY`. Checks whether the software specification can be used on any vSAN witness hosts in the cluster. For information about how you can manage a vSAN cluster by using vSphere Lifecycle Manager, see *`vSAN Clusters and vSphere Lifecycle Manager`* chapter in the *`Managing Host and Cluster Lifecycle`* documentation.

You can also get information about which clusters in your environment are managed with a single software specification.

Enabling a Standalone Host to Use a Software Specification

Starting with vSphere 8.0, you can use the vSphere Lifecycle Manager to manage the life cycle of standalone hosts in your vCenter Server system.

You can add a standalone host to a vCenter Server instance under a data center object or into a folder. See Hosts. If you remove a host from a cluster but leave it in the data center, the host is also considered a standalone host.

To manage a standalone host with a software specification, you must first enable the vSphere Lifecycle Manager on that host. You can run a preliminary check to establish whether the standalone host meets all requirements for enabling the vSphere Lifecycle Manager. . The pre-checks that you can skip before enabling the vSphere Lifecycle Manager on a standalone host are the same as for a cluster.

To enable vSphere Lifecycle Manager on a standalone host, . Currently, you can skip only the check for orphaned vSphere Installation Bundles (VIBs).

Hardware Compatibility Data

The hardware compatibility data contains information about the compatibility between ESXi hosts and ESXi versions.

You can use the service to retrieve information about the compatibility data or to update the local compatibility data on a vCenter Server instance. To retrieve information about the compatibility data stored on your vCenter Server instance, . To update the local compatibility data with the latest version available from the official VMware source, .

Checking the Hardware Compatibility of an ESXi Host

You can query the hardware compatibility for a host before upgrading to a new ESXi version. You can also download the information generated by the hardware compatibility report.

To use services form the package, you must verify that you have accepted to participate in the CEIP and there is available compatibility data.

You can use the to list available releases for generating a compatibility report for a specific ESXi host. To list the locally available ESXi releases for the host that can be used to generate a compatibility report,. The list includes only major and update releases. Patch releases are not listed.

You can use the to generate a hardware compatibility report for an ESXi host against a specific ESXi release. To return the last generated hardware compatibility report for a specific host, . To generate a hardware compatibility report for a specific host against specific ESXi release, .

You can use the to download information generated by the hardware compatibility report. To retrieve the URI location for downloading a compatibility report, .

Configuring Remediation Settings

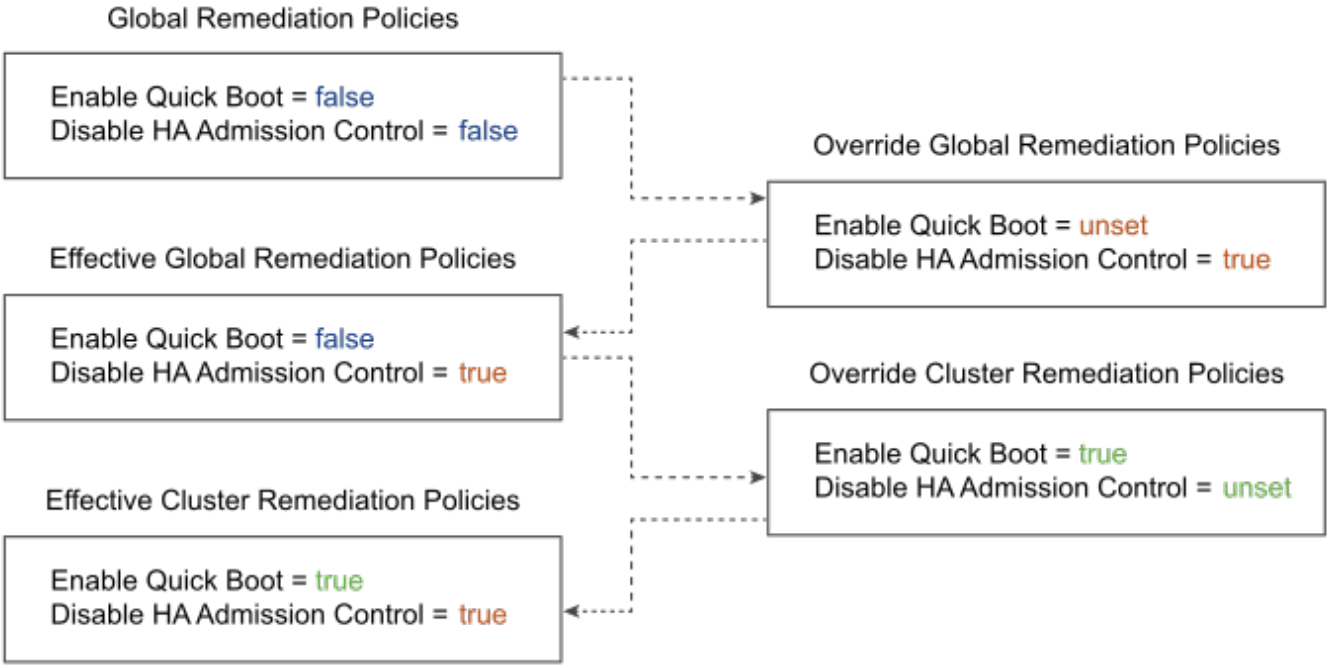
You can control the behavior of the ESXi hosts and virtual machines during the remediation process. You can create a global remediation policy that applies to all clusters and standalone hosts in a vCenter Server instance. You can also set a remediation policy to a specific cluster and a standalone host.

When you run cluster and standalone host compliance checks, the Coordinator module runs a series of checks on each host to determine their state and whether some additional actions must be taken to ensure the success of the remediation operation. In case one or more hosts in the cluster and any of the standalone hosts are evaluated as non-compliant, additional checks are run on those hosts to evaluate whether they must be rebooted or put into maintenance mode. Currently, VMware provides a set of behavior controls (remediation policies) regarding the virtual machines and the hosts in a cluster or a standalone host. This set of remediation policies might change with the next vSphere release.

How Remediation Policies Overrides Work

The vSphere Lifecycle Manager provides a default global policy configuration that must be applied on each cluster and standalone host during remediation. Through the vSphere AutomationAPI, you can change the global policies and create some cluster-and host-specific policies. Before remediating a cluster and standalone host, you can use the API to determine the effective global and cluster-and host-specific remediation policies. The following graphic describes how the mechanism of the policy overrides works.

Figure 107: How Remediation Policies Work



All clusters and standalone hosts in a vCenter Server instance inherit the default or the overridden global policy settings unless the global policy is explicitly overridden on a cluster and host level.

Editing Global or Cluster- and Host-Specific Remediation Policies

Remediation Policy Options for Clusters

To describe a global or cluster-specific remediation policy, use . For the vSphere 8.0 release, VMware provides the following methods to configure a global or cluster-specific policy.

	Description
	Deactivate the VMware Distributed Power Management (DPM) feature for all clusters or for a specific cluster. DPM monitors the resource consumption of the virtual machines in a cluster. If the total available resource

Table continued on next page

Continued from previous page

	Description
	capacity of the hosts in a cluster is exceeded, DPM powers off (or recommends powering off) one or more hosts after migrating their virtual machines. When resources are considered underutilized and capacity is needed, DPM powers on (or recommends powering on) hosts. Virtual machines are migrated back to these hosts. During the cluster remediation, the vSphere Lifecycle Manager cannot wake up and remediate hosts that are automatically put into a stand-by mode by DPM. These hosts stay non-compliant when DPM turns them on. The vSphere Distributed Resource Scheduler (DRS) is unable to migrate virtual machines to the hosts which are not remediated with the desired state for the cluster. To deactivate DPM during the cluster remediation, . By default, the vSphere Lifecycle Manager temporarily deactivates DPM and turns on the hosts to complete the remediation. DPM is enabled again when the cluster remediation finishes.
	Deactivate the vSphere HA admission control. vSphere HA uses admission control to ensure that a cluster has sufficient resources to guarantee the virtual machines recovery when a host fails. If vSphere HA admission control is enabled during remediation, putting a cluster into maintenance mode fails because vMotion cannot migrate virtual machines within the cluster for capacity reasons. To allow the vSphere Lifecycle Manager to temporary deactivate vSphere HA admission control, . By default, the vSphere HA admission control is enabled because DRS should be able to detect issues with the admission control and deactivate it to allow the remediation to complete.
	Migrate the suspended and powered off virtual machines from the hosts that must enter maintenance mode to other hosts in the cluster. To enable this remediation policy, . By default, this setting is deactivated in the global remediation policy.
	Specify what actions vSphere Lifecycle Manager must take if a host fails to enter maintenance mode during the remediation. To configure this policy on a global or cluster-specific level, . You can set the number of times that vSphere Lifecycle Manager tries to put a host into maintenance mode and the delay between the tries. When the threshold is reached and the host failed to enter maintenance mode, the cluster remediation fails. By default, vSphere Lifecycle Manager tries to put a host into maintenance mode three times with a five minute delay between each try before the cluster remediation fails.
	Prevents the remediation of vSAN clusters if vSphere Lifecycle Manager reports hardware compatibility issues

Table continued on next page

Continued from previous page

	Description
	during the hardware compatibility check performed as part of the remediation pre-check or the remediation tasks of the cluster., detected hardware issues are reported as warnings and do not prevent the remediation of the vSAN cluster.
	Enable simultaneous remediation of all hosts that are in maintenance mode with in the cluster. to indicate the maximum number of hosts that can be remediated in parallel. **NOTE** If the hosts have NSX virtual distributed switches that are ready to be migrated to vSphere Distributed Switches, you must manually set the maximum number of parallel remediations to no more than 4. In cases when host switch migration is needed, if more than 4 hosts are remediated in parallel, the remediation might fail, because the host switch migration takes more time than the time vSphere Lifecycle Manager needs to complete the parallel remediation.
	Specify how the power state of the virtual machines must change before the host enters maintenance mode. If DRS is not enabled on a cluster or the automation level of a DRS cluster is not set to fully automated, the Coordinator module fails to remediate the cluster if the remediation requires a reboot or maintenance mode. You can set a policy that powers off or suspends the virtual machines on hosts that must be rebooted or must enter maintenance mode during remediation. The DRS takes care of changing the power state of the virtual machines when the host enters and exits maintenance mode. To set a policy for the power state of the virtual machines during the remediation, one of the following values: • `DO_NOT_CHANGE_VMS_POWER_STATE`. Indicates that the power state of the virtual machines must not be changes. • `POWER_OFF_VMS`. Indicates that the virtual machines must be powered off before the hosts enter maintenance mode. • `SUSPEND_VMS`. Indicates that the virtual machines must be suspended before the hosts enter maintenance mode. • `SUSPEND_VMS_TO_MEMORY`. Indicates that the virtual machines must be suspended before the hosts enter maintenance mode. By default, the Coordinator must leave the power state of the virtual machines unchanged.

Table continued on next page

Continued from previous page

	Description
	Reduce the reboot time of an ESXi host by skipping all the hardware initialization processes and restarting only the hypervisor. This policy is applicable only if the host platform supports the Quick Boot feature. To enable the Quick Boot feature on the hosts during the remediation, . By default, this policy is deactivated.

Remediation Policy Options for Standalone Hosts

To describe a global or cluster-specific remediation policy, use the . For the vSphere 8.0 release, VMware provides the following methods to configure a global or host-specific policy.

	Description
	Optimize the host patching and upgrade operations by reducing the reboot time of an ESXi host. Since the patching and upgrading operations do not affect the hardware of the host, the hardware initialization processes can be skipped. This policy is applicable only if the host platform supports the Quick Boot feature. For more information about which hosts are Quick Boot compatible, see the following KB article https://kb.vmware.com/s/article/52477.
	Specify how the power state of the virtual machines must change before the standalone host enters maintenance mode. You can choose between the following power actions: • `DO_NOT_CHANGE_VMS_POWER_STATE`. Indicates that the power state of the virtual machines must not be changed. • `POWER_OFF_VMS`. Indicates that the virtual machines must be powered off before the standalone host enters maintenance mode. • `SUSPEND_VMS`. Indicates that the virtual machines must be suspended before the standalone host enters maintenance mode. • `SUSPEND_VMS_TO_MEMORY`. Indicates that the virtual machines must be suspended before the standalone host enters maintenance mode.
	Specify what actions vSphere Lifecycle Manager must take if a standalone host fails to enter maintenance mode during the remediation. You can set the number of times that vSphere Lifecycle Manager tries to put a standalone host into maintenance mode and the delay between the tries. When the threshold is reached and the standalone host failed to enter maintenance mode, the host remediation fails.

Remediating an ESXi Cluster and a Standalone Host

You can use the vSphere Automation API to initiate the remediation of a cluster or a standalone host through the vSphere Lifecycle Manager.

To remediate a cluster with a desired state, . You can also set the minimum commit ID of the draft software specification that must be used for remediating the cluster. Upon successful completion of the remediation task, all hosts in the cluster have the same software state.

To remediate a standalone host with a desired state, . Use the software apply specification to set the minimum commit identifier of the draft software specification. See Committing the Draft Software Specification. You can also specify whether the VMware End User License Agreement (EULA) must be accepted.

Integrate Third-Party Solutions with vSphere Lifecycle Manager

A solution is an ESXi software package that extends the functionality and capabilities of a host and integrates with the vCenter Server system. To be able to manage the life cycle of your third-party solutions on a cluster managed with images, you must integrate the solutions with vSphere Lifecycle Manager.

Examples of VMware integrated solutions are VMware NSX®, vSAN, and vSphere Supervisor. For more information about the VMware integrated solutions, see *vSphere Lifecycle Manager Images and Other VMware Products and Solutions* section in the *Managing Host and Cluster Lifecycle* documentation.

Third-party software providers can use the vSphere APIs, VMware vSphere APIs for I/O Filtering (VAIO), VMware Daemon Software Development Kit, and others to develop third-party solutions for their vSphere platforms.

You can use the vSphere Lifecycle Manager automation APIs to manage the life cycle of third-party solutions on a cluster managed with a single image. First you need to package and upload your solution components to the vSphere Lifecycle Manager depot. Use the depot to store and manage the software updates for your third-party solutions. To make a solution available on a cluster, create a software specification that contains the solution and remediate all hosts in the cluster with that image. vSphere Lifecycle Manager manages the life cycle of the solution components by consuming the software updates from the vSphere Lifecycle Manager depot.

> **NOTE**
> If you export an image from the cluster where your third-party solution is running, the solution components are not part of the exported image.

Prerequisites

To enable vSphere Lifecycle Manager to manage your third-party solutions, you must use the ESXi Packaging Kit (EPK) to create installable packages. As of the vSphere 7.0 release, partner development kits generate components as installation packages. For more information about how to use the EPK to create components, assemble components into add-ons, then merge the add-ons with the base image to author a depot, see the *ESXi Packaging Kit (EPK) Development Guide* documentation.

Integrate a Third-Party Solution to Work with vSphere Lifecycle Manager

1. Create an online or offline depot to host your third-party solutions. See Working with Online Depots and Working with Offline Depots.
   – The following example creates an online depot that can be accessed through the `http://my_online_depot.com` URL.
   `POST https://<vcenter_server_ip_or_fqdn>/api/esx/settings/depots/online`

```
{
   "description" : "My online depot adds the My Solution component to the ESXi
7.0U3d base image",
```

```
    "ownerdata" : "ACME Company",

    "location" : "http://my_online_depot.com",

    "enabled" : true

}
```

– The following example creates an offline depot, also called an offline bundle, that can be imported to the vSphere Lifecycle Manager depot.

```
POST https://{server}/api/esx/settings/depots/offline?vmw-task=true


{

    "file_id" : "string",

    "description" : "string",

    "ownerdata" : "string",

    "source_type" : "PULL",

    "location" : "http://myurl.com"

}
```

2. Synchronize the online depot to download the depot metadata and make the vSphere Lifecycle Manager aware of your solutions. See Synchronizing Software Depots.

```
POST https://<vcenter_server_ip_or_fqdn>/api/esx/settings/depots/online?vmw-
task=true&action=sync
```

3. Create a draft software specification. See Creating a Draft Software Specification.

```
POST https://<vcenter_server_ip_or_fqdn>/api/esx/settings/clusters/<cluster_id>/
software/drafts
```

4. Add your solution to the created draft software specification. See Editing a Draft Software Specification.

```
PUT https://<vcenter_server_ip_or_fqdn>/api/esx/settings/clusters/<cluster_id>/
software/drafts/<draft_id>/software/components/<component_id>
```

5. Save the created draft software specification to make it the desired state for the cluster. See Committing the Draft Software Specification.

```
POST https://<vcenter_server_ip_or_fqdn>/api/esx/settings/clusters/<cluster_id>/
software/drafts/<draft_id>?action=commit&vmw-task=true
```

6. Remediate the ESXi cluster with the desired state that contains your solution to apply the desired state on all hosts in that cluster. See Remediating an Cluster and a Standalone Host.

```
POST https://<vcenter_server_ip_or_fqdn>/api/esx/settings/clusters/<cluster_id>/
software?action=apply&vmw-task=true

{

    "hosts" : [

        "obj-103",

        "obj-103"

    ],

    "commit" : "obj-103",
```

```
    "accept_eula" : true
}
```

Results

You set up an image for the cluster which includes your third-party solution. You now manage all hosts in the cluster collectively with a single image. Upon remediation, the image is installed on all hosts in the cluster.

What to do next

You can update, delete or add new solutions to the draft software specification and then commit the changes to make the draft the desired state for the cluster.

Enable vSphere Lifecycle Manager on a Cluster Managed with Baselines

For more information about how to convert a cluster to use vSphere Lifecycle Manager images instead of baselines, see Enabling an Existing Cluster to Use .

vSphere Lifecycle Manager APIs Equivalent to the vSphere Host Patch Manager APIs

To manage the life cycle of a single host with baselines, you use the vSphere Host Patch Manager APIs which are part of the vSphere Web Services API. To manage the life cycle of all hosts in a cluster collectively with an image, you use the vSphere Lifecycle Manager APIs which are part of the vSphere Automation APIs. The following table compares the APIs for life cycle management of the hosts and clusters in your environment.

Operation	vSphere Lifecycle ManagerAPI	vSphere Host Patch Manager API
Check whether hosts can be remediated	`com.vmware.esx.settings.clusters.Software.check`	`vim.host.PatchManager.CheckHostPatch_Task`
Check the compliance of the cluster or host with the desired state	`com.vmware.esx.settings.clusters.Software.scan`	`vim.host.PatchManager.ScanHostPatchV2_Task`
Remediate the cluster or host with the desired state	`com.vmware.esx.settings.clusters.Software.apply`	`vim.host.PatchManager.InstallHostPatchV2_Task`
Retrieve information about the cluster or host that you want to remediate	• `com.vmware.esx.settings.clusters.Software.get` - View information about the current desired state. • `com.vmware.esx.settings.clusters.software.Components.list` - View all components in the current desired state. • `com.vmware.esx.settings.clusters.software.Components.get` - View detailed information about a specific component.	`vim.host.PatchManager.QueryHostPatch_Task` - View information about the bulletins installed on an ESXi host.
Stage a desired state on a cluster or host	`com.vmware.esx.settings.clusters.Software.stage`	`vim.host.PatchManager.StageHostPatch_Task`
Uninstall a component	`com.vmware.esx.settings.clusters.Software.apply` - When a software component is removed from the desired software specification and then the cluster is remediated, the	`vim.host.PatchManager.UninstallHostPatch_Task`

Table continued on next page

Continued from previous page

Operation	vSphere Lifecycle ManagerAPI	vSphere Host Patch Manager API
	component will be uninstalled from all hosts in that cluster.	

Virtual Machine Configuration and Management

A virtual machine is a software computer that, like a physical computer, runs an operating system and applications.

The virtual machine consists of a set of specification and configuration files and is backed by the physical resources of a host. Each virtual machine encapsulates a complete computing environment and runs independently of the underlying hardware.

Starting with vSphere 6.5, you can create virtual machines, configure virtual machine settings, and perform power operations on the virtual machines through the vSphere Automation APIs.

Starting with vSphere 7.0, you can use the vSphere Automation APIs to perform various virtual machine management operations. For example, you can deploy virtual machines by using several approaches, clone an existing virtual machine, create an instant clone of a running virtual machine. You can also install VMware Tools which enables you to manage the life cycle and customize the networking and identity settings of the guest operating system installed on the virtual machine.

Creating Virtual Machines

You can use the vSphere Automation APIs to create virtual machines depending on your needs and infrastructure setup.

You can create a basic virtual machine and then configure it according to your needs. You can also create a more comprehensive virtual machine and then edit its settings. To create a virtual machine, you must specify the datastore, resource pool, folder, or host where the virtual machine is placed. Later, you can customize the virtual machine by specifying the boot options, number of CPUs, the guest OS, and virtual NIC. See Creating a Virtual Machine Without a Clone or Template and Configuring Virtual Machines.

You can use a turned off virtual machine to create a VM template from which to deploy other virtual machines. See Create a VM Template in a Content Library from a Virtual Machine. You can also mark a virtual machine as template by calling the `VirtualMachine.MarkAsTemplate` method from the vSphere Web Services APIs. See *vSphere Web Services SDK Programming Guide*.

You can capture a virtual machine or a vApp in an OVF template and store the template in a content library. Then you can use the OVF template to deploy a virtual machine or a vApp in your environment. See Creating Virtual Machines and vApps from Templates in a Content Library.

Creating a Virtual Machine Without a Clone or Template

You can create a virtual machine by using the `VM.create` method. The method takes as parameter a `CreateSpec` instance that describes the details of the virtual machine.

When you create a virtual machine without a template or clone, you can configure the virtual hardware, including processors, hard disc, memory. To create a virtual machine, you must specify the virtual machine attributes by using the `CreateSpec` class. For example, you can specify a name, boot options, networking, and memory for the new virtual machine.

All attributes are optional except the virtual machine placement information that you must provide by using the `PlacementSpec` class. Use the virtual machine placement specification to set the datastore, cluster, folder, host, or resource pool of the created virtual machine. You must make sure that all these vSphere objects are located in the same data center in a vCenter Server instance.

For more information, refer to the *API Reference* documentation inside the SDK.

Configuring Virtual Machines

You configure a virtual machine in the process of creation by using `com.vmware.vcenter.CteareSpec`. You can later view and edit virtual machine settings by adding or changing the type of the storage controllers, configure the virtual disks, boot options, CPU and memory information, or networks. Virtual machine settings can be configured when cloning, registering and relocating an existing virtual machine.

Name and Location

You specify the display name and the location of the virtual machine by using the `CreateSpec` and `PlacementSpec` classes.

When you create your virtual machine, use the `setName` method of the `CreateSpec` class to pass as argument the display name of the virtual machine.

You must create also a `PlacementSpec` instance that describes the location of the virtual machine in regards to the resources of a given vCenter Server instance. Use the `setPlacement(PlacementSpec placement)` method of the `CreateSpec` class to set the placement information for the virtual machine. You can set one or all of the following vSphere resources: datastore, cluster, folder, host, and resource pool.

Hardware Version

The hardware version of a virtual machine reflects the virtual hardware features supported by a virtual machine. These features depend on the physical hardware available on the ESXi host on which the virtual machine is running.

Virtual hardware features include the BIOS and Extensible Firmware Interface (EFI), the maximum number of CPUs, the maximum memory configuration, and other hardware characteristics.

When you create a virtual machine, the default hardware version of the virtual machine is the most recent version available on the host where the virtual machine is created. For information about the latest VMware products and virtual hardware versions, see Virtual machine hardware versions (1003746).

To set a different than the default hardware version, call the `setHardwareVersion(hardwareVersion)` function of the `com.vmware.vcenter.VMTypes.CreateSpec` class. Use the `HardwareTypes.Version` class to define a valid hardware version for a virtual machine. For information about the hardware features available for the virtual hardware versions, see Hardware features available with virtual machine compatibility settings (2051652).

You can set a lower virtual hardware version of a virtual machine than the highest supported by the ESXi host on which the virtual machine is running. Setting a lower hardware version can provide flexibility and is useful in the following cases:

- To help you standardize testing and deployment in your environment.
- In case you do not need the hardware features of the latest hardware version of the host.
- To maintain compatibility with hosts with a lower hardware version.

Boot Options

You can configure the boot options of a virtual machine by using the `setBoot(CreateSpec boot)` method of the `CreateSpec` class.

The method takes as argument the `BootTypes.CreateSpec` class. You can select one of the following settings when booting the virtual machine:

- Delay - Indicates a delay in milliseconds before starting the firmware boot process when the virtual machine is powered on.

- Retry - Indicates whether the virtual machine automatically retries to boot after a failure.
- Retry delay - Indicates a delay in milliseconds before retrying the boot process after a failure.
- Enter setup mode - If set to `true`, indicates that the firmware boot process automatically enters BIOS setup mode the next time the virtual machine boots. The virtual machine resets this flag to `false` once it enters setup mode.
- EFI legacy boot - If set to `true`, indicates that the EFI legacy boot mode is used.

Guest Operating System

The guest operating system that you specify affects the supported devices and available number of virtual CPUs.

When you create a virtual machine, you specify the guest operating system by using the `setGuestOS(GuestOS guestOS)` method of the `VMTypes.CreateSpec` class. The `GuestOS` class defines the valid guest OS types that you can choose from for configuring a virtual machine.

After the create operation finishes successfully, you can install the guest operating system on the mew virtual machine in the same way as you install it on a physical machine. For further information on installing a guest operating system, refer to the *`Guest Operating System Installation Guide`* at http://partnerweb.vmware.com/GOSIG/home.html and the *`vSphere Virtual Machine Administration`* guide.

Starting with vSphere 7.0, you can use the vSphere Automation APIs to install the VMware Tool on the quest operating system and perform some guest OS customizations. See Installing VMware Tools.

CPU and Memory

The `CreateSpec` class allows you to specify the CPU and memory configuration of a virtual machine.

To change the CPU and memory configuration settings, use the `CpuTypes.UpdateSpec` and `MemoryTypes.UpdateSpec` classes.

CPU Configuration

You can set the number of CPU cores in the virtual machine by using the `setCount` method of the `CpuTypes.UpdateSpec` class. The supported range of CPU cores depends on the guest operating system and virtual hardware version of the virtual machine. If you set `CpuTypes.Info.getHotAddEnabled()` and `CpuTypes.Info.getHotRemoveEnabled()` to `true`, you allow virtual processors to be added or removed from the virtual machine at runtime.

Memory Configuration

You can set the memory size of a virtual machine by using the `setSizeMiB` method of the `MemoryTypes.UpdateSpec` class. The supported range of memory sizes depends on the configured guest operating system and virtual hardware version of the virtual machine. If you set `MemoryTypes.UpdateSpec.setHotAddEnabled()` to `true` while the virtual machine is not powered on, you enable adding memory while the virtual machine is running.

Networks

You configure network settings so that a virtual machine can communicate with the host and with other virtual machines. When you configure a virtual machine, you can add network adapters (NICs) and specify the adapter type.

You can add virtual Ethernet adapters to a virtual machine by using the `VMTypes.CreateSpec.setNics` method. Pass as argument a `List` of `EthernetTypes.CreateSpec` objects that provide the configuration information of the created virtual Ethernet adapters. You can set the MAC address type to `EthernetTypes.MacAddressType.MANUAL`, `EthernetTypes.MacAddressType.GENERATED`, or `EthernetTypes.MacAddressType.ASSIGNED`. Select `MANUAL` to specify the MAC address explicitly.

You can specify also the physical resources that back a virtual Ethernet adapter by using the `EthernetTypes.BackingSpec.setType` method. The method takes as argument one of the following types: `EthernetTypes.BackingType.STANDARD_PORTGROUP`, `HOST_DEVICE`, `DISTRIBUTED_PORTGROUP`, or `OPAQUE_NETWORK`.

Managing Virtual Machines

Virtual machines can be configured like physical computers. You can change the guest operating system settings after installing VMware Tools. You can add and remove virtual machines from the vCenter Server inventory. You can also move virtual machines from one host or storage location to another.

Filtering Virtual Machines

You can retrieve commonly used information about virtual machines that match specific criteria. You can retrieve information for up to 4000 virtual machines in a single vCenter Server instance.

You can retrieve a list of virtual machines in a single vCenter Server instance by filtering the results based on a specific requirement. For example, you can use as filter criteria the power state of the virtual machines, or the host, cluster, data center, folder, or resource pool that must contain the virtual machines. In case you specify multiple filter criteria, only virtual machines that match all filter criteria are returned.

To retrieve a list of the virtual machines that match your specific criteria, call the `list` methods of the `VM` service. The method takes as parameter the `VMTypes.FilterSpec` instance that you can use to provide your filter criteria.

Installing VMware Tools

VMware Tools is a set of drivers and utilities that you install on the guest operating system of a virtual machine to enhance the performance of the guest OS. VMware Tools also improve the management of the virtual machine. For each guest OS, VMware provides a specific binary-compatible version of VMware Tools.

Before you install VMware tools, you must install and boot the guest operating system.

To mount and unmount the VMware Tools installer CD as a CD-ROM for the guest operating system, use the methods of the `com.vmware.vcenter.vm.tools.Installer` interface.

To mount the VMware Tools, call the `connect(vm_ID)` method of the `Installer` interface. On Windows guest operating systems with activated Autorun feature, this method automatically initiates the installation of the VMware Tools and requires a user input to complete. On other guest operating systems, this method only mounts the VMware Tools disk image on the virtual CD/DVD drive and the user is required to do some guest OS-specific actions. For example, for some Linux distributions, the user is required to extract the contents of the VMware Tools installation archive and run the installer.

To unmount the VMware Tools installer CD image from the virtual CD/DVD drive, call the `disconnect(vm_ID)` method of the `Installer` interface.

To monitor the status of the VMware Tools installation, you can first retrieve the properties of the VMware Tools by calling the `get(vm_VM)` method of the `com.vmware.vcenter.vm.Tools` interface. The method returns a `ToolsTypes.Info` object which you can use to call the `getVersionStatus()` and `getRunState()` methods.

To upgrade the VMware Tools, call the `upgrade(vm_ID, commandLineOptions)` method of the `Tools` interface. The method takes as arguments the ID of the virtual machine on which the VMware Tools is installed and running. Use the `commandLineOptions` argument to specify the command-line options that you want to pass to the installer to modify the tools installation procedure. You can monitor the upgrade operation in the same way you monitor the installation operation.

To update the properties of the VMware Tools, call the `update(vm_ID, update_spec)` method of the `Tools` interface. Pass as argument a `ToolsTypes.UpdateSpec` object and define the tools upgrade policy settings for the virtual machine by calling the `setUpgradePolicy(upgradePolicy)` method.

Performing Virtual Machine Power Operations

You can start, stop, reboot, and suspend virtual machines by using the methods of the `Power` class.

A virtual machine can have one of the following power states:
- `PowerTypes.State.POWERED_ON` - Indicates that the virtual machine is running. If a guest operating system is not currently installed, you can perform the guest OS installation in the same way as for a physical machine.
- `PowerTypes.State.POWERED_OFF` - Indicates that the virtual machine is not running. You can still update the software on the physical disk of the virtual machine, which is impossible for physical machines.
- `PowerTypes.State.SUSPENDED` - Indicates that the virtual machine is paused and can be resumed. This state is the same as when a physical machine is in standby or hibernate state.

To perform a power operation on a virtual machine, you can use one of the methods of the `Power` class. Before you call one of the methods to change the power state of a virtual machine, you must first check the current state of the virtual machine by using the `Power.get` method. Pass as argument the virtual machine identifier.

Following is a list if the power operations:
- `Power.start` - Powers on a powered off or suspended virtual machine. The method takes as argument the virtual machine identifier.
- `Power.stop` - Powers off a powered on or suspended virtual machine. The method takes as argument the virtual machine identifier.
- `Power.suspend` - Pauses all virtual machine activity for a powered on virtual machine. The method takes as argument the virtual machine identifier.
- `Power.reset` - Shuts down and restarts the guest operating system without powering off the virtual machine. Although this method functions as a `stop` method that is followed by a `start` method, the two operations are atomic with respect to other clients, meaning that other power operations cannot be performed until the `reset` method completes.

Registering and Unregistering Virtual Machines

When you create a virtual machine, it becomes part of the vCenter Server inventory and is registered to the host and vCenter Server. If you remove a virtual machine from the vCenter Server inventory, it becomes unusable. Virtual machine files remain in the same datastore but you cannot power on the virtual machine when it is not registered in the inventory.

You can temporarily remove a virtual machine from vCenter Server by unregistering it. Virtual machine files are not deleted from the datastore. Though all high-level information about the virtual machine such as statistics, resource pool association, permissions, and alarms, is removed from the host and the vCenter Server instance. To remove a virtual machine from the inventory, call the `unregister(vm_ID)` method of the `VM` service and pass as argument the ID of the virtual machine.

To restore a virtual machine to the vCenter Server inventory, and make it usable again, call the `register(register_spec)` method of the `VM` service. You pass as argument a `VMTypes.RegisterSpec` instance that contains information about the current location of the virtual machine files on the datastore. You can also define the location within the vCenter Server inventory, for example, the cluster, folder, or the host, where you want to register the virtual machine. After registration, the virtual machine takes its resources (CPU, memory, and so on) from the resource pool or host to which it is registered.

If you no longer need a virtual machine and you want to free up datastore space, you can permanently delete a virtual machine from the inventory. Call the `delete(vm_ID)` method of the `VM` service and pass as argument the ID of the virtual machine. Upon a successful completion of the operation, the virtual machine files are removed from the datastore, including the configuration file and the virtual disk files.

Virtual Machine Guest Operations

The vSphere Automation APIs enable you to run operations on the guest operating system such as running scripts, performing power operations, and customizing the network and identity settings.

Upload and Run a Script on a Guest Operating System

You can create a directory, create files, copy a script, and run the script on a guest operating system.

Before you perform operations on the guest operating system, you must prepare the environment. You must create the `Process.CreateSpec` for initiating processes in the guest and create the `Transfer.CreateSpec` for the file transfer to or from the guest. The content of the optional `FileAttributeCreateSpec` associated with the `Transfer.CreateSpec` establishes the direction of the transfer and controls guest operating system specific file attributes. You must set up equivalent `_download` and `_upload` functions for URL management of the file transfer to or from the guest. You can also create an argument parser for standard inputs, such as server, user name, and password, and you can add custom input arguments.

1. Find the virtual machine on which the guest operating system runs, verify that VMware Tools is running, and provide credentials.
2. Create a temporary directory from which to run the script and capture any output.
3. Create temporary files for `stdout` and `stderr`.
4. Copy the script that you want to run.
5. Start the script and capture `stdout` and `stderr` in the temporary files that you created earlier.
6. Create a loop to handle processes that run longer.
7. Create a download URL and copy the results.
8. Clean up the temporary files and directories on the guest operating system.

Python Example of Uploading and Running a Script on a Guest Operating System

This example is based on the code in the `guest_ops.py` sample file.

> **NOTE**
> For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...


    # Create the Process.CreateSpec for initiating processes in the guest
    def _process_create_spec(self, path, args=None, dir=None, env={}):
        return Processes.CreateSpec(path=path,
                                    arguments=args,
                                    working_directory=dir,
                                    environment_variables=env)


    # Create the Transfer.CreateSpec for the file transfer to/from the guest
    def _create_transfer_spec(self,
```

```
                              path,
                              attributes=None):
    return Transfers.CreateSpec(attributes=attributes,
                                path=path)

# Create a FileAttributeCreateSpec for a generic (non-OS specific) guest
def _fileAttributeCreateSpec_Plain(self,
                                   size,
                                   overwrite=None,
                                   last_modified=None,
                                   last_accessed=None):
    return Transfers.FileCreationAttributes(size,
                                            overwrite=overwrite,
                                            last_modified=last_modified,
                                            last_accessed=last_accessed)

# Create a FileAttributeCreateSpec for a linux (Posix) guest
def _fileAttributeCreateSpec_Linux(self,
                                   size,
                                   overwrite=None,
                                   last_modified=None,
                                   last_accessed=None,
                                   owner_id=None,
                                   group_id=None,
                                   permissions=None):
    posix = Transfers.PosixFileAttributesCreateSpec(owner_id=owner_id,
                                                    group_id=group_id,
                                                    permissions=permissions)
    return Transfers.FileCreationAttributes(size,
                                            overwrite=overwrite,
                                            last_modified=last_modified,
                                            last_accessed=last_accessed,
```

```
                                          posix=posix)

def _download(self,
              url,
              expectedLen=None):
    urloptions = urlparse(url)
    # Skip server cert verification.
    # This is not recommended in production code.
    conn = httpclient.HTTPSConnection(urloptions.netloc,
                                      context=ssl._create_unverified_context())

    conn.request("GET", urloptions.path + "?" + urloptions.query)
    res = conn.getresponse()
    if res.status != 200:
        print("GET request failed with errorcode : %s" % res.status)
        raise HTTPError(res.status, res.reason)
    body = res.read().decode()

    return body

def _upload(self, url, body):
    urloptions = urlparse(url)
    conn = httpclient.HTTPSConnection(urloptions.netloc,
                                      context=ssl._create_unverified_context())

    headers = {"Content-Length": len(body)}
    # Skip server cert verification.
    # This is not recommended in production code.
    conn.request("PUT", urloptions.path + "?" + urloptions.query,
                 body,
                 headers)
    res = conn.getresponse()
```

```
    if res.status != 200:
        print("PUT request failed with errorcode : %s" % res.status)
        raise HTTPError(res.status, res.reason)

    return res

def __init__(self):
    # Create argument parser for standard inputs:
    # server, username, password, cleanup and skipverification
    parser = sample_cli.build_arg_parser()

    # Add your custom input arguments
    parser.add_argument('--vm_name',
                        action='store',
                        help='Name of the testing vm')
    parser.add_argument('--root_user',
                        action='store',
                        help='Administrator account user name')
    parser.add_argument('--root_passwd',
                        action='store',
                        help='Administrator account password')

    args = sample_util.process_cli_args(parser.parse_args())
    self.vm_name = args.vm_name
    self.root_user = args.root_user
    self.root_passwd = args.root_passwd

    self.cleardata = args.cleardata

    # Skip server cert verification if needed.
    # This is not recommended in production code.
    session = get_unverified_session() if args.skipverification else None
```

```
        # Connect to vSphere client
        self.client = create_vsphere_client(server=args.server,
                                            username=args.username,
                                            password=args.password,
                                            session=session)

    def run(self):
        # Using vAPI to find VM.
        filter_spec = VM.FilterSpec(names=set([self.vm_name]))
        vms = self.client.vcenter.VM.list(filter_spec)
        if len(vms) != 1:
            raise Exception('Could not locate the required VM with name ' +
                            self.vm_name + '. Please create the vm first.')
        if vms[0].power_state != 'POWERED_ON':
            raise Exception('VM is not powered on: ' + vms[0].power_state)
        vm_id = vms[0].vm

        # Check that vmtools svc (non-interactive user) is running.
        info = self.client.vcenter.vm.guest.Operations.get(vm_id)
        if info.guest_operations_ready is not True:
            raise Exception('VMware Tools/open-vm-tools is not running as required.')

        # Establish the user credentials that will be needed for all Guest Ops APIs.
        creds = Credentials(interactive_session=False,
                            user_name=self.root_user,
                            password=self.root_passwd,
                            type=Credentials.Type.USERNAME_PASSWORD)

        # Step 2 - Create a temporary directory from which to run the command and capture
any output
        tempDir = self.client.vcenter.vm.guest.filesystem.Directories.create_temporary(
                    vm_id, creds, '', '', parent_path=None)
```

```
# Step 3 - Create temproary files to reveive stdout and stderr as needed.
stdout = self.client.vcenter.vm.guest.filesystem.Files.create_temporary(
           vm_id, creds, '', '.stdout', parent_path=tempDir)
stderr = self.client.vcenter.vm.guest.filesystem.Files.create_temporary(
           vm_id, creds, '', '.stderr', parent_path=tempDir)

# Step 4 - (Optional) Copy the script to be run.
scriptPath = self.client.vcenter.vm.guest.filesystem.Files.create_temporary(
               vm_id, creds, '', '.sh', tempDir)

# Create script contents and transfer to the guest.
baseFN = os.path.basename(scriptPath)
script = ('#! /bin/bash\n'
          '#    ' +
          baseFN + '\n'
          '\n'
          'sleep 5    # Adding a little length to the process.\n'
          'ps -ef\n'
          'echo\n'
          'rpm -qa | sort\n'
          '\n')
print(script)
attr = self._fileAttributeCreateSpec_Linux(size=len(script),
                                           overwrite=True,
                                           permissions='0755')
spec = self._create_transfer_spec(path=scriptPath,
                                  attributes=attr)
toURL = self.client.vcenter.vm.guest.filesystem.Transfers.create(vm_id,
                                                                 creds,
                                                                 spec)
res = self._upload(toURL, script)
```

```
        # Check that the uploaded file size is correct.
        info = self.client.vcenter.vm.guest.filesystem.Files.get(vm_id,
                                                                 creds,
                                                                 scriptPath)
        if info.size != len(script):
            raise Exception('Uploaded file size not as epected.')


        # Step 5 - Start the program on the guest, capturing stdout and stderr in the
separate temp files obtained earlier.
        options = (" > " + stdout + " 2> " + stderr)


        spec = self._process_create_spec(scriptPath,
                                         args=options,
                                         dir=tempDir)
        pid = self.client.vcenter.vm.guest.Processes.create(vm_id, creds, spec)
        print('process created with pid: %s\n' % pid)


        # Step 6 - Need a loop to wait for the process to finish to handle longer running
processes.
        while True:
            time.sleep(1.0)
            try:
                # List the single process for pid.
                result = self.client.vcenter.vm.guest.Processes.get(vm_id,
                                                                    creds,
                                                                    pid)
                if result.exit_code is not None:
                    print('Command: ' + result.command)
                    print('Exit code: %s\n' % result.exit_code)
                    break
                if result.finished is None:
                    print('Process with pid %s is still running.' % pid)
                    continue
```

```
        except Exception as e:
            raise e

    # Step 7 Copy out the results (stdout).
    spec = self._create_transfer_spec(path=stdout)
    # Create the download URL
    fromURL = self.client.vcenter.vm.guest.filesystem.Transfers.create(vm_id,
                                                                       creds,
                                                                       spec)
    body = self._download(fromURL, expectedLen=info.size)
    print("-----------  stdout  ------------------")
    print(body)
    print("----------------------------------------")

    # Optionally the contents of "stderr" could be downloaded.

    # And finally, clean up the temporary files and directories on the
    # Linux guest.  Deleting the temporary diretory and its contents.
    self.client.vcenter.vm.guest.filesystem.Directories.delete(vm_id,
                                                               creds,
                                                               tempDir,
                                                               recursive=True)

...
```

Managing Data Sets

You can use data sets to share information between a virtual machine and its guest operating system.

Data sets provide a communication infrastructure that you can use to share information between the host and guest. This information can then be used by guest agents or applications to configure the guest itself or guest applications. The information is grouped into data sets, each of which contains key-value entries comprising the data. Each application that uses the service should have at least one unique data set in which to store its data to avoid conflict with other applications. Each data set has attributes that define its access control and interoperability configuration.

> **NOTE**
> You should not store sensitive data, such as passwords or private keys, in plain text.

To perform data set operations, the virtual machine must be running virtual hardware version 20 or later.

You should modify each data set by using the application that originally created that data set. If your application modifies a data set owned by another application, the other application might stop working.

For more information about data sets, see *vSphere Virtual Machine Administration* and the *VMware Guest SDK Programming Guide*.

Data Set Operations

You can use the vSphere Automation API to manipulate entries in data sets.

You can use data set operations to share information between a virtual machine and its guest operating system. The available operations include listing data sets, retrieving data set information, creating, modifying, and deleting a data set. You can also manipulate individual entries in a data set. The available operations include listing entry keys in a data set, retrieving the value of an entry in a data set, modifying and deleting an entry in a data set.

Table 71: User Operations

Operation	Description
List data sets	You can list all available data sets of a virtual machine.
Get data set	You can retrieve information about a specific data set.
Create data set	You can create a new data set.
Update data set	You can modify the attributes of a data set.
Delete data set	You can delete a data set.
List data set entry keys	You can list all entry keys in a data set.
Get data set entry key value	You can retrieve the value of a specific entry in a data set.
Set data set entry key value	You can create or update an entry in a data set. If the new entry does not contain a key that matches an existing key, the operation creates a new entry. If the new entry contains a key that matches an existing key, the new key overwrites the value of the existing key.
Delete data set entry key value	You can delete an entry in a data set.

Using Content Libraries

Use the vSphere Automation API to access the Content Library service and use it to store and share different types of content in your vSphere environment. Content libraries are vSphere container objects for storing and sharing OVF and OVA packages, VM templates, vApp templates, and other types of files.

You can use content libraries to share VM and vApp templates, and other types of files, such as ISO images, text files, and so on, across the vCenter Server instances in the same or different locations. Sharing templates across your virtual environment promotes consistency, compliance, efficiency, and automation in deploying workloads at scale.

You use library items to store and manage content in a content library. A single library item can contain one or more files. For example, an OVF package consists of a `.vmdk` file, manifest file, and others, but it is represented by a single library item.

Starting with vSphere 7.0 Update 3, you can add a security policy to a local or subscribed content library and thus protect your system when you synchronize or download library content from third party providers.

How to Use Content Libraries

The Content Library API provides services that allow you to create and manage content libraries programmatically. You can create a local library and publish it for the entire virtual environment. You can also subscribe to use the contents of a local library and enable automatic synchronization to ensure that you have the latest content.

Create a Local Content Library

You can create a local content library programmatically by using thevSphere Automation API. The API allows you to populate the content library with OVF and vApp templates. You can use these templates to deploy virtual machines or vApps in your virtual environment.

Required privileges:
- **Content library › Create local library** on thevCenter Server instance where you want to create the library.
- **Datastore › Allocate space** on the destination datastore.

1. Access the `LocalLibrary` service that provides support for creating local content libraries.
2. Create a `StorageBacking` instance and define the storage location.
3. Create a `SecurityPolicies` instance to define the security policy rules for the library. For vSphere 7.0 Update3, you can only define rules for an OVF and OVA templates in the library by using the `OVF_STRICT_VERIFICATION` security rule. When the OVF security policy is configured for a local content library, an OVF or OVA template can be synchronized or imported to the library only after its certificate is validated against a trusted certificate authority.
4. Create a `LibraryModel` instance and set the properties of the new local library.

   If you want to apply a security policy on the local library, you must use the `setSecurityPolicyId(java.lang.String securityPolicyId)` method or the `security_policy_id` parameter of the `LibraryModel` instance.
5. Call the create function on the `LocalLibrary` object and pass the `LibraryModel` as a parameter.

A local content library is created on thevCenter Server instance and you can edit its contents.

You maintain the contents of the local library by managing its library items. See How To Use Library Items and Content Library Support for OVF and OVA Packages. You can also share the library content by publishing the local library. See Publish an Existing Content Library.

Java

This example is based on the code in the `LibraryCrud.java` sample file.

This example uses the steps that are described in the Create a Local Content Library procedure.

> **NOTE**
> For a complete and up-to-date version of the Java sample code, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...


// Create a StorageBacking instance to back the library content on the local file
system.

    StorageBacking libraryBacking = new StorageBacking();

    libraryBacking.setType(Type.OTHER);

    libraryBacking.setStorageUri(URI.create("file:///tmp"));
```

```
    libraryModel.setStorageBackings(Collections.singletonList(libraryBacking));


// Create a LibraryModel that represents a local library.

    LibraryModel libraryModel = new LibraryModel();

    libraryModel.setType(LibraryModel.LibraryType.LOCAL);

    libraryModel.setName("AcmeLibrary");


// Access the LocalLibrary service by using the  endpoint.

    LocalLibrary localLibraryService =
this.vapiAuthHelper.getStubFactory().createStub(LocalLibrary.class,
sessionStubconfig);


// Call the create method of the LocalLibrary service passing as an

// argument the LibraryModel instance.

    String libraryId = localLibraryService.create(UUID.randomUUID().toString(),
libraryMod
```

Python

This example creates a local library with name `AcmeLibrary` , which is stored on the local file system wherevCenter Server runs.

This example uses the steps that are described in the Create a Local Content Library procedure.

> **NOTE**
> For related code samples, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...


# 1 - Create a storage backing instance on a local file system.

library_backing = library_client.StorageBacking()

library_backing.type = library_client.StorageBacking.Type.OTHER

library_backing.storage_uri = 'file:///tmp'


# 2 - Create a Library model to specify properties of the new library.

library_model = content_client.LibraryModel()

library_model.type = content_client.LibraryModel.LibraryType.LOCAL

library_model.name = 'AcmeLibrary'

library_model.storage_backings = [library_backing]
```

```
# 3 - Call the create() method, passing the library model as a parameter.

idem_token = str(uuid.uuid4())

local_library_stub = content_client.LocalLibrary(my_stub_config)

library_id = local_library_stub.create(create_spec=library_model,

                                       client_token=idem_token)
```

Publish an Existing Content Library

To make the library content available for other vCenter Server instances across the vSphere Automation environment, you must publish the library. Depending on your workflow, select a method for publishing the local library. You can publish a local library that already exists in your vSphere Automation environment.

1. Retrieve a reference to the `LocalLibrary` service.
2. Retrieve an existing local library by using the library ID.
3. Create a `PublishInfo` instance to define how the library is published.
4. Specify the authentication method to be used by a subscribed library to authenticate to the local library. You can enable either basic authentication or no authentication. Basic authentication requires a user name and password.
5. If you publish the library with basic authentication, you must specify a user name and password for the `PublishInfo` instance, which must be used for authentication.

   IMPORTANT
   Use the predefined user name `vcsp` or leave the user name undefined. You must set only a password to protect the library.

6. Set the local library to published.
7. Use the retrieved local library to configure it with the `PublishInfo` instance.
8. Update the properties of the `LibraryModel` object returned for the local library.

Java

This example is based on the code in the `LibraryPublishSubscribe.java` sample file.

NOTE
For a complete and up-to-date version of the Java sample code, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...


// Access the LocalLibrary service.

      LocalLibrary localLibraryService =
this.vapiAuthHelper.getStubFactory().createStub(LocalLibrary.class,
sessionStubconfig);



// Retrieve an existing local library.

      LibraryModel libraryModel = localLibraryService.get(libraryId);
```

```
      PublishInfo publishInfo = new PublishInfo();


// Configure how the local library is published by using BASIC authentication.
      publishInfo.setUserName("vcsp");
      publishInfo.setPassword("password".toCharArray());
      publishInfo.setAuthenticationMethod(AuthenticationMethod.BASIC);


// Set the local library to published and update the library instance.
      publishInfo.setPublished(true);
      libraryModel.setPublishInfo(publishInfo);
      localLibraryService.update(libraryModel.getId(), libraryModel);
```

Python

This example is based on the code in the `library_publish_subscribe.py` sample file.

> **NOTE**
> For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...


# Retrieve an existing local library.
local_library_stub = content_client.LocalLibrary(my_stub_config)
local_library = local_library_stub.get(my_library_id)


# Specify how the local library is published, using BASIC authentication.
publish_info = library_client.PublishInfo()
publish_info.user_name = 'vcsp' # Can omit this value; if specified, it must be
'vcsp'.
publish_info.password = 'password'
publish_info.authentication_method =
library_client.PublishInfo.AuthenticationMethod.BASIC
publish_info.published = True


# Update the LibraryModel object retieved in step 1
# and configure it with the PublishInfo object.
local_library.publish_info = publish_info
```

```
# Use the LibraryModel object to update the library instance.
local_library_stub.update(library_id=my_library_id,
update_spec=local_library)
```

Publish a Library at the Time of Creation

You can publish a local library at the time of creation to enable other libraries to subscribe and use the library content.

1. Retrieve the `LocalLibrary` service.
2. Create a `PublishInfo` instance to define how the library is published.
3. Specify the authentication method to be used by a subscribed library to authenticate to the local library.

   You can enable either basic authentication or no authentication on the library. Basic authentication requires a user name and password.

4. If you publish the library with basic authentication, you must specify a user name and password for the `PublishInfo` instance, which must be used for authentication.

   IMPORTANT
   Use the predefined user name `vcsp` or leave the user name undefined. You must set only a password to protect the library.

5. Create a `SecurityPolicies` instance to define the security policy rules for the library. For vSphere 7.0 Update3, you can only define rules for an OVF and OVA templates in the library by using the `OVF_STRICT_VERIFICATION` security rule. When the OVF security policy is configured for a local content library, an OVF or OVA template can be synchronized or imported to the library only after its certificate is validated against a trusted certificate authority.
6. Create a `LibraryModel` instance and configure the instance.

   If you want to apply a security policy on the local library, you must use the `setSecurityPolicyId(java.lang.String securityPolicyId)` method or the `security_policy_id` parameter of the `LibraryModel` instance.

7. Set the library type to local and use the configured `PublishInfo` instance to set the library to published.
8. Define where the content of the local library is stored by using the `StorageBacking` class.
9. Create a published local library.

Subscribe to a Content Library

You can subscribe to local content libraries. When you subscribe to a library, you must specify the backing storage for the library content. If the library requires basic authentication, you must also provide the correct user name and password.

Required privileges:
- **Content library › Create subscribed library** on the vCenter Server instance where you want to create the library.
- **Datastore › Allocate space** on the destination datastore.

NOTE
If you subscribe to libraries created with basic authentication on a vCenter Server instance, make sure that you pass `vcsp` as an argument for the user name.

1. Create a `StorageBacking` instance to define the location where the content of the subscribed library is stored.

2. Create a `SubscriptionInfo` instance to define the subscription behavior of the library.
   a) Provide the mechanism to be used by the subscribed library to authenticate to the published library.

      You can select between no authentication and basic authentication depending on the settings of the published library you subscribe to. If the library is published with basic authentication, you must set the basic authentication in the `SubscriptionInfo` instance. To match the credentials of the published library, set the user name and the password of the `SubscriptionInfo` instance.

   b) Provide the URL to the endpoint where the metadata of the published library is stored.
   c) Define the synchronization mechanism of the subscribed library.

      You can select between two synchronization modes: automatic and on demand. If you enable automatic synchronization for a subscribed library, both the content and the metadata are synchronized with the published library. To save storage space, you can synchronize the subscribed library on demand and update only the metadata for the published library content.

   d) Set the thumbprint that is used for validating the certificate of the published library.
3. Create a `LibraryModel` instance and set the library type to subscribed (`LibraryModel.LibraryType.SUBSCRIBED`).
4. Access the `SubscribedLibrary` service and create the subscribed library instance.

Java

This example is based on the code in the `LibraryPublishSubscribe.java` sample file.

NOTE
For a complete and up-to-date version of the Java sample code, see the `vsphere-automation-sdk-java` VMwar repository at GitHub.

```
...


// Create a StorageBacking instance to store

// the contents of the subscribed library on the local file system.

       StorageBacking libraryBacking = new StorageBacking();

       libraryBacking.setType(StorageBacking.Type.OTHER);

       libraryBacking.setStorageUri(URI.create("/mnt/nfs/cls-root"));


// Create a new SubscriptionInfo object to define the subscription

// behavior of the library.

       SubscriptionInfo subscriptionInfo = new SubscriptionInfo();

       subscriptionInfo.setAuthenticationMethod

         (com.vmware.content.library.SubscriptionInfo.AuthenticationMethod.BASIC);

       subscriptionInfo.setUserName("libraryUser");

       subscriptionInfo.setPassword("password".toCharArray());

       subscriptionInfo.setSubscriptionUrl(URI.create("https://www.acmecompary.com/
library_inventory/lib.json"));
```

```
// Specify that the content of the subscribed library will be downloaded immediately.
       subscriptionInfo.setAutomaticSyncEnabled(true);

// Set an SHA-1 hash of the SSL certificate of the remote endpoint.
subscriptionInfo.setSslThumbprint("9B:00:3F:C4:4E:B1:F3:F9:0D:70:47:48:E7:0B:D1:A7:0E:DE:60

// Create a new LibraryModel object for the subscribed library.
       LibraryModel libraryModel = new LibraryModel();
       libraryModel.setType(LibraryModel.LibraryType.SUBSCRIBED);
       libraryModel.setName("SubscrLibrary");

// Attach the storage backing and the subscription info to the library model.
       libraryModel.setStorageBackings(Collections.singletonList(libraryBacking));
       libraryModel.setSubscriptionInfo(subscriptionInfo);

// Create the new subscribed library.
       String clientToken = UUID.randomUUID().toString();
       SubscribedLibrary subscribedLibService =
this.vapiAuthHelper.getStubFactory().createStub(SubscribedLibrary.class, sessionStubconfig)
       String subscribedLibId = subscribedLibService.create(clientToken, libraryModel);
```

Python

This example is based on the code in the `library_publish_subscribe.py` sample file.

> **NOTE**
> For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...

# Create a StorageBacking instance on a local file system.
library_backing = library_client.StorageBacking()
library_backing.type = library_client.StorageBacking.Type.OTHER
library_backing.storage_uri = '/mnt/nfs/cls-root'
```

```
# Create a new SubscriptionInfo object to describe the subscription behavior.

subscription_info = library_client.SubscriptionInfo()

subscription_info.authentication_method =
library_client.SubscriptionInfo.AuthenticationMethod.BASIC

subscription_info.user_name = 'libraryUser'

subscription_info.password = 'password'

subscription_info.subscription_url = 'https://www.acmecompary.com/library_inventory/lib.jso

subscription_info.automatic_sync_enabled = True

subscription_info.ssl_thumbprint = '9B:00:3F:C4:4E:B1:F3:F9:0D:70:47:48:E7:0B:D1:A7:0E:DE:6


# Create a new LibraryModel object for the subscribed library.

library_model = content_client.LibraryModel()

library_model.type = content_client.LibraryModel.LibraryType.SUBSCRIBED

library_model.name = 'subscrLibrary'


# Attach the storage backing and the subscription info to the library model.

library_model.storage_backings = [library_backing]

library_model.subscription_info = subscription_info


# Create the new library instance.

idem_token = str(uuid.uuid4())

local_library_stub = content_client.LocalLibrary(my_stub_config)

library_id = local_library_stub.create(create_spec=library_model,client_token=idem_token)
```

Synchronize a Subscribed Content Library

When you subscribe to a published library, you can configure the settings for downloading and updating the library content.

- You can enable automatic synchronization of the subscribed library and download a copy of the content of the local library immediately.
- You can save storage space and download only the metadata for the items that are part of the local library.

To ensure that your subscribed library contains the most recent published library content, you can force a synchronization task.

1. Access the `SubscribedLibrary`vSphere Automation service.
2. Retrieve the subscribed library ID from the `SubscribedLibrary` service.
3. Force the synchronization of the subscribed library.

The synchronization operation depends on the update settings of the subscribed library. If the subscribed library is configured to update only on demand, only the metadata of the library items will be synchronized.

Editing the Settings of a Content Library

You can update the settings of content library types in your virtual environment by using the vSphere Automation API.

Table 72: Options for Updating a Content Library

Content Library Types	Option
Local content library	You can change the settings of a local library before calling the update function on the `LocalLibrary` object: Before a library is published, you can edit the name, description, version, and so on. After a library is published, you must first retrieve the `PublishInfo` instance of the published library you want. You can use the instance to configure the following settings. • Unpublish the local library. • Change the authentication method of the library. • Change the password that must be used for authentication.
Subscribed content library	You can edit the settings of a subscribed library if you retrieve the `SubscriptionInfo` instance associated with it. To apply the changes, you must update the library by using the `SubscribedLibrary` object. You can configure the following settings: • The authentication method required by the local library • The user name and password of the subscribed library for authentication to the local library • The method for synchronizing the metadata and the content of the subscribed library • The thumbprint used for validating the SSL certificate of the local library

NOTE
Starting with vSphere 7.0 Update 3, you can use the Security Policies service to secure a content library. You can set, update, or remove a security policy when you update a local or subscribed content library.

- To update or set a security policy, pass `false` to the `unset_security_policy_id` / `setUnsetSecurityPolicyId(java.lang.Boolean unsetSecurityPolicyId)` parameter and then use `security_policy_id` / `setSecurityPolicyId(java.lang.String securityPolicyId)` to pass the security policy identifier.
- To remove a security policy, pass `true` to the `unset_security_policy_id` / `setUnsetSecurityPolicyId(java.lang.Boolean unsetSecurityPolicyId)` parameter of the `LibraryModel` update specification.

Removing the Content of a Subscribed Library

You can free storage space in your virtual environment by removing the subscribed library content that you no longer need.

You can create a subscribed library with the option to download the library content on demand. As a result, only the metadata for the library items is stored in the associated with the subscribed library storage. When you want to deploy a virtual machine from a VM temple in the subscribed library, you must synchronize the subscribed library to download the entire published library content. When you no longer need the VM template, you can call the `evict` function on the `SubscribedLibrary` service. You must provide the subscribed library ID to this function. As a result, the subscribed library content that is cached on the backing storage is deleted.

If the subscribed library is not configured to synchronize on demand, an exception is thrown. In this case the subscribed library always attempts to have the most recent published library content.

Delete a Content Library

When you no longer need a content library, you can invoke the `delete` method on either the `LocalLibrary` or the `SubscribedLibrary` service depending on the library type.

1. Access the `SubscribedLibrary` or the `LocalLibrary` service by using the vSphere Automation Endpoint.
2. Retrieve the library ID you want to delete.
3. Call the `delete` function on the library service and pass the library ID as argument.
   All library items cashed on the storage backing are removed asynchronously. If this operation fails, you must manually remove the content of the library.

How To Use Library Items

A library item groups multiple files within one logical unit. Use the vSphere Automation API to perform various tasks with the items in a content library.

You can upload files to a library item in a local library and update existing items. You can download the content of a library item from a subscribed library and use the item, for example, to deploy a virtual machine. You can remove the content of a library item from a subscribed library to free storage space and keep only the metadata of the library item. When you no longer need local library items, you can delete them and they are removed from the subscribed library when a synchronization task is completed.

You can create a library item from a specific item type, for example `.ovf` and VM template. The Content Library service must support the library item type to handle the item correctly. If no support is provided for a specified type, the Content Library service handles the library item in the default way, without adding metadata to the library item or guiding the upload process. For information about the supported VM template types, see the *`vSphere Virtual Machine Administration`* documentation.

Create an Empty Library Item

You can create as many library items as needed and associate them with a local content library.

1. Access the `Item` service by using the vSphere Automation endpoint.
2. Instantiate the `ItemModel` class.
3. Define the settings of the new library item.
4. Associate the library item with an existing local library.
5. Invoke the `create` function on the `Item` object to pass the library item specification and the unique client token.

Upload content to the new library item. See Upload a File from a Local System to a Library Item and Upload a File from a URL to a Library Item.

Java

This example shows how to create an empty library item that stores an ISO image file.

> **NOTE**
> For related code samples, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...


// Create an instance of the ItemModel class and specify the item settings.

ItemModel libItemSpec = new ItemModel();

libItemSpec.setName("ESXi ISO image");

libItemSpec.setDescription("ISO image with the latest security patches for ESXi");

libItemSpec.setType("iso");


// Associate the item with an existing content library.

libItemSpec.setLibraryId("<content_library_ID>");


// Create the new Item instance, using the specified model.

Item libItemService = this.vapiAuthHelper.getStubFactory().createStub(Item.class,
sessionStubconfig);

String itemID = UUID.randomUUID().toString();

String newItem = libItemService.create(itemID, libItemSpec);
```

Python

This example shows how to create an empty library item that stores an ISO image file.

> **NOTE**
> For related code samples, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...


# 1 - Create an instance of the ItemModel class and specify the item settings.

item_model = library_client.ItemModel()

item_model.name = 'ESXi ISO image'

item_model.description = 'ISO image with the latest security patches for ESXi'

item_model.type = 'iso'
```

```
# 2 - Associate the new item with an existing library.
item_model.library_id = my_library_id


# 3 - Create the new instance of the Item class, using the specified model.
idem_token = str(uuid.uuid4())
item_stub = library_client.Item(my_stub_config)
item_id = item_stub.create(create_spec=item_model, client_token=idem_token)
```

Querying Library Items

You can perform numerous query operations on library items.

You can retrieve a list of all items in a library, retrieve a library item that has a specific type or name, and find a library item that is not cached on the disk. You can then update the library item content from the subscribed library.

List Library Items

You can use the `list` method of the `Item` object to retrieve a list of all items in a particular library.

Verify that you have access to the `Item` service.

1. Retrieve the ID of the content library whose items you want to list.
2. List the items of the specific library.
3. Retrieve a list of the files that belong to a library item.

You can see an example query operation in the code example for Edit the Settings of a Library Item. The beginning of the example lists the items of a published library and prints a list with the names and size of each file in the listed items.

List Library Items That Match Specific Criteria

You can filter the items contained in a library and retrieve only the items matching specific criteria. For example, you might want to remove or update only specific items in a library.

Verify that you have access to the `Item` service.

1. Create an instance in the `FindSpec` class.
2. Specify the filter properties by using the `FindSpec` instance.
3. List the items matching the specified filter.

A list of items matching the filter criteria is created as a result.

Edit the Settings of a Library Item

You can edit the name, description, and type of a library item.

Verify that you have access to the `Item` service.

1. Retrieve the item that you want to update.

2. Create an `ItemModel` instance.
3. Change the human-readable name and description of the library item.
4. Update the library item with the configured item model.

The settings of the library item are sucessfully edited.

Java

This example shows how to find an item by using the item name and then how to change the name and description of the retrieved item.

NOTE
For related code samples, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...


// List the items in a published library

     Item libItemService = this.vapiAuthHelper.getStubFactory().createStub(Item.class,
sessionStubconfig);

     List<String> itemIds = libItemService.list(libraryId.getId());

     for (String itemId : itemIds) {

          ItemModel singleItem = libItemService.get(itemId);


// List the files uploaded to each library item and print their names and size

          com.vmware.content.library.item.File itemFilesService =
this.vapiAuthHelper.getStubFactory().createStub(com.vmware.content.library.item.File.class,
sessionStubconfig);

          List<com.vmware.content.library.item.FileTypes.Info> fileInfos =

                 itemFilesService.list(itemId);

              for (com.vmware.content.library.item.FileTypes.Info singleFile : fileInfos) {

                  System.out.println("Library item with name "+ singleFile.getName() + " has
size

                                     " + singleFile.getSize());

              }


// Change the name and description of the library item with the specified name

                if (singleItem.getName().equals("simpleVmTemplate")) {

                     ItemModel libItemUpdated = new ItemModel();

                     libItemUpdated.setName("newItemName");

                     libItemUpdated.setDescription("Description of the newItemName");
```

```
                    libItemService.update(singleItem.getId(), libItemUpdated);
                }
            }
```

Python

This example shows how to find an item by using the item name and then how to change the name and description of the retrieved item.

NOTE
For related code samples, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...


# 1 - List the items in a published library.
item_stub = library_client.Item(my_stub_config)
item_ids = item_stub.list(my_library_id)


# 2 - List the files uploaded to each library item and print their names and sizes.
file_stub = item_client.File(my_stub_config)
for item_id in item_ids :
  item = item_stub.get(item_id)
  file_infos = file_stub.list(item_id)
  for file_info in file_infos :
    print('Library item {} has file {} with size {}'.format(item.name, file_info.name,
file_info.size))


# 3 - For a library item with a specified name,
#     create an ItemModel to change the name and description of the library item.
if item.name == 'simpleVmTemplate' :
  print('Library item {} with description {}'.format(item.name, item.description))
  item_model = library_client.ItemModel()
  item_model.name = 'newItemName'
  item_model.description = 'Description of the newItemName'
  item_stub.update(library_item_id=item_id,
                   update_spec=item_model)
  print('has been changed to:')
```

```
  print('library item {} with description {}'.format(item_model.name,
item_model.description))
```

Upload a File from a Local System to a Library Item

You can upload different types of files from a local system to a library item that you want to use in the vSphere Automation environment.

- Create an empty library item. See Create an Empty Library Item.
- Verify that you have access to the `UpdateSession` and `File` services.

1. Create an `UpdateSessionModel` instance to track the changes that you make to the library item.
2. Create an update session by using the `UpdateSession` service.
3. Create an `AddSpec` instance to describe the upload method and other properties of the file to be uploaded.
4. Create the request for changing the item by using the `File` service.
5. Upload the file that is on the local system.
6. Complete and delete the update session to apply the changes to the library item.

The file is uploaded to the library item.

Java

This example shows how to upload an ISO image file from a local system to a library item.

NOTE
For related code samples, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...


// Access the com.vmware.content.library.item.updatesession.File.

// and the UpdateSession services by using the vSphere Automation Endpoint.

    File uploadFileService =
this.vapiAuthHelper.getStubFactory().createStub(File.class, sessionStubconfig);

    UpdateSession uploadService=
this.vapiAuthHelper.getStubFactory().createStub(UpdateSession.class,
sessionStubconfig);


// Create an UpdateSessionModel instance to track the changes you make to the item.

    UpdateSessionModel updateSessionModel = new UpdateSessionModel();

    updateSessionModel.setLibraryItemId(newItem);


// Create a new update session.

    String clientToken = UUID.randomUUID().toString();
```

```
    String sessionId = uploadService.create(clientToken, updateSessionModel);


// Create an instance of the HttpClient class which is part of the
// com.vmware.vcloud.suite.samples.common package.
    try {
       HttpClient httpClient = new HttpClient(true);


// Create a new AddSpec instance to describe the properties of the file to be
uploaded.
       FileTypes.AddSpec fileSpec = new FileTypes.AddSpec();
       fileSpec.setName("ESXi patch");
       fileSpec.setSourceType(FileTypes.SourceType.PUSH);


// Link the ISO file specification to the update session.
       FileTypes.Info fileInfo = uploadFileService.add(sessionId, fileSpec);


// Use the HTTP library to upload the file to the library item.
       URI uploadUri = fileInfo.getUploadEndpoint().getUri();
       java.io.File file = new java.io.File("/updates/esxi/esxi_patch.iso");
       String transferUrl = uploadUri.toURL().toString();
       httpClient.upload(file, transferUrl);


// Mark the upload session as completed.
       uploadService.complete(sessionId);
    } finally {
         uploadService.delete(sessionId);
    }
```

Python

This example shows how to upload an ISO image file from the local system to a library item.

> **NOTE**
> For related code samples, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...
```

```
# 1 - Create an instance of the ItemModel class and specify the item settings.
item_model = library_client.ItemModel()
item_model.name = 'ESXi patches'
item_model.description = 'ESXi security patches'
item_model.type = 'iso'
item_model.library_id = my_library_id
idem_token = str(uuid.uuid4())
item_stub = library_client.Item(my_stub_config)
item_id = item_stub.create(create_spec=item_model,
client_token=idem_token)

# 2 - Create an UpdateSessionModel instance to track the changes you make to the
item.
update_session_model = item_client.UpdateSessionModel()
update_session_model.library_item_id = item_id

# 3 - Create an update session from the model.
idem_token = str(uuid.uuid4())
update_session_stub = update_session_client.UpdateSession(my_stub_config)
session_id = update_session_stub.create(create_spec=update_session_model
client_token=idem_token)

try :
  # 4 - Create a new AddSpec instance to describe the properties of the file to be
uploaded.
  file_spec = update_session_client.AddSpec()
  file_spec.name = 'ESXi patch'
  file_spec.source_type = update_session_client.File.SourceType.PUSH

  # 5 - Link the ISO file spec to the update session.
  update_file_stub = update_session_stub.File(my_stub_config)
  file_info = update_file_stub.File.add(update_session_id=session_id,
```

```
                                        file_spec=file_spec)
# 6 - Use HTTP library to upload the file to the library item.
upload_uri = file_info.upload_endpoint.uri
file_name = "/updates/esxi/esxi_patch.iso"
host = urlparse.urlsplit(upload_uri)
connection = httplib.HTTPConnection(host.netloc)
with open(file_name, "rb") as f :
connection.request("PUT", upload_uri, f)

# 7 - Commit the updates.
library_item_service.UpdateSession.complete(session_id)

finally :
# 8 - Delete the session.
library_item_service.UpdateSession.delete(session_id)
```

Upload a File from a URL to a Library Item

You can upload different types of files from a local system to a library item that you want to use in the vSphere Automationenvironment.

- Create an empty library item. See Create an Empty Library Item.
- Verify that you have access to the `UpdateSession` and `File` services.

1. Create an `UpdateSessionModel` instance to track the changes that you make to the library item.
2. Create an update session by using the `UpdateSession` service.
3. Create a file specification to describe the upload method and other properties of the file to be uploaded.
4. Specify the location of the file to be uploaded by creating a `TransferEndpoint` instance.
5. Add the file source endpoint to the file specification.
6. Create a request for changing the item by using the configured file specification.
7. Complete the update session to apply the changes to the library item.

The files are uploaded to the library item.

Java

This example shows how to upload a file from a URL to a library item. The example is based on the code in the `ItemUploadHelper.java` sample file.

NOTE
For a complete and up-to-date version of the Java sample code, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...


// Create a new library item. See Create an Empty Library
    Item.


...


// Access the com.vmware.content.library.item.updatesession.File

// and the UpdateSession services by using the vSphere Automation Endpoint.

   File uploadFileService =
this.vapiAuthHelper.getStubFactory().createStub(File.class, sessionStubConfig);

   UpdateSession uploadService =
this.vapiAuthHelper.getStubFactory().createStub(UpdateSession.class,
sessionStubConfig);


// Create an UpdateSessionModel instance to track the changes you make to the item.

        UpdateSessionModel updateSessionModel = new UpdateSessionModel();

        updateSessionModel.setLibraryItemId(newItem);


// Create a new update session.

        String clientToken = UUID.randomUUID().toString();

        String sessionId = uploadService.create(clientToken, updateSessionModel);


// Create a new AddSpec instance to describe the properties of the file to be
uploaded.

         FileTypes.AddSpec fileSpec = new AddSpec();

         fileSpec.setName("ESXi patch");

         fileSpec.setSourceType(SourceType.PULL);


// Specify the location from which the file is uploaded to the library item.

         TransferEndpoint endpoint = new TransferEndpoint();

         endpoint.setUri(URI.create("http://www.acme.com/patches_ESXi65/
```

```
ESXi_patch.iso"));
         fileSpec.setSourceEndpoint(endpoint);
         uploadFileService.add(sessionId, fileSpec);

// Mark the session as completed.
         uploadService.complete(sessionId);
```

Python

This example shows how to upload a file from a URL to a library item. The example is based on the code in the `cls_api_helper.py` sample file.

> **NOTE**
> For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...


# 1 - Create a new library item to hold the uploaded file.

item_model = library_client.ItemModel()

item_model.name = 'ESXi patches'

item_model.description = 'ESXi security patches'

item_model.type = 'iso'

item_model.library_id = my_library_id

idem_token = str(uuid.uuid4())

item_stub = library_client.Item(my_stub_config)

item_id = item_stub.create(create_spec=item_model, client_token=idem_token)


# 2 - Create an UpdateSessionModel instance to track the changes you make to the
item.

update_session_model = item_client.UpdateSessionModel()

update_session_model.library_item_id = item_id


# 3 - Create an update session from the model.

idem_token = str(uuid.uuid4())

update_session_stub = update_session_client.UpdateSession(my_stub_config)

session_id = update_session_stub.create(create_spec=update_session_model,
client_token=idem_token)


try :

  # 4 - Create a new AddSpec instance to describe the properties of the file to be
uploaded.

  file_spec = update_session_client.AddSpec()

  file_spec.name = 'ESXi patch'

  file_spec.source_type = update_session_client.File.SourceType.PULL
```

```
  # 5 - Specify the location from which the file is to be uploadod to the library
item.

  endpoint = item_client.TransferEndpoint()

  endpoint.uri = 'http://www.example.com/patches_ESXi65/ESXi_patch.iso'

  file_spec.source_endpoint = endpoint


  # 6 - Link the file specification to the update session.

  update_file_stub = update_session_client.File(my_stub_config)

  update_file_stub.File.add(update_session_id=session_id, file_spec=file_spec)


  # 7 - Mark session as completed, to initiate the asynchronous transfer.

  update_session_stub.complete(session_id)
```

Download Files to a Local System from a Library Item

You might want to download files to a local system from a library item and then make changes to the files before you use them.

1. Create a download session model to specify the item, which contains the file that you want to download.
2. Access the `File` service and retrieve the file that you want to export to your system within the new download session.
3. Prepare the files that you want to download and wait until the files are in the prepared state.
4. Retrieve the download endpoint URI of the files.
5. Download the files with an HTTP GET request.
6. Delete the download session after all files are downloaded.

The files are downloaded from the library item to the local system.

Java

This example is based on the code in the `ItemDownloadHelper.java` sample file.

> **NOTE**
> For a complete and up-to-date version of the Java sample code, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...


// Access the DownloadSession service.

     DownloadSession downloadSessionService =
vapiAuthHelper.getStubFactory().createStub(DownloadSession.class, sessionStubConfig);
```

```
// Create a new download session model.

     DownloadSessionModel downloadSessionModel = new DownloadSessionModel();

     downloadSessionModel.setLibraryItemId(libItem.getId());

     String downloadSessionId =
downloadSessionService.create(UUID.randomUUID().toString(), downloadSessionModel);



// Access the File service and retrieve the files you want to export.

     File downloadFileService =
vapiAuthHelper.getStubFactory().createStub(File.class, sessionStubConfig);

     List<FileTypes.Info> downloadFileInfos =
downloadFileService.list(downloadSessionId);

     for (FileTypes.Info downloadFileInfo : downloadFileInfos) {



// Make sure all files are in the prepared state before you precede with the
downloading operation.

          downloadFileService.prepare(downloadSessionId, downloadFileInfo.getName(),
EndpointType.HTTPS);

          long timeOut = 360;

          Long endTime = System.currentTimeMillis() + timeOut * 1000;

          try {

               Thread.sleep(5000);

          } catch (InterruptedException e) {

                    System.out.println(e);

          }

          FileTypes.PrepareStatus expectedStatus =
com.vmware.content.library.item.downloadsession.File.PrepareStatus.PREPARED;

          downloadFileInfo = downloadFileService.get(downloadSessionId,
downloadFileInfo.getName());

          FileTypes.PrepareStatus currentStatus = downloadFileInfo.getStatus();

          if (currentStatus == expectedStatus) {



// When the files are prepared, you can retrieve the download information for each
file.

             downloadFileInfo = downloadFileService.get(downloadSessionId,
downloadFileInfo.getName());
```

```
            try {
                 URI downloadUri = downloadFileInfo.getDownloadEndpoint().getUri();
                 String downloadUrl = downloadUri.toURL().toString();

// Run an HTTP GET request and pass the download endpoints of the files.
                 HttpClient httpClient = new HttpClient(true);
                 InputStream inputStream = httpClient.downloadFile(downloadUrl);
                 String fileNameDownload = downloadFileInfo.getName();
                 File tmpDir = new java.io.File("tmp");
                 tmpDir.mkdir();
                 String fullPath = tmpDir.getAbsolutePath() +
                       System.getProperty("file.separator") + fileNameDownload;

// Copy the files to the directory on your machine.
                 Files.copy(inputStream,
Paths.get(fullPath),StandardCopyOption.REPLACE_EXISTING);
            } catch (MalformedURLException e) {
               System.out.println("Failed to download due to IOException!" + e);
               throw new RuntimeException("Failed to download due to IOException!",
e);
            } catch (IOException e) {
               System.out.println("IO exception during download" + e);
               throw new RuntimeException("Failed to download due to IOException!",
e);

// Delete the download session after all files are downloaded.
            } finally {
                 downloadFileService.delete(downloadSessionId);
            }
        } else {
            while (endTime > System.currentTimeMillis()) {
                downloadFileInfo = downloadFileService.get(downloadSessionId,
                             downloadFileInfo.getName());
```

```
                currentStatus = downloadFileInfo.getStatus();

                if (currentStatus == expectedStatus) {

                    return;

                } else if (currentStatus ==
com.vmware.content.library.item.downloadsession.File.PrepareStatus.ERROR) {

                   System.out.println("DownloadSession Info : " +
downloadSessionService.get(downloadSessionId));

                  throw new RuntimeException("Error while waiting for download file
status to be PREPARED...");

              }

          }

      }

   }
```

Python

This example uses the code in the `cls_api_helper.py` sample file.

> **NOTE**
> For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...


# 1 - Create a new download session model.

download_session_model = item_client.DownloadSessionModel()

download_session_model.library_item_id = my_library_item_id

download_session_stub = item_client.DownloadSession(my_stub_config)

idem_token = str(uuid.uuid4())

download_session_id =
download_session_stub.create(create_spec=download_session_model,

client_token=idem_token)


# 2 - Access the File service and retrieve the files you want to export.

download_session_file_stub = download_session_client.File(my_stub_config)

file_infos = download_session_file_stub.list(download_session_id)

for file_info in file_infos :

  download_session_file_stub.prepare(download_session_id, file_info.name)
```

```
# 3 - Wait until the file is in the prepared state before downloading.
  download_info = download_session_file_stub.get(download_session_id, file_info.name)
  while (DownloadSessionFile.PrepareStatus.PREPARED != download_info.status) :
  time.sleep(30)

# 4 - Download the file with an HTTP GET request.
  response = urllib2.urlopen(download_info.download_endpoint.uri)
  file_path = os.path.join(my_directory, file_info.name)
  with open(file_path, 'wb') as f :
  f.write(response.read())

# 5 - Delete the download session after all files are downloaded.
download_session_stub.delete(download_session_id)
```

Synchronizing a Library Item in a Subscribed Content Library

The items in a subscribed library have features that are distinct from the items in a local library. Synchronizing the content and the metadata of an item in a subscribed library depends on the synchronization mechanism of the subscribed library.

Table 73: Options for Synchronizing a Library Item

Synchronization Type of the Subscribed Library	Description
Synchronized on demand	If the subscribed library is synchronized on demand, you can use the `sync` method on the `SubscribedItem` service and pass as arguments the library item ID and `true`. When you perform the task, both the item metadata and the content are synchronized. To synchronize only the metadata of the item, pass the library ID and `false` as arguments to the method.
Not synchronized on demand	If the subscribed library is not synchronized on demand, you can use the `sync` method on the `SubscribedItem` service and pass as argument the item ID. In this case, the content of the item is always synchronized and the Boolean value is ignored when the call is run.
Synchronized automatically	If the subscribed library is synchronized automatically, you can also use the `sync` method to force the synchronization of an item. Method execution depends on whether the subscribed library is synchronized on demand.

Removing the Content of a Library Item

You can remove the content from a library item to free space on your storage.

If you create a subscribed library with the option to synchronize library content on demand, only the metadata for the library items is stored. When you want to use the items in the library, you must force synchronization on the items to download their content. When you no longer need the files in an item, you can remove the cached content of the library item and free storage space. To achieve this task use the `evict` function of the `SubscribedItem` object.

Deleting a Library Item

You can remove a library item from a local library when you no longer need it.

To remove a library item from a library, you can call the `delete` method on the `Item` object and pass the library item ID as an argument. The item content is asynchronously removed from the storage.

You cannot remove items from a subscribed library. If you remove an item from a local library, the item is removed from the subscribed library when you perform a synchronization task on the subscribed library item.

vSphere Tag Service

The vSphere Automation Tag Service supports the definition of tags that you can associate with vSphere objects or vSphere Automation resources.

Starting with vSphere 6.5, the vSphere Automation APIs provide programmatic access to creating and managing vSphere tags in your vSphere inventory.

For example, if you want to tag your virtual machines by guest operating system type, you can create a category called `operating system`. You can specify that it applies to virtual machines only and that only a single tag can be applied to a virtual machines at any time. This category can include the following tags: `Windows`, `Linux`, and `Mac OS`.

Creating vSphere Tags

You create a vSphere tag to add metadata to objects in the vSphere inventory.

Tags are grouped in categories and each tag must have at least one category related to it. After you create the tag, you can associate the tag with a vSphere object.

Tags and categories can span multiple vCenter Server instances.
- If multiple on-premises vCenter Server instances are configured to use Enhanced Linked Mode, tags and tag categories are replicated across all these vCenter Server instances.
- When you use Hybrid Linked Mode, tags and tag categories are maintained across your linked domain. That means the on-premises SDDC and the VMware Cloud on AWS SDDC share tags and tag attributes. For more information about Hybrid Linked Mode, see "Hybrid Linked Mode" in the *`VMware Cloud on AWS Product Documentation`*.

Creating a Tag Category

You create tags in the context of a tag category. You must create a category before you can add tags within that category.

A tag category has the following properties:

- name
- description
- cardinality, or how many tags it can contain
- the types of elements to which the tags can be assigned

You can associate tags with both vSphere API managed objects and VMware vSphere Automation API resources.

Creating a Tag

After you create a tag category, you can create tags within that category.

A tag has the following properties:

- name
- description
- category ID

Java

This example is based on code in the `TaggingWorkflow.java` sample file.

This example creates a tag specification and then uses it to create the tag. The tag specification references the category identifier that was returned from the category create operation. Use the returned tag identifier for subsequent operations on the tag.

> **NOTE**
> For a complete and up-to-date version of the Java sample code, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...


Tag tagStub = myStubFactory.createStub(Tag.class,
                                       myStubConfiguration);


// Set up a tag create spec.
TagTypes.CreateSpec spec = new TagTypes.CreateSpec();
spec.setName("red");
spec.setDescription("My favorite color");
spec.setCategoryId(newCategoryId);


String tagId = tagStub.create(spec);
```

Python

This example is based on code in the `tagging_workflow.py` sample file.

This example creates a tag specification and then uses it to create the tag. The tag specification references the category identifier that was returned from the category create operation. Use the returned tag identifier for subsequent operations on the tag.

> **NOTE**
> For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...


# Set up a tag create spec.
```

```
tag_create_spec = tag_stub.CreateSpec(name='red',
                                      description='My favorite color',
                                      category_id=fav_category_id)


# Create the tag.
tag_stub = tagging_client.Tag(my_stub_config)
tag_id = tag_stub.create(create_spec)
```

Creating Tag Associations

After you create a tag category and create a tag within the category, you can associate the tag with a vSphere managed object or a vSphere Automation resource.

An association is a simple link that contains no data of its own. You can enumerate objects that are attached to a tag or tags that are attached to an object.

Tag associations are local to a vCenter Server instance. When you request a list of tag associations from a vCenter Server system, it enumerates only the associations that it has stored.

When you associate a tag with an object, the object's type must match one of the associable types specified for the category to which the tag belongs.

Assign a Tag to a Content Library

After you create a tag, you can assign the tag to avSphere Automation resource.

1. Construct a dynamic object identifier for the library.
   The dynamic identifier includes the type and ID of the object.
2. Attach the tag to the content library.

The tag is assigned to the content library.

Java

This example is based on code in the `TaggingWorkflow.java` sample file.

This example uses the steps that are described in the Assign a Tag to a Content Library procedure.

> **NOTE**
> For a complete and up-to-date version of the Java sample code, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...


// 1 - Create a dynamic type ID for the content library.
DynamicID libraryDynamicId = new DynamicID(Library.RESOURCE_TYPE,
                                           myLibrary.getId());
```

```
// 2- Attach the tag to the ClusterComputeResource managed object.
TagAssociation tagAssociationStub = myStubFactory.createStub(TagAssociation.class,
                                                             myStubConfig);
tagAssociationStub.attach(myLibrary.getId(),
                          libraryDynamicId);
```

Python

This example is based on code in the `tagging_workflow.py` sample file.

This example uses the steps that are described in the Assign a Tag to a Content Library procedure.

> **NOTE**
> For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...


# 1 - Create a dynamic type ID for the content library.
library_dynamic_id = DynamicID(type=Library.RESOURCE_TYPE,
                               id=my_library.id)


# 2- Attach the tag to the ClusterComputeResource managed object.
tag_association_stub = tagging_client.TagAssociationStub(my_stub_config)
tag_association_stub.attach(tag_id,
                            library_dynamic_id)
```

Assign a Tag to a Cluster

After you create a tag, you can assign the tag to a vSphere managed object. Tags make the inventory objects in your virtual environment more sortable and searchable.

Obtain the managed object identifier for the specified cluster.

To get the managed object identifier of the `ClusterComputeResource`, you must accessvCenter Server by using the vSphere Web Services API. For more information about how to access Web Services, see Create a Web Services Session.

This procedure describes the steps for applying tag a to a cluster object in your inventory.

1. Construct a dynamic object identifier for the cluster.

   The dynamic identifier includes the type and ID of the managed object reference.

2. Attach the tag to the cluster.

The tag is assigned to the cluster.

Java

This example is based on code in the `TaggingWorkflow.java` sample file.

NOTE
For a complete and up-to-date version of the Java sample code, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...


// 1 - Determine the MOID of the ClusterComputeResource from its name.
ManagedObjectReference clusterMoRef = VimUtil.getCluster(vimPort,
                                                         serviceContent,
                                                         myClusterName);


// 2 - Create a dynamic type ID for the cluster.
DynamicID clusterDynamicId = DynamicID(clusterMoRef.getType(),
                                       clusterMoRef.getValue());


// 3 - Attach the tag to the ClusterComputeResource managed object.
TagAssociation tagAssociationStub = myStubFactory.createStub(TagAssociation.class,
                                                             myStubConfig);
tagAssociationStub.attach(tagId,
                          clusterDynamicId);
```

Python

This example is based on code in the `tagging_workflow.py` sample file.

NOTE
For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...


# 1 - Determine the MOID of the ClusterComputeResource from its name.
cluster_object = get_obj(service_content,
                         [vim.ClusterComputeResource],
                         my_cluster_name)
cluster_moid = cluster_obj._GetMoId()
```

```
# 2 - Create a dynamic type ID for the cluster.
dynamic_id = DynamicID(type='ClusterComputeResource', id=cluster_moid)

# 3 - Attach the tag to the ClusterComputeResource managed object.
tag_association_stub = tagging_client.TagAssociation(my_stub_config)
tag_association_stub.attach(tag_id=tag_id,
                            object_id=dynamic_id)
```

Updating a Tag

To update a tag, you must create an update spec for the tag.

In the update spec, you set values for the fields to be changed, and omit values for the other fields. When you do an update operation using the update spec, only the fields that contain values are changed.

For example, you might use a timestamp in a tag description to identify a resource's last reconfiguration. After reconfiguring the resource, you update the tag description to contain the current time.

Java

This example is based on code in the `TaggingWorkflow.java` sample file.

This example adds timestamp in a tag description to identify when a resource was last reconfigured. The tag description is updated with the timestamp after the resources is reconfigured.

> **NOTE**
> For a complete and up-to-date version of the Java sample code, see the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
...

String newDateTime = Dateformat.getDateInstance().format(new Date());
String newDescription = String.format("Server tag updated at (%s).", newDateTime);

TagTypes.UpdateSpec updateSpec = new TagTypes.UpdateSpec();
updateSpec.setDescription(newDescription);
tagStub.update(myTagId, updateSpec);
```

Python

This example is based on code in the `tagging_workflow.py` sample file.

This example adds timestamp in a tag description to identify when a resource was last reconfigured. The tag description is updated with the timestamp after the resources is reconfigured.

NOTE
For a complete and up-to-date version of the sample code, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
...


tag_stub = tagging_client.Tag(my_stub_config


# 1 - Format the current time.

date_time = time.strftime('%d/%m/%Y %H:%M:%S')

description = 'Server tag updated at ' + date_time


# 2 - Set up a tag update spec.

tag_update_spec = tag_stub.UpdateSpec()

tag_update_spec.description = description


# 3 - Apply the update spec to change the tag description.

tag_stub.update(tag_id, tag_update_spec)
```

Using Tags to Create and Manage Compute Policies

Compute policies rely on tags to identify inventory objects on which to enforce a policy.

Create a Compute Policy

You can create a compute policy and check the compliance status of the policy for a specific virtual machine.

1. Retrieve the object ID of a virtual machine.
2. Retrieve the object ID of the host you want the virtual machine to run on.

   NOTE
   The host should be different from the host the virtual machine is already running on.

3. Tag the virtual machine.
   For example, use the *tag-1* tag.
4. Tag the host.
   For example, use the *tag-2* tag.
5. Create a compute policy.
   For example, use the *tag-1* and *tag-2* tags.
6. Check the compliance status of the policy on this virtual machine.

   NOTE
   The compliance status can take up to 3 minutes to update.

vSphere Supervisor Configuration and Management

Starting with vSphere 7.0, you can use the vSphere AutomationAPIs to enable and configure vSphere Supervisor on a vSphere cluster. Then you can run Kubernetes workloads directly on ESXi hosts.

You can use the vSphere Automation to automate the enabling and disabling of vSphere Supervisor on a vSphere cluster, creating and managing namespaces.

The information in this chapter is intended for vSphere administrators who want to use the vSphere AutomationAPIs to configure their environment to run Kubernetes workloads in vSphere. To take fully advantage of these vSphere AutomationAPIs, you must have basic knowledge about the Kubernetes technology and containers.

For more information about how to configure and manage vSphere Supervisor through the vSphere Client, see the *Installing and Configuring vSphere IaaS Control Plane* and *vSphere with Tanzu Configuration and Management* documentation.

vSphere Supervisor Terminology

You must understand the basic terminology in this chapter to be able use the vSphere Supervisor automation APIs effectively.

vSphere Supervisor Basic Terms

Term	Description
Supervisor	A vSphere cluster that has the vSphere Supervisor enabled.
TKG cluster	An upstream Kubernetes cluster provisioned and managed by using the TKG. A TKG resides in a vSphere Namespace. You can deploy workloads and services to such clusters in the same way as you do with standard Supervisor.
vSphere Namespace	A namespace that is created within a Supervisor. Each namespace sets the resource boundaries for CPU, memory, storage, and also the number of Kubernetes objects that can run within the namespace. After a namespace is configured, you can run Kubernetes workloads within the namespace.
vSphere Pod	A virtual machine with a small footprint that runs one or more Linux containers. A vSphere Pod is equivalent to a Kubernetes pod. vSphere Pods are compatible with the Open Container Initiative (OCI) and can run OCI compatible containers regardless the operating system.
Spherelet	A spherelet is an implementation of the kubelet functionality ported natively on each host in the Supervisor.
Kubernetes Workload	Workloads are applications that are deployed in one of the following ways: • As containers running inside vSphere Pods. • Workloads provisioned through the VM service. • TKGclusters deployed by using the TKG. • Applications running inside the TKG clusters.

Table continued on next page

Continued from previous page

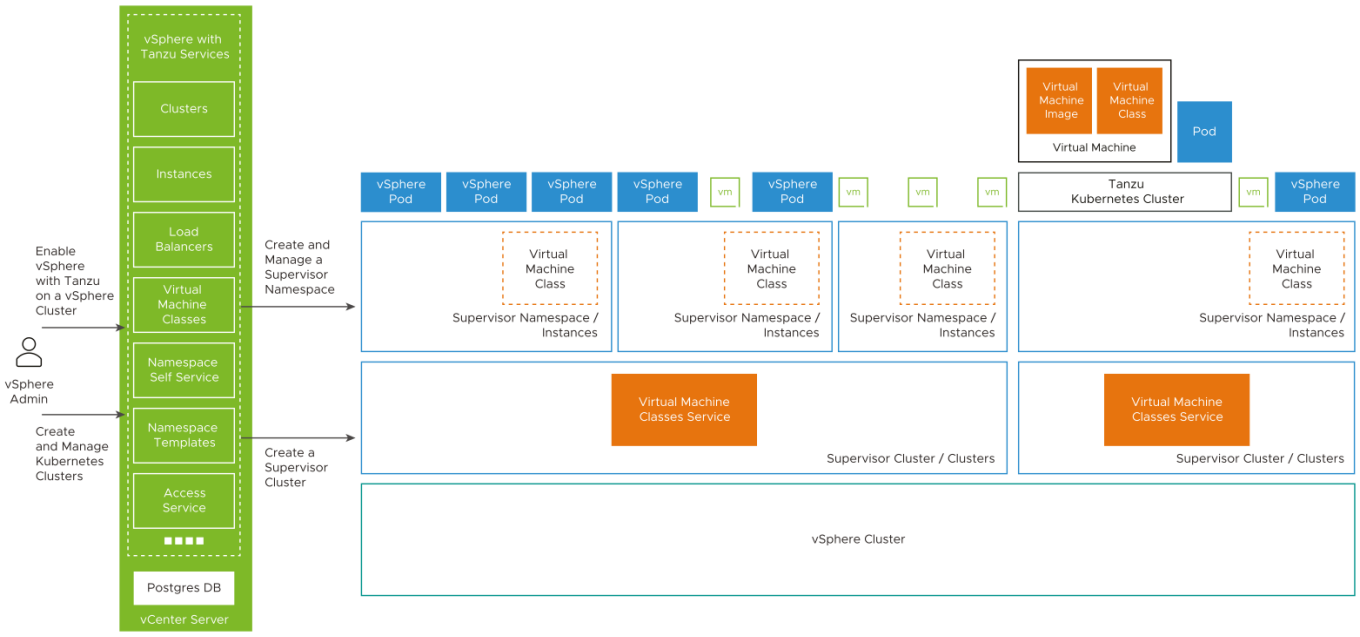
Term	Description
Supervisor control plane	vSphere Supervisor creates a Kubernetes control plane directly on the hypervisor layer. The control plane manages the worker nodes and the vSphere Pods in the Supervisor.
Supervisor worker nodes	ESXi hosts that are part of a Supervisor are considered as worker nodes. You run your Kubernetes workloads on the worker nodes.
Container Runtime Executive (CRX)	CRX is an isolated Linux execution environment similar to a VM that works together with ESXi.
VM Service	The VM Service functionality allows DevOps engineers to deploy and manage virtual machines in their Kubernetes environment through standard Kubernetes APIs. vSphere administrators are responsible for providing VM Classes and VM Images for the DevOps engineers to choose from, as well as managing resource allocations to self-service provisioned VMs.
Self-Service Namespace	vSphere administrators can activate the Self-Service Namespace service on a Supervisor and create namespace templates for DevOps engineers to create a vSphere Namespace themselves.
vSphere Zones	vSphere Zones provide high availability against clusters-level failures to workloads deployed on vSphere Supervisor. You can configure a three-zone Supervisor mapped to three vSphere clusters or a one-zone Supervisor mapped to a single vSphere cluster. In a single cluster deployment, the high availability is provided by vSphere HA and is only on a host level.

vSphere Supervisor Components and Services

Before you can automate some of the administrative tasks for using vSphere Supervisor, you must first familiarize yourself with the high-level system architecture and components involved.

The vSphere Supervisor API consists of two packages, `namespace_management` and `namespaces`. In the `namespace_management` package, you can find APIs for enabling a vSphere cluster with vSphere Supervisor, configuring the network and storage policies of the Supervisor, upgrading a cluster to the desired version of vSphere Supervisor, and so on. In the `namespaces` package, you can find APIs for creating, configuring, and deleting a vSphere Namespace, and also for setting the necessary permissions for accessing the namespace.

Figure 108: Services and Components Involved in Using vSphere Supervisor



The vSphere Kubernetes Services component runs on vCenter Server and communicates the vSphere admin requests to the Supervisor control plane. The component comprises of several services which vSphere Automation endpoints you can use to enable vSphere Supervisor on a vSphere cluster and create Kubernetes workloads.

You can use the Cluster Compatibility service to query a vCenter Server instance about the available clusters that meet the requirements for enabling vSphere Supervisor.

You can use the Clusters service to enable or disable vSphere Supervisor on a cluster. You can also reconfigure the settings of a Supervisor.

You can use the Instances service to create, edit, and delete a vSphere Namespace from a Supervisor. You can also change all or some of the settings of an existing namespace.

Starting with vSphere 7.0 Update 1, a Supervisor backed by a vSphere Distributed Switch uses the HAProxy load balancer to provide connectivity to DevOps and external service. The Load Balancer service represents the user provisioned load balancers.

Starting with vSphere 7.0 Update 2a, vSphere administrators can use the VM Service functionality to enable DevOps engineers to deploy and run VMs and containers in one shared Kubernetes environment through a single Kubernetes native interface. Use the vSphere Supervisor APIs to define VM Classes and content libraries to allocate resources to virtual machines provisioned by DevOps engineers.

As of vSphere 7.0 Update 2a, vSphere administrators can also configure a vSphere Namespace as a template on a cluster. Then the DevOps engineers can use it to self-service the creation of vSphere Namespaces and deploy workloads within them.

How to Configure and Manage a Supervisor

You use the `Clusters` service to enable and disable a Supervisor, or edit the configuration of an existing Supervisor. The `Clusters` service is provided within the `namespace_management` package.

You can enable a vSphere cluster to manage Kubernetes workload objects, only after you enable vSphere DRS in a fully automated mode and enable HA on the cluster.

Before you enable a vSphere Supervisor on a vSphere cluster, you must prepare your environment to meet the specific networking, storage, and infrastructure requirements. See the *Installing and Configuring vSphere IaaS Control Plane* documentation.

For more information about how to configure the storage settings to meet the requirements of vSphere Supervisor, see Creating Storage Policies for .

For more information about how to configure the networking settings for Supervisors that are configured with the VMware NSX-T™ Data Center as the networking stack, see Configuring for .

Starting with vSphere 7.0 Update 1, you can enable a Supervisor with vSphere networking or NSX-T Data Center, to provide connectivity between control planes, services, and workloads. A Supervisor that is configured with vSphere networking uses a vSphere Distributed Switch to provide connectivity to Kubernetes workloads and control planes. The cluster also requires a third-party load balancer that provides connectivity to DevOps users and external services. You can install in your vSphere environment the HAProxy load balancer implementation that VMware provides. See Configuring the vSphere Networking Stack for and Installing and Configuring the HAProxy Load Balancer.

Staring with vSphere 7.0 Update 2, if you are using vSphere networking, you can use the VMware NSX®Advanced Load Balancer™ to support TKG clusters provisioned by the TKG. See Using the with vSphere Networking.

Enable vSphere Supervisor on a Cluster with NSX as the Networking Stack

Through the vSphere Automation APIs, you can enable a vSphere cluster for managing Kubernetes workloads. A cluster configured with NSX supports running vSphere Pod and TKG clusters.

- Verify that your environment meets the system requirements for enabling vSphere Supervisor on the cluster. For more information about the requirements, see the *vSphere IaaS Control Plane Concepts and Planning* documentation.
- Verify that the NSX is installed and configured. See Configuring for .
- Create storage policies for the placement of pod ephemeral disks, container images, and Supervisor control plane cache.
- Verify that DRS is enabled in fully automated mode and HA is also enabled on the cluster.
- Configure shared storage for the cluster. Shared storage is required for vSphere DRS, HA, and storing persistent volumes of containers.
- Verify that the user who you use to access the vSphere Automation services has the **Modify cluster-wide configuration** privilege on the cluster.
- Create a subscribed content library on the vCenter Server system to accommodate the VM image that is used for creating the nodes of the TKG clusters.

To enable a vSphere cluster for Kubernetes workload management, you use the services under the `namespace_management` package.

1. Retrieve the IDs of the tag-based storage policies that you configured for vSphere Supervisor.

   Use the `Policies` service to retrieve a list of all storage policies and then filter the policies to get the IDs of the policies that you configured for the Supervisor.

2. Retrieve the IDs of the vSphere Distributed Switch and the Key definition for "edge_short" not found in the DITA map. cluster that you created when configuring the NSX for vSphere Supervisor.

   Use the `DistributedSwitchCompatibility` service to list all vSphere Distributed Switches associated with the specific vSphere cluster and then retrieve the ID of the Distributed Switch that you configured to handle overlay networking for the Supervisor. Use the `EdgeClusterCompatibility` service to retrieve a list of the created Key definition for "edge_short" not found in the DITA map. clusters for the specific vSphere cluster and associated with the specific vSphere Distributed Switch. Retrieve the ID of the Key definition for "edge_short" not found in the DITA map. cluster that has the tier-0 gateway that you want to use for the namespaces networking.

3. Retrieve the ID of the port group for the management network that you configured for the management traffic.

Use the `Networks` service to list the visible networks available on the vCenter Server instance that match some criteria and then retrieve the ID of the management network you previously configured.

4. Create a `ClustersTypes.EnableSpec` instance and define the parameters of the Supervisor that you want to create.

   You must specify the following required parameters of the enable specification:

   - Storage policies settings and file volume support. The storage policy you set for each of the following parameters ensures that the respective object is placed on the datastore referenced in the storage policy. You can use the same or different storage policy for the different inventory objects.

Parameter	Description
`setEphemeralStoragePolicy(java.lang.String ephemeralStoragePolicy)`	Specify the ID of the storage policy that you created to control the storage placement of the vSphere Pods.
`setImageStorage(ClustersTypes.ImageStorageSpec imageStorage)`	Set the specification of the storage policy that you created to control the placement of the cache of container images.
`setMasterStoragePolicy(java.lang.String masterStoragePolicy)`	Specify the ID of the storage policy that you created to control the placement of the Supervisor control plane cache.

     Optionally, you can activate the file volume support by using `setCnsFileConfig(CNSFileConfig cnsFileConfig)`. See Enabling ReadWriteMany Support.
   - Management network settings. Configure the management traffic settings for the Supervisor control plane.

Parameter	Description
`setNetworkProvider(ClustersTypes.NetworkProvider networkProvider)`	Specify the networking stack that must be used when the Supervisor is created. To use the NSX as the network solution for the cluster, select `NSXT_CONTAINER_PLUGIN`.
`setMasterManagementNetwork(ClustersTypes.NetworkSpec masterManagementNetwork)`	Enter the cluster network specification for the Supervisor control plane. You must enter values for the following required properties: • `setNetwork(java.lang.String network)` - Use the management network ID retrieved in Step 3. • `setMode(ClustersTypes.NetworkSpec.Ipv4Mode mode)` - Set `STATICRANGE` or `DHCP` for the IPv4 address assignment mode. The `DHCP` mode allows an IPv4 address to be automatically assigned to the Supervisor control plane by a DHCP server. You must also set the floating IP address used by the HA primary cluster by using `setFloatingIP(java.lang.String floatingIP)`. Use the DHCP mode only for test purposes. The `STATICRANGE` mode, allows the Supervisor control plane to have a stable IPv4

Table continued on next page

Continued from previous page

Parameter	Description
	address. You can use it in a production environment. • `setAddressRange(ClustersTypes.Ipv4Range addressRange)` - Optionally, you can configure the IPv4 addresses range for one or more interfaces of the management network. Specify the following settings: – The starting IP address that must be used for reserving consecutive IP addresses for the Supervisor control plane. Use up to 5 consecutive IP addresses. – The number of IP addresses in the range. – The IP address of the gateway associated with the specified range. – The subnet mask to be used for the management network.
`setMasterDNS(java.util.List<java.lang.String> masterDNS)`	Enter a list of the DNS server addresses that must be used from the Supervisor control plane. If your vCenter Server instance is registered with an FQDN, you must enter the IP addresses of the DNS servers that you use with the vSphere environment so that the FQDN is resolvable in the Supervisor. The list of DNS addresses must be specified in the order of preference.
`setMasterDNSSearchDomains(java.util.List<java.lang.String> masterDNSSearchDomains)`	Set a list of domain names that DNS searches when looking up for a host name in the Kubernetes API server. Order the domains in the list by preference.
`setMasterNTPServers(java.util.List<java.lang.String> masterNTPServers)`	Specify a list of IP addresses or DNS names of the NTP server that you use in your environment, if any. Make sure that you configure the same NTP servers for the vCenter Server instance, all hosts in the cluster, the NSX, and vSphere Supervisor. If you do not set an NTP server, VMware Tools time synchronization is enabled.

- Workload network settings. Configure the settings for the networks for the namespaces. The namespace network settings provide connectivity to vSphere Pods and namespaces created in the Supervisor.

Parameter	Description
`setNcpClusterNetworkSpec(ClustersTypes.NCPClusterNetworkEnableSpec ncpClusterNetworkSpec)`	Set the specification for the Supervisor configured with the NSX networking stack. Specify the following cluster networking configuration parameters for `NCPClusterNetworkEnableSpec`: • `setClusterDistributedSwitch(java.lang.String clusterDistributedSwitch)` -

Table continued on next page

Continued from previous page

Parameter	Description

	The vSphere Distributed Switch that handles overlay networking for the Supervisor.
	• `setNsxEdgeCluster(java.lang.String nsxEdgeCluster)` - The Key definition for "edge_short" not found in the DITA map. cluster that has tier-0 gateway that you want to use for namespace networking.
	• `setNsxTier0Gateway(java.lang.String nsxTier0Gateway)` - The tier-0 gateway that is associated with the cluster tier-1gateway. You can retrieve a list of `NSXTier0Gateway` objects associated with a particular vSphere Distributed Switch and determine the ID of the tier-0 gateway you want to set.
	• `setNamespaceSubnetPrefix(java.lang.Long namespaceSubnetPrefix)` - The subnet prefix that defines the size of the subnet reserved for namespaces segments. Default is 28.
	• `setRoutedMode(java.lang.Boolean routedMode)` - The NAT mode of the workload network. If set to `false`:
	  • The IP addresses of the workloads are directly accessible from outside the tier-o gateway and you do not need to configure the egress CIDRs.
	  • File Volume storage is not supported.
	Default is `true`.
	• `setEgressCidrs(java.util.List<Ipv4Cidr> egressCidrs)` - The external CIDR blocks from which the NSX Manager assigns IP addresses used for performing source NAT (SNAT) from internal vSphere Pods IP addresses to external IP addresses. Only one egress IP address is assigned for each namespace in the Supervisor. These IP ranges must not overlap with the IP ranges of the vSphere Pods, ingress, Kubernetes services, or other services running in the data center.
	• `setIngressCidrs(java.util.List<Ipv4Cidr> ingressCidrs)` - The external CIDR blocks from which the ingress IP range for the Kubernetes services is determined. These IP ranges are used for load balancer services and Kubernetes ingress. All Kubernetes ingress services in the same namespace share a common IP address. Each load balancer service is assigned a unique IP address. The ingress IP ranges must not overlap with the IP ranges of the

Table continued on next page

Continued from previous page

Parameter	Description
	vSphere Pods, egress, Kubernetes services, or other services running in the data center. • `setPodCidrs(java.util.List<Ipv4Cidr> podCidrs)` - The internal CIDR blocks from which the IP ranges for vSphere Pods are determined. The IP ranges must not overlap with the IP ranges of the ingress, egress, Kubernetes services, or other services running in the data center. All vSphere Pods CIDR blocks must be of at least /23 subnet size.
`setWorkerDNS(java.util.List<java.lang.String> workerDNS)`	Set a list of the IP addresses of the DNS servers that must be used on the worker nodes. Use different DNS servers than the ones you set for the Supervisor control plane.
`setServiceCidr(Ipv4Cidr serviceCidr)`	Specify the CIDR block from which the IP addresses for Kubernetes services are allocated. The IP range must not overlap with the ranges of the vSphere Pods , ingress, egress, or other services running in the data center. For the Kubernetes services and the vSphere Pods, you can use the default values which are based on the cluster size that you specify.

- Supervisor size. You must set a size to the Supervisor which affects the resources allocated to the Kubernetes infrastructure. The cluster size also determines default maximum values for the IP addresses ranges for the vSphere Pods and Kubernetes services running in the cluster. You can use the `ClusterSizeInfo.get()` calls to retrieve information about the default values associated with each cluster size.
- Optional. Associate the Supervisor with the subscribed content library that you created for provisioning TKG clusters. See Creating, Securing, and Synchronizing Content Libraries for Releases.
  To set the library, use the `setDefaultKubernetesServiceContentLibrary(java.lang.String defaultKubernetesServiceContentLibrary)` method and pass the subscribed content library ID.

5. Enable vSphere Supervisor on a specific cluster by passing the cluster enable specification to the `Clusters` service.

A task runs on vCenter Server for turning the cluster into a Supervisor. Once the task completes, Kubernetes control plane nodes are created on the hosts that are part of the cluster enabled with vSphere Supervisor. Now you can create vSphere Namespaces.

Create and configure namespaces on the Supervisor. See Create a .

Java

This example enables vSphere Supervisor on a cluster that has NSX-T configured as the networking stack.

The following code snippet is part of the `EnableSupervisorCluster.java` sample. Some parts of the original code sample are omitted to save space. You can view the complete and up-to-date version of this sample in the `vsphere-automation-sdk-java` VMware repository at GitHub.

```
      (...)
@Override
  protected void run() throws Exception {


      System.out.println("We are building the Spec for enabling vSphere supervisor
cluster");


      ClustersTypes.EnableSpec spec = new ClustersTypes.EnableSpec();

      (...)

      spec.setSizeHint(SizingHint.TINY);

      (...)

      spec.setServiceCidr(serCidr);

      spec.setNetworkProvider(ClustersTypes.NetworkProvider.NSXT_CONTAINER_PLUGIN);

      (...)

      spec.setNcpClusterNetworkSpec(NCPSpec);

      (...)

      spec.setMasterManagementNetwork(masterNet);

      (...)

      spec.setMasterDNS(masterDNS);

      (...)

      spec.setWorkerDNS(workerDNS);

      (...)

      spec.setMasterNTPServers(NTPserver);


      spec.setMasterStoragePolicy(this.storagePolicyId); // Storage policy identifier

      spec.setEphemeralStoragePolicy(this.storagePolicyId);// Storage policy
identifier

      spec.setLoginBanner("This is your first Project pacific cluster");

      (...)

      spec.setImageStorage(imageSpec);


      this.ppClusterService.enable(clusterId, spec);

      System.out.println("Invocation is successful for enabling vSphere supervisor
```

```
cluster, check H5C");


  }

      (...)
```

Enable vSphere Supervisor on a Cluster with the vSphere Networking Stack

Starting with vSphere 7.0 Update 1, you can select between creating a Supervisor with the vSphere networking stack or with NSX as the networking solution. A Supervisor that is configured with the vSphere networking stack only supports TKG clusters. vSphere Pods are not supported.

- Verify that your environment meets the system requirements for enabling vSphere Supervisor on the cluster. For more information about the requirements, see the documentation.
- Verify that DRS is enabled in fully automated mode and HA is also enabled on the cluster.
- Configure shared storage for the cluster. Shared storage is required for vSphere DRS, HA, and storing persistent volumes of containers.
- Create storage policies for the placement of Kubernetes control planes.
- Create a subscribed content library on the vCenter Server system to accommodate the VM image that is used for creating nodes of TKG clusters. See Creating, Securing, and Synchronizing Content Libraries for Releases.
- Add all hosts from the cluster to a vSphere Distributed Switch and create port groups for workload networks. See Configuring the vSphere Networking Stack for .
- Configure an HAProxy load balancer instance that is routable to the vSphere Distributed Switch that is connected to the hosts from the vSphere cluster.
- Verify that the user who you use to access the vSphere Automation services has the **Namespaces.Manage** privilege on the cluster.

To enable a cluster configured with the vSphere networking stack for Kubernetes workloads management, you must use the services under the `namespace_management` package.

1. Retrieve the ID of the cluster which hosts were added to the vSphere Distributed Switch.

   Use the `ClusterCompatibility` service to filter the clusters by using their network providers. To retrieve a list of all clusters in the vCenter Server system which are configured with the vSphere networking stack, set the network provider in the filter specification to `VSPHERE_NETWORK`.

2. Retrieve the IDs of the tag-based storage policies that you configured for vSphere Supervisor.

   Use the `Policies` service to retrieve a list of all storage policies and then filter the policies to get the IDs of the policies that you configured for the Supervisor.

3. Retrieve the ID of the port group for the management network that you configured for the management traffic.

   To list the visible networks available on the vCenter Server instance that match some criteria and then retrieve the ID of the management network you previously configured, use the `Networks` service.

4. Create a Supervisor enable specification and define the parameters of the Supervisor that you want to enable.

   You must specify the following required parameters of the enable specification:

   - Supervisor size. You must set a size to the Supervisor which affects the resources allocated to the Kubernetes infrastructure. The cluster size also determines default maximum values for the IP addresses ranges for the vSphere Pod and Kubernetes services running in the cluster. You can use the to retrieve information about the default values associated with each cluster size.

- Storage policy settings and file volume support. To specify the ID of the storage policy that you created to control the placement of the Supervisor control plane cache, use. Optionally, you can activate the file volume support by using . See Enabling ReadWriteMany Support.
- Load balancer. To specify the user-provisioned load balancer configuration for the cluster, use the `setLoadBalancerConfigSpec(LoadBalancersTypes.ConfigSpec loadBalancerConfigSpec)` parameter of the enable specification. You must specify the following parameters of the `LoadBalancersTypes.ConfigSpec` specification:

Parameter	Description
`setId(java.lang.String id)`	A user-friendly name of the load balancer. The name must be an alphanumeric string with a maximum length of 63 characters which is unique across the namespaces in the vCenter Server instance.
`setProvider(LoadBalancersTypes.Provider provider)`	The type of the load balancer that you want to use. In vSphere 7.0 Update 2, you can choose between the HAProxy load balancer and the Avi Load Balancer. Pass as a value to this parameter one of the following constants: `HA_PROXY` or `AVI`.
`setAddressRanges(java.util.List<IPRange> addressRanges)`	The IP address ranges in CIDR format from which HAProxy allocates the IP addresses for the virtual servers. You must provide at least one IP range which is reserved by HAProxy. The CIDR range specified with this parameter must not overlap with the IPs allocated for the Kubernetes control planes and workloads. The IP range that you configure must be on a separate subnet.
`setHaProxyConfigCreateSpec(LoadBalancersTypes.HAProxyConfigCreateSpec haProxyConfigCreateSpec)`	The HAProxy runtime configuration. See Installing and Configuring the HAProxy Load Balancer.
`setAviConfigCreateSpec(LoadBalancersTypes.AviConfigCreateSpec aviConfigCreateSpec)`	The Avi Load Balancer configuration. See Using the with vSphere Networking.

- Management network settings. Configure the network parameters for the Kubernetes control planes.

Parameter	Description
`setNetworkProvider(ClustersTypes.NetworkProvider networkProvider)`	Specify the networking stack that must be used when the Supervisor is created. To use the vSphere network as the solution for the cluster, select `VSPHERE_NETWORK`.

Table continued on next page

Continued from previous page

Parameter	Description
`setMasterManagementNetwork(ClustersTypes.NetworkSpec masterManagementNetwork)`	Enter the cluster network specification for the Supervisor control plane. You must enter values for the following required properties: • `setNetwork(java.lang.String network)` - Use the management network ID retrieved in Step 3. • `setMode(ClustersTypes.NetworkSpec.Ipv4Mode mode)` - Set `STATICRANGE` or `DHCP` for the IPv4 address assignment mode. The `DHCP` mode allows an IPv4 address to be automatically assigned to the Supervisor control plane by a DHCP server. You must also set the floating IP address used by the HA primary cluster by using `setFloatingIP(java.lang.String floatingIP)`. Use the DHCP mode only for test purposes. The `STATICRANGE` mode, allows the Supervisor control plane to have a stable IPv4 address and can be used in a production environment.
`setMasterDNS(java.util.List<java.lang.String> masterDNS)`	Enter a list of the DNS server addresses that must be used from the Supervisor control plane. If your vCenter Server instance is registered with an FQDN, you must enter the IP addresses of the DNS servers that you use with the vSphere environment so that the FQDN is resolvable in the Supervisor. The list of DNS addresses must be specified in the order of preference.
`setMasterDNSSearchDomains(java.util.List<java.lang.String> masterDNSSearchDomains)`	Set a list of domain names that DNS searches inside the Kubernetes control plane nodes, so that the DNS server can resolve them. Order the domains in the list by preference.
`setMasterNTPServers(java.util.List<java.lang.String> masterNTPServers)`	Specify a list of IP addresses or DNS names of the NTP server that you use in your environment, if any. Make sure that you configure the same NTP servers for the vCenter Server instance, all hosts in the cluster, and vSphere Supervisor. If you do not set an NTP server, VMware Tools time synchronization is enabled.

- Workload network settings. Configure the settings for the network that will handle the networking traffic for Kubernetes workloads running on the Supervisor.

Parameter	Description
`setServiceCidr(Ipv4Cidr serviceCidr)`	Specify the CIDR block from which the IP addresses for Kubernetes services are allocated. The IP range must not overlap with the ranges of the vSphere Pods

Table continued on next page

Continued from previous page

Parameter	Description
	, ingress, egress, or other services running in the data center. For the Kubernetes services and the vSphere Pods, you can use the default values which are based on the cluster size that you specify.
`setWorkloadNetworksSpec(ClustersTypes.WorkloadNetworksEnableSpec workloadNetworksSpec)`	Enter the workload networks specifications for the cluster. To configure the primary workload network that is used to expose the Supervisor control plane to DevOps and other workloads, create a `NetworksTypes.CreateSpec` instance. Enter the following parameters of the vSphere Distributed Switch: • `setNetwork(java.lang.String network)`. The name of the vSphere Distributed Switch that is associated with the hosts in the cluster. The name must be a unique alphanumeric string that does not exceed 63 characters. • `setNetworkProvider(ClustersTypes.NetworkProvider networkProvider)`. Pass `VSPHERE_NETWORK` as value to this parameter. • `setVsphereNetwork(NetworksTypes.VsphereDVPGNetworkCreateSpec vsphereNetwork)`. Optionally, you can create a `NetworksTypes.VsphereDVPGNetworkCreateSpec` instance to describe the configuration of the namespace network backed by the vSphere Distributed port group. You must define the following parameters for the vSphere Distributed port group specification: – `setPortgroup(java.lang.String portgroup)`. Specify the port group that serves as the primary network to the Supervisor. – `setAddressRanges(java.util.List<IPRange> addressRanges)`. Set the IP range for allocating IP addresses for the Kubernetes control planes and workloads. You must use unique IP ranges for each workload network. – `setGateway(java.lang.String gateway)`. Set the gateway for the primary network. – `setSubnetMask(java.lang.String subnetMask)`. Specify the subnet mask of the network.

- Content library settings. Add the subscribed content library that contains the VM images for deploying the nodes of TKG clusters. See Creating, Securing, and Synchronizing Content Libraries for Releases.
  To set the library, use `setDefaultKubernetesServiceContentLibrary(java.lang.String defaultKubernetesServiceContentLibrary)` and pass the subscribed content library ID.

5. Enable the Supervisor by passing the enable specification to the `Clusters` service.

A task runs on vCenter Server for enabling vSphere Supervisor on the cluster. Once the task completes, three Kubernetes control planes are created on the hosts that are part of the cluster.

Create and configure namespaces on the Supervisor.

Upgrading a Supervisor

You can use the vSphere SupervisorAPIs to upgrade a single or a group of clusters to a specific version.

vSphere Supervisor supports rolling upgrades through the vSphere AutomationAPIs for Supervisors and for the infrastructure supporting these clusters. This model ensures that there is minimal downtime for the cluster workloads during the upgrade process.

To retrieve a list of all available vSphere Supervisor upgrade versions for a specific vCenter Server system, use the Cluster Available Versions service. You can get information about the release version, name, description, release date, and release notes for each available upgrade.

You must use the Software Clusters service for upgrading a Supervisor. You can retrieve upgrade information about all Supervisors enabled on a vCenter Server system by using the . You receive a list of basic upgrade-related information for each cluster, such as the current software version, the date of the last successful upgrade, the upgrade status of the cluster, and so on. In case some of the clusters are in the process of upgrading, you can retrieve also information about their desired upgrade version. If you want to view a more detailed upgrade-related information about a cluster, you must use the .

After you view the details about the upgrade versions that you can apply on a single or multiple Supervisors, you can create upgrade specifications that define the versions you want to upgrade to. When you upgrade a batch of Supervisor and for some reason one of the clusters fails to upgrade, you receive information about the pre-check exceptions that led to that cluster upgrade failure.

Monitoring the Enable and Upgrade Supervisor Operations

When you run the Supervisor enable and upgrade operations, the status of the tasks is not returned. Since these operations might be time-consuming but critical when automating second- and third-party products with vSphere Supervisor, you can write a logic to track the task status.

To monitor the status of the enable and upgrade Supervisor operations, use the and query the Kubernetes and configuration status of the cluster. Track the status every two minutes or so, until you receive `READY` for and `RUNNING` for . These statuses indicate that the Supervisor reached the desired configuration status and is ready for running Kubernetes workloads.

Reconfiguring a Supervisor

You can change some or all the predefined settings of a Supervisor through the vSphere AutomationAPIs.

To update only some of the Supervisor settings, . To reconfigure entirely the Supervisor, .

The settings you can configure with the `UpdateSpec` and `SetSpec` specifications are the same as the ones that you used for enabling the Supervisor. For example, you can change the storage settings on the Supervisor. Note that the changes that you make to the storage settings after the initial cluster configuration, apply only to the newly created Supervisor control planes.

Deactivating a Supervisor

You can programmatically deactivate vSphere Supervisor on a vSphere cluster by using the vSphere AutomationAPIs.

When you deactivate a Supervisor, the vSphere Kubernetes Service forcefully deletes from the cluster all objects and configurations part of the Kubernetes infrastructure. To deactivate Kubernetes workloads on a cluster,

vSphere Security

This chapter provides information about securing your vSphere environment for vCenter Server and ESXi.

Managing Certificates

Starting with vSphere 6.7 Update 2, you can use the vSphere Automation API to manage certificates in your vSphere environment. You can not only refresh default certificates that are issued by the VMware Certificate Authority (VMCA) but also add third-party or custom-made certificates to your environment.

Certificate Management Operations

Use the vSphere Automation API to manage trusted root certificate chains, VMware Certificate Authority (VMCA) root certificates, machine SSL (TLS) certificates, and Security Token Service (STS) signing certificates.

With the vSphere Automation API, you can refresh the VMCA-issued certificates but also add external and third-party certificates to your vSphere environment. For more information on vSphere certificate management, see the *vSphere Authentication* guide.

Location of the Certificate Management Services

You can find the vSphere certificate management services for your automation in the `com.vmware.vcenter.certificate_management.vcenter` Java package and the `com.vmware.vcenter.certificate_management.vcenter_client` Python module.

Certificate Management Interfaces

You can use the following interfaces to manage certificates with the vSphere Automation API:

- `TrustedRootChains`
- `VMCARoot`
- `TLSCSR`
- `TLS`
- `SigningCertificate`

Certificate Management Operations

You can use the operations listed in the following table to manage certificates.

Table 74: Certificate Management Operations

Operation	Interface	Java / Python Operation	Description	Introduced in
List trusted root certificates	`TrustedRootChains`	`list()`	You can retrieve the identifiers of all trusted root certificates that are present in vCenter Server.	vSphere 6.7 U2

Table continued on next page

Continued from previous page

Operation	Interface	Java / Python Operation	Description	Introduced in
Get trusted root certificate information	`TrustedRootChains`	`get(chain)`	You can retrieve a root certificate chain by providing its identifier. You can retrieve the identifier by using the List trusted root certificates operation.	vSphere 6.7 U2
Add a trusted root certificate	`TrustedRootChains`	`create(spec)`	You can add a trusted root certificate chain to your vCenter Server system.	vSphere 6.7 U2
Delete a trusted root certificate	`TrustedRootChains`	`delete(chain)`	You can delete a root certificate by providing its unique identifier. You can retrieve the identifier by using the List trusted root certificates operation.	vSphere 6.7 U2
Replace the VMCA root certificate	`VMCARoot`	`create(spec)`	You can replace the VMCA root certificate with a new VMCA-signed certificate. The operation triggers a restart of the services that are using this certificate.	vSphere 7.0
Generate a CSR	`TLSCSR`	`create(spec)`	You can generate a CSR and use it to issue a custom certificate from the given spec.	vSphere 6.7 U2
Get the Machine SSL certificate	`TLS`	`get()`	You can retrieve the machine SSL certificate of your vCenter Server system.	vSphere 6.7 U2
Renew the Machine SSL certificate	`TLS`	`renew(duration)`	You can renew the validity of the machine SSL certificate for a specified period in days. The duration must be less than or equal to 730 days. If you pass `null`/`None`, the default duration of 730 days is applied.	vSphere 6.7 U2
Replace the Machine SSL certificate with a custom signed certificate	`TLS`	`set(spec)`	You can replace the vCenter Server machine SSL certificate with a custom certificate signed by an external or a third-party Certificate Authority (CA).	vSphere 6.7 U2
Replace the Machine SSL certificate with a VMCA-signed certificate	`TLS`	`replaceVmcaSigned(spec)`	You can replace the vCenter Server machine SSL certificate with a VMCA-signed certificate.	vSphere 7.0

Table continued on next page

Continued from previous page

Operation	Interface	Java / Python Operation	Description	Introduced in
Retrieve the STS signing certificate chains	`SigningCertificate`	`get()`	You can retrieve the STS signing certificate chains, which are used for validating tokens signed by vCenter Server.	vSphere 7.0 U3
Replace the STS signing certificate	`SigningCertificate`	`set(spec)`	You can replace the current STS signing certificate with a certificate of your choice. The accepted file format is PEM.	vSphere 7.0 U3
Refresh the STS signing certificate	`SigningCertificate`	`refresh(force)`	You can replace the current STS signing certificate with a new VMCA-signed certificate. The newly-generated certificate is set as the active STS signing certificate for the vCenter Server token service. You can pass `true` as a value to the `force` parameter for environments that might otherwise prevent the operation from succeeding.	vSphere 7.0 U3

Add a Root Certificate to vCenter Server

You can use the `TrustedRootChains` interface to add, delete and read trusted root certificate chains.

- Verify that the root certificate or certificate chain you want to add is available on your machine.
- Verify that you have the required privileges: **CertificateManagement › Manage** and **CertificateManagement › Administer**.

If you want to use an enterprise or third-party certificate authority (CA) for certificate management of your vSphere environment, you must first establish trust with that CA. You can do this by adding the root certificate of the external CA to the trusted root store of your vCenter Server system.
Adding a root certificate or certificate chain to the vCenter Server trusted certificate store establishes trust with an enterprise or third-party certificate authority. You can add a root certificate to vCenter Server as a prerequisite for other scenarios such as setting a third-party or enterprise machine SSL certificate.

1. Retrieve the root certificates on your vCenter Server system by calling the `list` function of the `TrustedRootChains` interface.
2. Create a `X509CertChain` instance with the root certificate you want to add.
3. Create a specification with the new `X509CertChain` instance.
4. To add the root certificate, call the `create` function of the `TrustedRootChains` interface.

If the operation is successful, the system returns the unique identifier of the trusted root certificate you added.

Delete a Root Certificate from vCenter Server

You can use the `TrustedRootChains` interface to add, delete and read trusted root certificate chains. This use case demonstrates how to delete a root certificate or certificate chain from the trusted root store of your vCenter Server system.

Verify that you have the required privileges for executing the method: **CertificateManagement › Administer** and **CertificateManagement › Manage**.

Deleting certificates is not available through the vSphere Client and you can only do this by using the vSphere Automation API or the CLI tools.

CAUTION
Deleting a root certificate or certificate chain that is in use might cause breakage of your systems. Proceed to delete a root certificate only if you are sure it is not in use by your vCenter Server or any connected systems.

1. Retrieve the root certificates from your vCenter Server system by calling the `list` function of the `TrustedRootChains` interface.
   The system lists the unique identifiers (chains) of the certificates in the trusted certificate store.
2. Retrieve the certificate you want to delete by calling the `get` function and passing the unique identifier (chain) of the certificate as an argument.
3. Delete the certificate by using the delete function of the `TrustedRootChains` interface and passing the unique identifier (chain) of the certificate as an argument.
4. To verify you deleted the certificate, list the root certificates from your vCenter Server system once again.

Change the Machine SSL Certificate of vCenter Server

You can change the machine SSL certificate of a vCenter Server system by using the TLS and the TLS CSR interfaces of the vSphere Automation API.

- Verify that the root certificate of the CA you are going to use is available on your machine.
- Verify that you have the required privileges: **CertificateManagement › Administer** and **CertificateManagement › M anage**.

The machine SSL certificate is used for server verification and for secure communication such as HTTPS or LDAPS. The machine certificates are the human-facing certificates in vSphere. They are used to create an SSL socket on the server side to which SSL clients can then connect.
Changing the machine SSL certificate with one issued by an official or enterprise certificate authority is an essential part of the Hybrid Mode of vSphere certificate management. In this mode, you replace the machine SSL certificate and you leave the VMCA to manage all other certificates automatically. The VMCA is a just-enough internal certificate authority that comes integral with your vSphere deployment. It has been purpose-built to serve the certificate needs of your vSphere environment. For more information on vSphere certificate management, see the *`vSphere Authentication`* guide.

1. Retrieve the current machine SSL certificate of your vCenter Server system by calling the `get` function of the `Tls` interface.
2. Generate a certificate signing request (CSR) by using the `TlsCsr` interface.
   a) Create a new object of type `TlsCsrTypes.Spec` by using the following parameters/methods.

Parameter/Method	Description
`setCountry` / `country`	Specify the country in the certificate subject.
`setStateOrProvince` / `state_or_province`	Specify the state or province in the certificate subject.
`setLocality` / `locality`	Specify the locality in the certificate subject.
`setOrganization` / `organization`	Specify the organization in the certificate subject.
`setOrganizationUnit` / `organization_unit`	Specify the organization unit in the certificate subject.
`setEmailAddress` / `email_address`	Specify the email address in the certificate subject.

   b) Create the CSR by calling the `create(TlsCsrTypes.Spec)` method and passing the CSR specification.
   The system returns the CSR in PEM format.
3. Save the CSR to your machine.
4. Send the CSR to the certificate authority of your choice.

NOTE
The private key corresponding to the public key generated by the CSR is stored in the vCenter Server keystore and does not leave your system.

5. Save the issued third-party machine SSL certificate to your machine.
6. Set the new custom certificate to your vCenter Server system by using the `Tls` service.
   a) Create a new object of type `TlsTypes.Spec` by using the following parameters/methods.

Parameter/Method	Description
`setCert` / `cert`	The Machine SSL certificate in PEM format. You must also paste the intermediate CA certificate, if you have one.
`setRootCert` / `root_cert`	The third-party root CA certificate in PEM format. You must also paste the intermediate CA certificate, if you have one.

   You do not provide the private key as it was generated with the CSR and is already present on your system.

   b) Set the new certificate to your vCenter Server system by calling the `set(TlsTypes.Spec)` method and passing the TLS specification as an argument.

NOTE
Restart of the vCenter Server services after the certificate change is not necessary.

The certificate replacement is completed seamlessly and all your sessions remain active.

Python

- *Generating a Certificate Signing Request (CSR) from vCenter Server*

  This example shows how to generate a certificate signing request (CSR) from vCenter Server. The generation of a CSR is an important step in the workflow to change the machine SSL certificate of your vCenter Server system with a third-party or enterprise certificate. The example is based on the code in the `gencsr.py` sample file.

  NOTE
  For related code samples, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
"""

Description: Demonstrates the generation of the Certificate Signing request

for the MACHINE SSL certificate

Sample Prerequisites:

- The user invoking the API should have the CertificateManagement.Administer or
the
```

```
CertificateManagement.Manage privilege.
"""

parser = sample_cli.build_arg_parser()

parser.add_argument('--keysize',
                    help='Key size used to generate the private key.'
                         'keysize will take 2048 bits if not modified')

parser.add_argument('--commonname',
                    help='Common name of the certificate subject field.'
                         'common name will take the Primary Network
Identifier(PNID) if not modified.')

parser.add_argument('--organization',
                    required=True,
                    help='Organization field in certificate subject.')

parser.add_argument('--organizationunit',
                    required=True,
                    help='Organization unit field in certificate subject')

parser.add_argument('--locality',
                    required=True,
                    help='Locality field in the certificate subject')

parser.add_argument('--stateorprovince',
                    required=True,
                    help='State field in certificate subject')

parser.add_argument('--country',
                    required=True,
```

```
                    help='Country field in the certificate subject')

parser.add_argument('--emailaddress',
                    required=True,
                    help='Email field in Certificate extensions')

parser.add_argument('--subjectaltname',
                    help='subjectaltname is list of Dns Names and Ip addresses')

args = sample_util.process_cli_args(parser.parse_args())

session = requests.session()
session.verify = False if args.skipverification else True

# Login to vCenter
vsphere_client = create_vsphere_client(server=args.server,
                                       username=args.username,
                                       password=args.password,
                                       session=session)

common_name = args.commonname
organization = args.organization
organization_unit = args.organizationunit
locality = args.locality
state_or_province = args.stateorprovince
country = args.country
email_address = args.emailaddress

if args.keysize is None:
    key_size = args.keysize
else:
```

```
    key_size = int(args.keysize)


if args.subjectaltname is None:
    subject_alt_name = args.subjectaltname
else:
    subject_alt_name = args.subjectaltname.split(',')


"""
Create the spec for input to the API
"""
spec = TlsCsr.Spec(key_size=key_size,
                   common_name=common_name,
                   organization=organization,
                   organization_unit=organization_unit,
                   locality=locality,
                   state_or_province=state_or_province,
                   country=country,
                   email_address=email_address,
                   subject_alt_name=subject_alt_name)


print('Generating the certificate signing request based on the information
provided in the spec ')
print(vsphere_client.vcenter.certificate_management.vcenter.TlsCsr.create(spec))
```

- *Setting a New Machine SSL Certificate to vCenter Server*

  This example shows how to set a a third-party or enterprise machine SSL certificate to your vCenter Server system. You must have already completed the step of generating a certificate signing request (CSR) and obtained the new certificate from an enterprise or third-party certificate authority. This example is based on the code in the `replace_tls_certificate.py` sample file.

  NOTE
  For related code samples, see the `vsphere-automation-sdk-python` VMware repository at GitHub.

```
"""
Description: Demonstrates the replacement of the MACHINE SSL certificate with a
custom
```

```
certificate signed by an external third party CA.

Sample Prerequisites:

- The user invoking the API should have the CertificateManagement.Administer
privilege.

"""


parser = sample_cli.build_arg_parser()


parser.add_argument('--cert',

                    required=True,

                    help='Leaf certificate for replace the MACHINE SSL
certificate.')


parser.add_argument('--key',

                    help='The private key.'

                         'Not required if the gencsr api was used to generated
the certificate signing request.')


parser.add_argument('--rootcert',

                    help='The root certificate and the intermediate root
certificates '

                         'required to establish the chain of trust.'

                         'Not required if the certificates are already present in
the vCenter.')


args = sample_util.process_cli_args(parser.parse_args())


session = requests.session()

session.verify = False if args.skipverification else True


# Login to vCenter

vsphere_client = create_vsphere_client(server=args.server,

                                       username=args.username,

                                       password=args.password,
```

```
                                        session=session)

cert = args.cert.encode(encoding='utf-8').decode('unicode_escape')

if args.key is not None:
    key = args.encode(encoding='utf-8').key.decode('unicode_escape')
else:
    key = args.key

if args.rootcert is not None:
    root_cert = args.rootcert.encode(encoding='utf-8').decode('unicode_escape')
else:
    root_cert = args.rootcert

"""
Create the spec for input to the API
"""
spec = Tls.Spec(cert=cert,
                key=key,
                root_cert=root_cert)



print('The MACHINE SSL certificate will be replaced with the custom certificate
')
vsphere_client.vcenter.certificate_management.vcenter.Tls.set(spec)
```

Refresh the vCenter Server STS Signing Certificate with a VMCA-Issued Certificate

You can refresh the vCenter Server Security Token Service (STS) signing certificate with a new VMCA-issued certificate by using the `SigningCertificate` interface. The STS is an internal entity that issues and verifies tokens so that vSphere services can communicate with and trust each other.

Verify that you have the required privilege: **CertificateManagement › Administer**.

You can refresh the current STS signing certificate of your vCenter Server system with a new VMCA-issued certificate. There are two valid reasons for refreshing your STS signing certificate or certificate chain.

- If it is close to expiry. The standard lifespan of the vCenter Server STS signing certificate is 10 years. Your vCenter Server system will notify you in advance of STS certificate expiry. An alarm is triggered once per week when your STS certificate is 90 days away from expiry, and then daily when seven days away.
- If you already replaced your signing certificate with a third-party or enterprise one and now want to revert back to a default VMCA-issued certificate. This procedure replaces the custom or third-party STS signing certificates you added.

1. Retrieve the current vCenter Server signing certificate chain by calling the `get` function of the `SigningCertificate` interface.

   ```
   get()
   ```

2. Refresh the vCenter Server signing certificate by calling the `refresh` function of the `SigningCertificate` interface.

   Pass `true` as a value to the `force` parameter to force the refresh of the signing certificates in environments that would otherwise prevent the operation from occurring, such as a mixed-version environment. If `null`, the refresh of the vCenter Server signing certificate chain is not forced.

If successful, the system returns the x509 certificate chain issued in accordance with the vCenter Server policies.

> **CAUTION**
> If you used a forced refresh, you must restart your vCenter Server and all linked services.

Set a Custom STS Signing Certificate to vCenter Server

You can import and replace the vCenter Server STS signing certificate with a custom generated or third-party certificate by using the `SigningCertificate` interface.

- Verify that the custom generated or third-party certificate chain and private key are available on your machine.
- Verify that you have the required privilege: **CertificateManagement › Administer**.

In the usual case, you must not replace the vCenter Server STS signing certificate as it is not an external-facing certificate. The STS is an internal service that enables communication between various vSphere services. A fresh installation of vSphere 7.0 and later comes with a signing certificate that is issued with a default duration of 10 years. Replace the STS signing certificate with a custom or third-party certificate only if your company security policy requires you to do so.

1. Create an object instance of type `SigningCertificateTypes.SetSpec` with the corresponding private key and certificate chain.
   a) Set the private key by calling the `setPrivateKey(privateKey)` method. Pass as value to the `privateKey` parameter the corresponding unencrypted PKCS#8 private key in base64-encoded PEM format.
   b) Set the custom generated or third-party certificate chain in base64-encoded PEM format by calling the `setSigningCertChain(X509CertChain)` method. It must be a valid certificate chain with the leaf certificate marked for Digital Signature key usage. The leaf certificate must be first in the sequence and the root must be last.
2. Set the STS signing certificate by calling `set(SigningCertificateTypes.SetSpec)` and passing the signing certificate specification.

CAUTION
The change of the STS signing certificate might leave systems in the local vCenter Server domain in a non-functional state. To prevent system failure, restart your vCenter Server instance and all linked services.

Managing vSphere Native Key Provider

You can use the vSphere Automation to perform VMware vSphere®Native Key Provider™ operations.

In vSphere 7.0 Update 2 and later, you can use the built-in vSphere Native Key Provider to enable encryption technologies, such as virtual TPMs (vTPM).

vSphere Native Key Provider is included in all vSphere editions and does not require an external key server (also called a Key Management Server (KMS) in the industry). You can also use vSphere Native Key Provider for vSphere Virtual Machine Encryption, but you must purchase the VMware vSphere®Enterprise Plus Edition™.

For details about vSphere Native Key Provider, see the *vSphere Security* documentation.

Retrieve vSphere Native Key Providers

You can use to list vSphere Native Key Providers and retrieve details about a specific vSphere Native Key Provider.

1. Retrieve a list of vSphere Native Key Providers.
2. Retrieve information about a vSphere Native Key Provider.

Manage vSphere Native Key Providers

You can use to perform vSphere Native Key Provider management operations.

You can add, update, and remove vSphere Native Key Providers.

1. Add a vSphere Native Key Provider.
2. Update a vSphere Native Key Provider.
3. Remove a vSphere Native Key Provider.

Export and Import vSphere Native Key Provider Configuration

You can use to export and import vSphere Native Key Provider configuration.

You can set a vSphere Native Key Provider as the default for a vCenter Server instance by using the `SetDefaultKmsCluster` method of the `CryptoManagerKmip` managed object from the vSphere Management SDK.

1. Export a vSphere Native Key Provider configuration.

   NOTE
   You must export the vSphere Native Key Provider configuration at least once before you can use a vSphere Native Key Provider.

2. Import a vSphere Native Key Provider configuration.

Retrieve vSphere Native Key Providers on a Host

You can use to list vSphere Native Key Providers on a host and retrieve details about a specific vSphere Native Key Provider.

1. Retrieve a list of vSphere Native Key Providers on a host.
2. Retrieve information about a vSphere Native Key Provider on a host.

vSphere Trust Authority

You can use the vSphere Automation to perform vSphere Trust Authority operations.

vSphere Trust Authority is a foundational technology that enhances workload security. vSphere Trust Authority establishes a greater level of trust in your organization by associating an ESXi host hardware root of trust to the workload itself. For details about vSphere Trust Authority, see the *vSphere Security* documentation.

Configure a vSphere Trust Authority Cluster

You can use to perform vSphere Trust Authority Cluster management operations.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can retrieve details about vSphere Trust Authority Clusters, update the state of a cluster, and check the result of the update operation.

1. Retrieve a list of clusters for a vCenter Server instance that are configured as Trust Authority Clusters.
2. Update the state of a cluster.
3. Check the result of the last update operation for the same cluster.

Configure Key Providers

You can use to perform Key Provider management operations.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can retrieve, add, update, remove, and retrieve details about Key Providers.

1. Retrieve a list of Key Providers to see which Key Providers the cluster is using.
2. Add a new Key Provider which all hosts in the cluster can use.
3. Retrieve information about a Key Provider to verify the configuration.
4. Update an existing Key Provider to modify the connection details and primary key for it.
5. Remove a Key Provider.

Establish Trust Between Key Provider and Key Server

You can use to perform trust management operations.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can list and update server certificates, retrieve, generate, and update client certificates, generate a CSR, and set the key server credential.

1. List the remote server certificates on the configured key servers to verify the trusted key servers.
2. Retrieve the list of trusted server certificates.
3. Update the trusted server certificates.

   NOTE
   This operation overwrites the existing list of trusted certificates.

4. Retrieve the existing client certificate.
   If the operation is successful, you receive the client certificate in PEM format.
5. Generate a new self-signed client certificate, used to establish a secure connection to the key server.

   NOTE
   This operation overwrites the existing client certificate.

   If the operation is successful, you can provide the newly generated self-signed client certificate to the key server to establish trust with the Key Provider.
6. Update the client certificate to specify what Key Provider should use to authenticate with the key server.

   NOTE
   If a client certificate exists, this operation overwrites it.

7. Generate a certificate signing request (CSR) for the client certificate.

   NOTE
   If a CSR exists, this operation overwrites it.

   If the operation is successful, you receive the client CSR in PEM format and the host ID which issued it. The generated CSR can later be signed by a third party. The signed CSR should be replicated and set on each host.
8. Set the key server credential for key servers that require a password.

Configure Trusted TPMs of Attested ESXi Hosts on a Cluster Level

You can use to manage remote attestation configuration for TPM trust.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can add, list, remove, and retrieve details about TPM CA certificates and TPM endorsement keys. You can also set and retrieve TPM 2.0 attestation settings.

1. Add a new TPM CA certificate to a Trusted Cluster to specify a trusted platform OEM.
2. Retrieve a list of configured TPM CA certificates on a Trusted Cluster to identify the trusted platform OEMs.
3. Remove a TPM CA certificate from a Trusted Cluster because a platform OEM is no longer trusted.
4. Retrieve details about a specific TPM CA certificate on a Trusted Cluster to get more information about the trusted platform OEM.
5. Add a new TPM endorsement key to a Trusted Cluster to specify a trusted ESXi host.
6. Retrieve a list of configured TPM endorsement keys in a Trusted Cluster to identify the trusted ESXi hosts.
7. Remove a TPM endorsement key from a Trusted Cluster because an ESXi host is no longer trusted.
8. Retrieve details about a specific TPM endorsement key on a Trusted Cluster to get more information about the trusted ESXi host.
9. Set the TPM 2.0 attestation settings by specifying that TPM endorsement keys on a Trusted Cluster do not need to be signed because the trusted OEM does not sign endorsement keys.

10. Determine the TPM 2.0 attestation settings in a Trusted Cluster.

Configure Trusted ESXi Builds on a Cluster Level

You can use to manage trusted instances of ESXi software on a cluster level.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can import, list, remove, and retrieve details about ESXi base images.

1. Import ESXi metadata as a new trusted base image to each host in a vSphere Trust Authority Cluster.
2. Retrieve a list of trusted ESXi base images in a vSphere Trust Authority Cluster.
3. Remove an ESXi base image that should no longer be trusted from a vSphere Trust Authority Cluster.
4. Retrieve details about a trusted ESXi base image version in a vSphere Trust Authority Cluster.

Retrieve vSphere Trust Authority Components Information

You can use to retrieve information about Attestation Service and Key Provider Service instances running on hosts.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can use the retrieved information to connect to the hosts running the vSphere Trust Authority components.

1. Retrieve detailed information, including the certificates, about the Attestation Service instance running on a Trust Authority Host.
2. List Trust Authority Hosts running an Attestation Service instance by using filters.
3. Retrieve detailed information, including the certificates, about the Key Provider Service instance running on a Trust Authority Host.
4. List Trust Authority Hosts running a Key Provider Service instance by using filters.

Configure vSphere Trust Authority Components

You can use to perform Key Provider Service and Attestation Service management operations.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can register, list, remove, and retrieve details about Key Provider Service and Attestation Service instances.

1. Register a Key Provider Service instance in a Workload vCenter Server.
   The Key Provider Service instance is propagated to all Workload ESXi hosts that the Workload vCenter Server manages.
2. Register an Attestation Service instance in a Workload vCenter Server.
   The Attestation Service instance is propagated to all Workload ESXi hosts that the Workload vCenter Server manages.
3. List Key Provider Service instances registered in a Workload vCenter Server by using filters.
4. List Attestation Service instances registered in a Workload vCenter Server by using filters.
5. Remove a registered Key Provider Service instance.
   The Workload ESXi hosts can no longer retrieve keys by using that Key Provider Service instance.
6. Remove a registered Attestation Service instance.
   The Workload ESXi hosts can no longer attest that their configuration is secure by using that Attestation Service instance.
7. Retrieve detailed information, including the certificates, for a registered Key Provider Service instance.

8. Retrieve detailed information, including the certificates, for a registered Attestation Service instance.

Configure vSphere Trust Authority Components for Trusted Clusters

You can use to manage Key Provider Service and Attestation Service instances that a Trusted Cluster is configured to use.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can configure, list, remove, and retrieve details about Key Provider Service and Attestation Service instances.

1. Configure a cluster in a Workload vCenter Server to use a registered Key Provider Service instance.
   You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

   If the operation is successful, the Key Provider Service instance is propagated to all Trusted ESXi hosts in the cluster.
2. Configure a cluster in a Workload vCenter Server to use a registered Attestation Service instance.
   You receive the task ID in the response body. You can use the task ID to check the status of the task by running the following HTTP request.

   If the operation is successful, the Attestation Service instance is propagated to all Trusted ESXi hosts in the cluster.
3. List Key Provider Service instances used by a cluster by using filters.
4. List Attestation Service instances used by a cluster by using filters.
5. Remove a Key Provider Service instance from the configuration of a Trusted Cluster.
   If the operation is successful, the Trusted ESXi hosts can no longer retrieve keys by using that Key Provider Service instance.
6. Remove a registered Attestation Service instance from the configuration of a Trusted Cluster.
   If the operation is successful, the Trusted ESXi hosts can no longer attest that their configuration is secure by using that Attestation Service instance.
7. Retrieve detailed information, including the certificates, for a configured Key Provider Service instance used by a Trusted Cluster.
8. Retrieve detailed information, including the certificates, for a registered Attestation Service instance used by a Trusted Cluster.

Establish Trust Between Hosts in a vSphere Trust Authority Cluster and a Workload vCenter Server

You can use to perform trust management operations.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can establish and remove trust between a Workload vCenter Server and the hosts in a vSphere Trust Authority Cluster. You can also list all Workload vCenter Server instances that have established trust with the host in a vSphere Trust Authority Cluster.

1. Establish trust between a vSphere Trust Authority Cluster and a Workload vCenter Server by creating a profile, so that the Workload vCenter Server can retrieve the health status of the vSphere Trust Authority components.
2. Remove the trust between a Workload vCenter Server and the hosts in the vSphere Trust Authority Cluster, so that the Workload vCenter Server stops using the hosts for attestation.
3. List all profiles which the vSphere Trust Authority Cluster trusts.

Check Trusted Cluster Health

You can use to retrieve information about the health of the applied vSphere Trust Authority component configurations in a Trusted Cluster.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can retrieve basic and detailed information about the health of Key Provider Service or Attestation Service configurations applied to a Trusted Cluster with respect to the desired state. You can also retrieve detailed information about the health of all applied vSphere Trust Authority component configurations in a Trusted Cluster.

1. Retrieve a summary about the health status of all Key Provider Service instances configured for use in a Trusted Cluster.
   If the operation is successful, you can verify whether all Key Provider Service configurations of the Trusted Cluster are applied successfully and every host in the cluster is consistent with the desired state.
2. Retrieve detailed information about the health status of a specific Key Provider Service instance configured for use in a Trusted Cluster.
   If the operation is successful, you can verify whether the specified Key Provider Service configuration of the Trusted Cluster is applied successfully and every host in the cluster is consistent with the desired state.
3. Retrieve a summary about the health status of all Attestation Service instances configured for use in a Trusted Cluster.
   If the operation is successful, you can verify whether all Attestation Service configurations of the Trusted Cluster are applied successfully and every host in the cluster is consistent with the desired state.
4. Retrieve detailed information about the health status of a specific Attestation Service instance configured for use in a Trusted Cluster.
   If the operation is successful, you can verify whether the specified Attestation Service configuration of the Trusted Cluster is applied successfully and every host in the cluster is consistent with the desired state.
5. Retrieve detailed information about the health status of all vSphere Trust Authority components configured for use in a Trusted Cluster.
   If the operation is successful, you can verify whether the vSphere Trust Authority component configuration is applied successfully and every host in the cluster is consistent with the desired state.

If there are errors, you can try to remediate the Trusted Cluster. See Remediate a Trusted Cluster.

Remediate a Trusted Cluster

You can use to remediate vSphere Trust Authority component configurations in a Trusted Cluster or remove the configurations.

- Verify that you have access to a working vSphere Trust Authority environment.
- Verify that you have Trusted Infrastructure administrative privileges.

You can update the applied Key Provider Service or Attestation Service configurations in a Trusted Cluster to become consistent with the desired state or you can remove the applied Key Provider Service or Attestation Service configurations. You can also update all applied vSphere Trust Authority component configurations in a Trusted Cluster or remove the configurations. By removing the configurations, you can move hosts from a Trusted Cluster to another cluster.

1. Remediate all Key Provider Service instances configured for use in a Trusted Cluster.
   If the operation is successful, the Key Provider Service configuration of every host in the cluster is consistent with the desired state.
2. Remove all Key Provider Service configurations from a Trusted Cluster.
   If the operation is successful, the applied Key Provider Service configurations are removed from the configuration of every host in the cluster without affecting the desired state.
3. Remediate all Attestation Service instances configured for use in a Trusted Cluster.

If the operation is successful, the Attestation Service configuration of every host in the cluster is consistent with the desired state.

4. Remove all Attestation Service configurations from a Trusted Cluster.
   If the operation is successful, the applied Attestation Service configurations are removed from the configuration of every host in the cluster without affecting the desired state.
5. Remediate all vSphere Trust Authority components configured for use in a Trusted Cluster.
   If the operation is successful, the vSphere Trust Authority component configuration of every host in the cluster is consistent with the desired state.
6. Remove all vSphere Trust Authority component configurations from a Trusted Cluster.
   If the operation is successful, the applied vSphere Trust Authority component configurations are removed from the configuration of every host in the cluster without affecting the desired state.

You can recheck the Trusted Cluster health after the remediation. See Check Trusted Cluster Health.

Managing Host Hardware Security

You can use the vSphere Automation to perform host hardware management operations.

You can retrieve host hardware TPM information and manage host hardware TPM endorsement keys. You can also register SGX hosts.

Retrieve Host Hardware TPM Information

You can use to retrieve a list of configured TPM devices on a host and information about each TPM device.

1. List configured TPM devices on a host.
2. Retrieve detailed information about a specific TPM device, including the manufacturer, model, and firmware version.

Manage Host Hardware TPM Endorsement Keys

You can use to retrieve a list of configured TPM endorsement keys on a host and information about each endorsement key. You can also retrieve the TPM event log and unseal a secret that is bound to an endorsement key.

You can retrieve the TPM event log for different purposes, such as configuring firmware trust with an attestation service or validating the boot time TPM measurements. You can unseal a secret that is bound to an endorsement key to verify reported measurements. For example, you can verify measurements from the TPM event log.

1. List configured TPM endorsement keys on a host.
2. Retrieve detailed information about a specific TPM endorsement key.
3. Retrieve the event log associated with a TPM device.
4. Unseal a secret that is bound to an endorsement key.

Register an SGX Host

You can use to register hosts by stored Intel Software Guard Extensions (SGX) configuration.

1. Register a host by stored SGX configuration.

vCenter Server Management

You can use the API to configure, monitor, maintain, and update vCenter Server.

Authorization Model for Administration of vCenter Server

There are three types of authorization levels in vCenter Server.

Table 75: Authorization Levels

Authorization Level	Description
operator	A user has read access to configuration settings.
administrator	A user has read and write access to configuration settings, but cannot manage user accounts.
super administrator	A user has all the capabilities of the other roles, and has the additional capabilities of creating local user accounts and accessing the local Bash shell.

This model applies to the API and all other interfaces to vCenter Server except when you use SSH and log in by using a local account.

Authorization Model Mapping to the vCenter Single Sign-On Domain

The three-level authorization model of vCenter Server maps to local roles and to vCenter Single Sign-On groups, depending on how the user authenticated. This model allows consistent security control regardless of operational context.

The authorization levels map to group and role.

Table 76: Authorization Mapping

Authorization Level	vCenter Single Sign-On Group	vCenter Server Local Role
operator	SystemConfiguration.Administrators	operator
administrator	SystemConfiguration.Administrators	admin
superAdministrator	SystemConfiguration.BashShellAdministrators	superAdmin

When a super administrator adds user accounts, the options available include a choice of the role to assign to the new user.

Using the Operator Role

The **operator** role is the most restricted of the authorization levels available to users who work with vCenter Server.

Operators are allowed to view information about vCenter Server. They are not allowed to alter its configuration. The **operator** role is suited for monitoring and reporting functions. For example, the **operator** role provides access to the following methods.

- `resources.system.health.get`
- `resources.storage.stats.list`
- `services.status.get`

Using the Admin Role

The **administrator** role provides an intermediate authorization level for users who manage vCenter Server.

An **administrator** role is required for users who alter the vCenter Server configuration, exercise control functions, or other operations that can affect regular users.

For example, the **administrator** role provides access the following methods.

- `networking.ip4v.renew`
- `networking.firewall.addr.inbound.add`
- `services.control`
- `shutdown.reboot`

Using the SuperAdmin Role

The **superAdmin** role is the most expansive authorization level for users who manage vCenter Server.

The **superAdmin** role allows unrestricted access to vCenter Server. This role is required for adding or altering user accounts and for using the Bash shell.

Performing Privilege Checks Operations

Privilege checks recorder is a feature which allows you to monitor and subsequently query the privileges that were checked. You can use the recordings to create scripts that automatically create roles with minimum required privileges to run specific workflows of operations.

Currently, it is very hard to find out the minimal set of privileges that are required to run a specific workflow of operations. The `PrivilegeChecks` provides for retrieving a list of the latest privilege checks along with the corresponding sessions, users, managed objects, and operation IDs (`opIDs`).

The following table lists the available privilege checks operations.

Operation	Description
List privilege checks	Returns a list of privilege checks that match criteria which you specified.
Get latest privilege check	Returns a marker to the last recorded privilege check. This allows retrieving all privilege checks up to or after a specified moment in time.

Filtering Privilege Checks

When you retrieve lists of privilege checks, you can refine your search by using the . The following table lists the available filtering options.

Filtering Option	Description
`Objects`	IDs of the managed objects on which the privilege check was performed. If , all objects match.
`OpIDs`	`OpIDs` of the requests for which the check was performed. If , all `opIDs` match.
`Principals`	Principals for which the privilege check was performed. The value matches privilege checks for anonymous sessions. If , all principals match.
`Privileges`	Privileges that were checked. If , all privileges match.
`Sessions`	Sessions for which the check was performed. If , all sessions match.

Using Advanced Settings

You can configure advanced settings by using the vSphere Client. Some of the advanced settings are not available in the API, but are required for the privilege checks recorder to function properly. For information about configuring advanced settings, see *vCenter Server Configuration*.

Advanced Setting Name	Description
config.vpxd.privCheck.pageSize	Specifies the default page size for privilege checks lists.
config.vpxd.privCheck.bufferSize	Specifies the count of privileges to be kept in memory. The default value is 0. If you do not change the default value, the privilege checks recorder does not record any data.
config.vpxd.privCheck.cleanupInterval	Specifies the interval on which privilege checks for unused sessions are cleaned up. The default value is 30 minutes.

vCenter Server Installation and Setup

You can use the API to perform operations related to stage 2 of the installation process. You can also perform backup, restore, and troubleshooting operations.

Install Stage 2

The vCenter Server API provides methods for performing stage 2 deployment operations on a newly installed vCenter Server instance.

The vCenter Server instance is deployed in two stages. With stage 1 of the deployment process, you deploy the OVA file, which is included in the installer. With stage 2 of the deployment process, you set up and start the services of the newly deployed vCenter Server instance.

To complete stage 1 of the deployment process, you can use the GUI installer or perform a CLI deployment. For details, see *vCenter Server Installation and Setup*. Alternatively, you can perform a deployment by using the VMware OVF Tool. For details, see *OVF Tool User's Guide*.

Setting Up a Newly Installed vCenter Server Instance

You can use the API to set up a newly deployed vCenter Server instance.

After stage 1 of the deployment process completes successfully, the vCenter Server instance enters in an `INITIALIZED` state. If the instance is not initialized, you cannot run stage 2 of the deployment process. You can get the state of the vCenter Server instance by using the `vcenter deployment` service. The vCenter Server instance can enter six states during the deployment process.

Figure 109: Install Stage 2 State Diagram

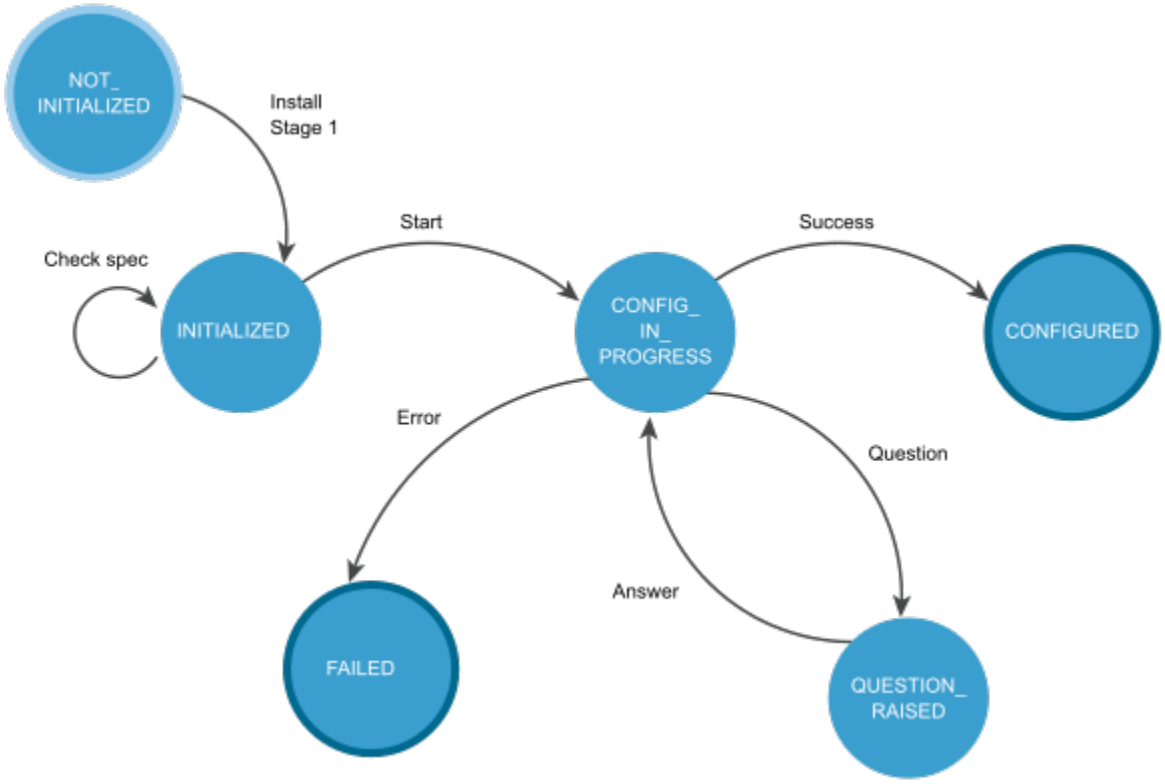


Table 77: vCenter Server Instance States During Install Stage 2

State	Description
NOT_INITIALIZED	The install stage 1 phase is in progress, not started, or failed.
INITIALIZED	The vCenter Server instance is deployed and ready for setup.
CONFIG_IN_PROGRESS	The setup process is in progress.
QUESTION_RAISED	You must answer the question to continue the setup process. The vCenter Server instance stays in the QUESTION_RAISED state until it receives the correct answer.
FAILED	Errors occurred during the setup process. You can check the errors, warnings, and info data structures.
CONFIGURED	The vCenter Server instance is installed and configured successfully.

FAILED and CONFIGURED are final states.

Table : User Operations lists operations that you can perform to set up your newly deployed vCenter Server instance.

Table 78: User Operations

Operation	Description
Get deployment information	You can retrieve information about the current deployment status. This operation is useful both before initiating stage 2 of the deployment and for monitoring the progress of the setup process.
Validate the configuration document	You can optionally verify whether your install spec is valid before starting the setup process.
Configure the vCenter Server instance	You can initiate the setup process by providing an install spec that defines the values for the settings that you want to configure.
Get question	You can retrieve a question raised during the setup process.
Answer question	You can provide an answer to the question raised during the setup process. The available answer values are `YES`, `NO`, `OK`, `CANCEL`, `ABORT`, `RETRY`, and `IGNORE`. The possible answer values depend on the type of the question. **NOTE** Each question has a default answer value. If you set questions to receive automatic answers in the install spec and a question is raised during the setup process, the default answer value is automatically provided as the answer to the question.

Workflows for Install Stage 2

You can use the `vcenter deployment` API to run the install stage 2 process of your vCenter Server instance.

Figure: Install Workflow and Figure: Install Stage 2 Workflow show example install workflows.

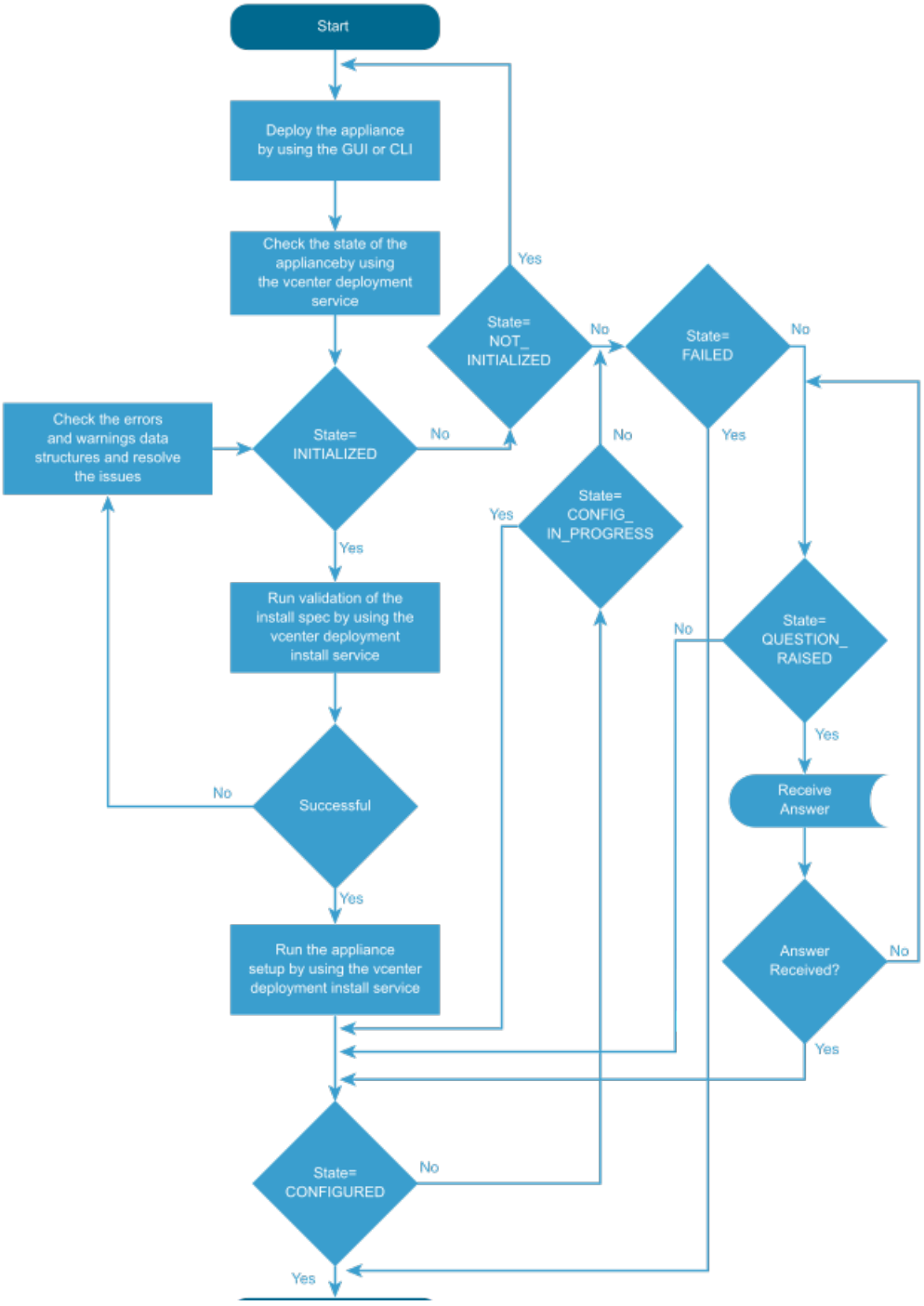
During stage 1, the vCenter Server instance is in a `NOT_INITIALIZED` state. After a successful deployment, the vCenter Server instance enters in an `INITIALIZED` state. If there are errors during stage 1, the vCenter Server instance stays in a `NOT_INITIALIZED` state and you must redeploy it.

You can check the state of the vCenter Server instance before, during, and after the setup process. You can run the install stage 2 process if the vCenter Server instance is initialized. You can check the setup configuration before you initiate stage 2 by running pre-checks. If errors or warnings appear during the validation of the install specification, you must remove the causes and correct the specification.

During the setup process, the regular vCenter Server instance state is `CONFIG_IN_PROGRESS`. The vCenter Server instance can also enter in a `FAILED` or `QUESTION_RASED` state. If a question appears during the setup, the vCenter Server instance enters in a `QUESTION_RAISED` state and stays in it until you provide an answer. You can set questions to receive automatic answers in the install spec and if a question is raised during the setup process, the default answer value is automatically provided as the answer to the question.

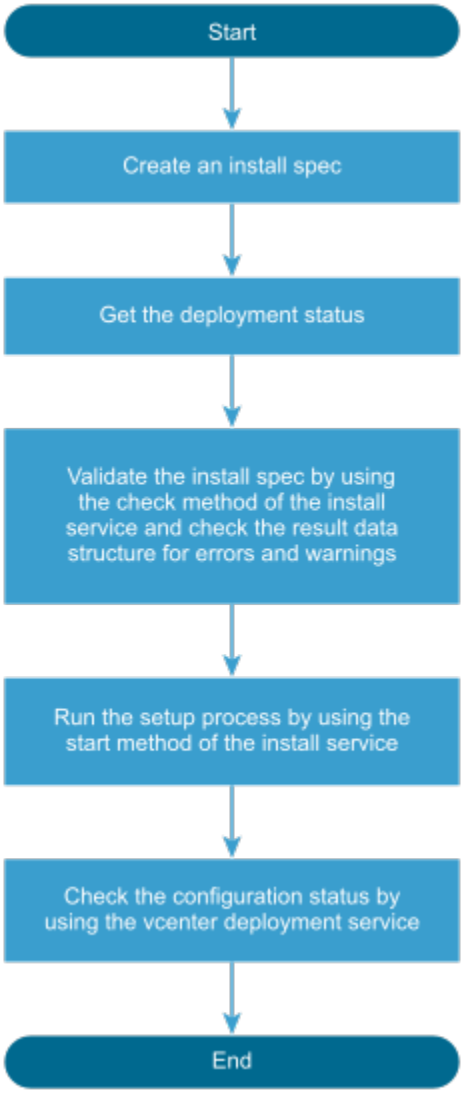
If errors occur during the setup process, the vCenter Server instance enters in a `FAILED` state and you must restart the setup after the causes are removed. If the setup is successful, the vCenter Server instance enters in a `CONFIGURED` state.

Figure 110: Install Workflow



For information about the states of the vCenter Server instance and available operations, see Setting Up a Newly Installed vCenter Server Instance.

Figure 111: Install Stage 2 Workflow



You can run the setup pre-checks and the install stage 2 process by creating and passing an `InstallSpec`. In `InstallSpec`, you define the setup configuration. See Figure: Install Stage 2 Workflow . You can run the setup in silent mode by setting the `InstallSpec.auto_answer` to `true`. The default value of `InstallSpec.auto_answer` is `false` and the setup is in interactive mode, in which you must provide answers to the raised questions.

For information about the classes, variables, and default values, see the *`API reference`* documentation.

File-Based Backup and Restore of vCenter Server

You can back up a vCenter Server instance and later restore the instance from the backup copy.

Backing up vCenter Server

The vCenter ServerKey definition for "mgmt_api" not found in the DITA map. supports backing up key parts of the vCenter Server instance. This allows you to protect vCenter Server data and to minimize the time required to restore data center operations.

The backup process collects key files into a tar bundle and compresses the bundle to reduce network load. To minimize storage impact, the transmission is streamed without caching in the vCenter Server instance. To reduce total time required to complete the backup operation, the backup process handles the different components in parallel.

You have the option to encrypt the compressed file before transmission to the backup storage location. When you choose encryption, you must supply a password which can be used to decrypt the file during restoration.

The backup operation always includes the vCenter Server database and system configuration files, so that a restore operation has all the data needed to re-create an operational vCenter Server instance. Current Alarms are included as well. You also have the option to specify additional data sets, called parts. In this release, you can specify a data set that includes Statistics, Events, and Tasks.

Backup and Restore Protocols for vCenter Server

The vCenter Server backup and restore feature supports a number of plug-in communication protocols.

Choose one of these protocols as the backup location type when you invoke the operation.

- FTP
- FTPS
- SCP
- HTTP
- HTTPS
- NFS
- SMB

The value `PATH` for the location type field indicates a locally mounted volume.

> **NOTE**
> If you specify the `SCP` protocol, you must specify an absolute path as the value of the location type field when you create the backup job.

Calculate the Size Needed To Store the Backup File

When you prepare to do a backup of avCenter Server instance, you can use the API to calculate the storage space needed for the backup file.

- Verify that you have avCenter Server instance running.
- Verify that you are familiar with authentication methods. See Authentication Mechanisms.

You can do this task when you are choosing a backup storage location or whenever your existing storage location might be approaching full capacity.

1. Authenticate to thevSphere Automation API endpoint and establish a session.
2. Request a list of backup parts available.
3. For each available backup part, request the size of the backup file.
   The backup process calculates the compressed size of each backup part.
4. Choose which parts to include in the backup, and sum their sizes.

   The backup storage server must have sufficient space to contain the chosen parts.

After you choose which backup parts you will store, and verify that the backup storage server has sufficient free space, you can launch a backup job. For information, see Back up a vCenter Server Instance by Using the API.

Python

This example shows how to use Python to collect the information you need to calculate the size needed to store a backup image of thevCenter Server instance.

```
from com.vmware.appliance.recovery.backup_client import Parts


# This example assumes you have previously created a session

# and stored the session ID in my_stub_config.


# Issue a request to list the backup image parts.

Parts_stub = Parts( my_stub_config )

parts = Parts_stub.list()


# Extract IDs of backup image parts.

sizes = {}

total = 0

for part in parts :

   size = Parts_stub.get( part.id )

   sizes[part.id] = size

   total += size


# Show the result.

print( 'Backup image parts:' )

for part_id in sizes.keys() :

   print( '  part {0} = {1}KB'.format( part_id, sizes[part_id] ) )

   print( 'Total size: {0}KB'.format( total ) )
```

Back up a vCenter Server Instance by Using the API

You can use the Key definition for "mgmt_api" not found in the DITA map. to create a backup of the vCenter Server database and key components of the vCenter Server instance.

- Verify that the vCenter Server instance is in a ready state. All processes with start-up type automatic must be running.
- Verify that no other backup or restore jobs are running.
- Verify that the destination storage location is accessible to the backup process.
- Verify that the path to the destination directory exists, as far as the parent directory.
- If the destination directory does not exist, the backup process creates it. If the directory does exist, verify that it is empty.

- Verify that the destination storage device has sufficient space for the backup file. For information about how to calculate the space needed for the backup file, see Calculate the Size Needed To Store the Backup File.

This procedure explains the sequence of operations you use to create a backup file of the vCenter Server instance. You can do this as part of a regular maintenance schedule.

1. Authenticate to the vSphere Automation API endpoint and establish a session.
2. to describe the backup operation.

   The specifies several , especially the backup location, the protocol used to communicate with the storage server, the necessary authorization, and which optional parts of the database you want to back up. The core inventory data and Alarms are always backed up, but you can choose whether or not to back up Statistics, Events, and Tasks. Collectively, this optional part of the backup is referred to as `seat`.

3. Issue a request to start the backup operation.
4. From the response, save the unique job identifier of the backup operation.
5. Monitor the progress of the job until it is complete.
6. Report job completion.

The backup file of the vCenter Server instance is created.

Schedule a Backup Job

You can automate the backup process by creating a schedule that runs backup jobs at specific times.

- Verify that you can access the backup server and you have read and write permissions.
- Verify that you have established a connection to the vAPI services.

You can keep existing backups on the backup server. The retention policy defines the maximum number of backups that the server keeps. You can also specify whether the backup job should run once, or on a recurring basis. The recurrence policy defines the days of the week and specific times at which the backup job is scheduled to run.

1. Create a .
2. Specify the retention and recurrence information.
3. Create a schedule by specifying the backup location, user credentials to access the location, retention, and recurrence information.
4. and pass the updated information.
5. Get a backup schedule by passing a schedule ID.

Run the backup job by using the schedule.

Restoring vCenter Server

The vCenter ServerKey definition for "mgmt_api" not found in the DITA map. supports restoring a vCenter Server instance from a backup copy. The API simplifies the process by unifying the handling of various components of vCenter Server in a single operation.

The process of restoring a vCenter Server instance from a backup has two phases.

1. Deploy a new vCenter Server instance. OVF deployment is described in the *vSphere Automation SDKs Programming Guide*.

2. Invoke the `restore` operation from the Key definition for "mgmt_api" not found in the DITA map. to apply configuration settings and load the vCenter Server database from the backup file.

   NOTE
   You cannot specify optional parts for the restore operation. The restore operation includes all optional parts, such as Events and Tasks, that were specified at the time when the backup file was created.

Authentication When Restoring a vCenter Server Instance

During the process of restoring a vCenter Server instance from a backup image, you cannot use token-based authentication. You must use basic authentication until the vCenter Server instance is fully configured.

When you restore your vCenter Server instance from a backup file, it begins in an unconfigured state. During this time, you must use basic authentication to access the Key definition for "mgmt_api" not found in the DITA map.. When you use basic authentication, do not use the vSphere Automation API endpoint. Instead, you must connect your client to port 5480 of the vCenter Server instance.

When you use basic authentication, you must pass user name and password credentials with each . Use credentials that are known to the guest operating system of the vCenter Server instance.

Availability of Services While Restoring a vCenter Server Instance

During the process of restoring the vCenter Server backup file, services in the vCenter Server instance must restart. While they are restarting, your API client receives an error message.

You can write your client to trap the error, but you have no way to know when the vCenter Server services are running again. To determine when the restore process is complete, you must retry the API connection until it succeeds, then request the status of the job.

Restore a vCenter Server Instance by Using the API

You can use the Key definition for "mgmt_api" not found in the DITA map. of to restore a vCenter Server instance from a backup file containing the vCenter Server database and key components of the vCenter Server instance.

- Verify that the backed up vCenter Server instance is powered off.
- A new vCenter Server instance must be deployed in an unconfigured state, except that it must have a fully qualified domain name or IP address that matches the old one.
- Verify that the new vCenter Server instance has the same build number as the one in the backup file.
- Verify that the new vCenter Server instance has a size equal to or greater than the old one. If the old vCenter Server instance was customized to exceed the largest template size, the new one must be customized to the same size.
- Verify that no other backup or restore jobs are running.
- Verify that the destination storage location is accessible to the vCenter Server restore process.

1. to describe the restore operation.
2. Issue a request to start the restore operation.
3. Monitor the progress of the job until it is complete.
4. Report job completion.

After the vCenter Server instance is fully configured by the restore operation, you can resume using the vSphere Automation API endpoint for subsequent operations.

Reconcile a vCenter Server Instance with Nodes in Embedded Linked Mode

You can run the reconciliation process after you successfully restored your vCenter Server instance. By using the API or HTTP requests, you can reconcile vCenter Server nodes that work in an embedded linked mode and are connected in a ring or daisy-chain.

- Verify that you successfully restored your node from an image.
- Verify that the replication partners are available.
- Verify that you restored your nodes in the correct order, if you use a daisy-chain topology.
- Verify that you have administrator's credentials to your Single Sign-On domain.
- Verify that there is no running or failed reconciliation job.

Reconciliation is a post-restore process that checks whether the vCenter Server partners in embedded linked mode are available, synchronizes the vCenter Server data and services with the partners, and runs the vCenter Server services. The processes of restore and reconciliation depend on the topology and if there are changes in the topology between the backup and restore, you cannot restore the embedded linked mode. If the replication partners are not available and you try to restore the first node, you must ignore the warnings. In this case, any changes that are made in the topology or infrastructure after the backup will be lost. If you restore a node different from the first one, you must add it to the domain of the first node. If you use a daisy-chain topology, you must first restore the first node, and after that to restore the second, link it to the first one, and apply the same to the following nodes.
You can use the reconciliation API after a file-based and an image-based restore. After an image-based restore, you can run the reconciliation process by using the API or UI. After a file-based restore, you can monitor the reconciliation process by using the `GET https://<vcenter_ip_address_or_fqdn>/api/appliance/recovery/reconciliation/job` HTTP request. For information about how to restore a vCenter Server instance from an image or a file by using the UI, see the *vCenter Server Installation and Setup* documentation.

1. , specify user name and password of Single Sign-On administrator, and set the `ignore_warnings` field to `true`.

   The default value of `ignore_warnings` is `false`. If you do not set `ignore_warnings` to `true`, the reconciliation fails due to the validation warnings.

2. Run a reconciliation job by using the .

   You can check the result of the operation by reading the . contains information about the job such as description, status, progress, error, start and end time.

3. Get the status of the job by .

   The possible states are `NONE`, `RUNNING`, `FAILED`, and `SUCCEEDED`.

Troubleshooting for vCenter Server Installation or Deployment

You can use the API to perform troubleshooting operations related to the installation and deployment of vCenter Server.

Managing System Logs

You can automate the forwarding of vCenter Server system log messages to remote logging servers by using the vCenter Server Management API.

You can configure the syslog forwarding by using the API or user interface. For information about how to manage the syslog by using the user interface, see the *vSphere Monitoring and Performance* documentation.

Managing Support Bundles

You can retrieve information about support bundles and create support bundles.

The following table lists the operations that you can perform to manage support bundles.

Table 79: User Operations

Operation	Description
Get support bundle components	You can retrieve a list of components and manifests included in the support bundle. Manifests specify the files that must be collected for a component as part of the support bundle.
List support bundles	You can retrieve a list of generated support bundles. The list contains details about each support bundle.
Create a support bundle	You can generate a support bundle. Optionally, you can specify the support bundle components and the partition where you want to save the support bundle.

You can run support bundle management operations by using the vSphere Automation SDK or sending an HTTP request.

REST

You can use HTTP requests to perform support bundle management operations.

The following HTTP requests show the syntax that you can use to perform the available user operations.

- Get support bundle components

  ```
  GET https://<vcenter_ip_address_or_fqdn>:443/api/appliance/support-bundle/
  components
  ```

- List support bundles

  ```
  GET https://<vcenter_ip_address_or_fqdn>:443/api/appliance/support-bundle
  ```

- Create a support bundle

  ```
  POST https://<vcenter_ip_address_or_fqdn>:443/api/appliance/support-bundle
  ```

For information about the content and syntax of the HTTP request body for each operation, see the *API reference* documentation.

The following examples provide sample input and output values for the support bundle management operations.

- *Example: Get Support Bundle Components*
  This operation does not require input.

  ```
  GET https://<vcenter_ip_address_or_fqdn>:443/api/appliance/support-bundle/
  components
  ```

Type	Example

Output	{
	"Group1" : List of manifests under this group,
	"Group2": ...
	}

- *Example: List Support Bundles*
  This operation does not require input, but supports optional query parameters.

  ```
  GET https://<vcenter_ip_address_or_fqdn>:443/api/appliance/support-bundle
  ```

Type	Example

Input	"IterationSpec":{
	"size" : 10
	}
	FilterSpec:{
	"creationStatus" : "SUCCEEDED",
	"available" : True/False
	}

Output	{
	"Description": "Reason for generation",
	"Generation time": 2 Mar 2021,
	"URL" : "Download URL",
	"Downloadable" : True/False
	"Expiration time": "URL expiration time"
	}

- *Example: Create a Support Bundle*
  This operation requires description input and supports optional body parameters. As output, you receive a task ID, which you can use to track the progress of the task. When the operation is completed, you receive the download URL in the task response.

  `POST https://<vcenter_ip_address_or_fqdn>:443/api/appliance/support-bundle`

Type	Example

Input	{
	"Description" : "Reason",
	"Inclusions" : List of components to be included in the bundle
	<Optional> path: partition to be used.
	}

Output	Task ID

vCenter Server Upgrade

You can use the API to perform operations related to stage 2 of the upgrade process. You can also perform historical data transfer operations.

Upgrade Stage 2

You can upgrade your vCenter Server instance by using the API, CLI, or GUI.

For information about how to upgrade the vCenter Server instance by using CLI and GUI, see the *vCenter Server Upgrade* documentation.

Upgrading a vCenter Server Instance

You can use the API during stage 2 of the vCenter Server instance upgrade.

By using the API, you can upgrade your vCenter Server instance. For information about the upgrade process, its stages, supported configurations, upgrade paths, prerequisites for upgrading, and the sequence for upgrading a vSphere environment, see the *vCenter Server Upgrade* documentation.

After you deploy the vCenter Server instance on stage 1 by using the GUI or CLI, the instance enters in an `INITIALIZED` state. If the vCenter Server instance is not initialized, you cannot run stage 2 of the upgrade process. You can get the state of the vCenter Server instance by using the `vcenter deployment` service. There are six states during the upgrade process.

Figure 112: Upgrade Stage 2 State Diagram

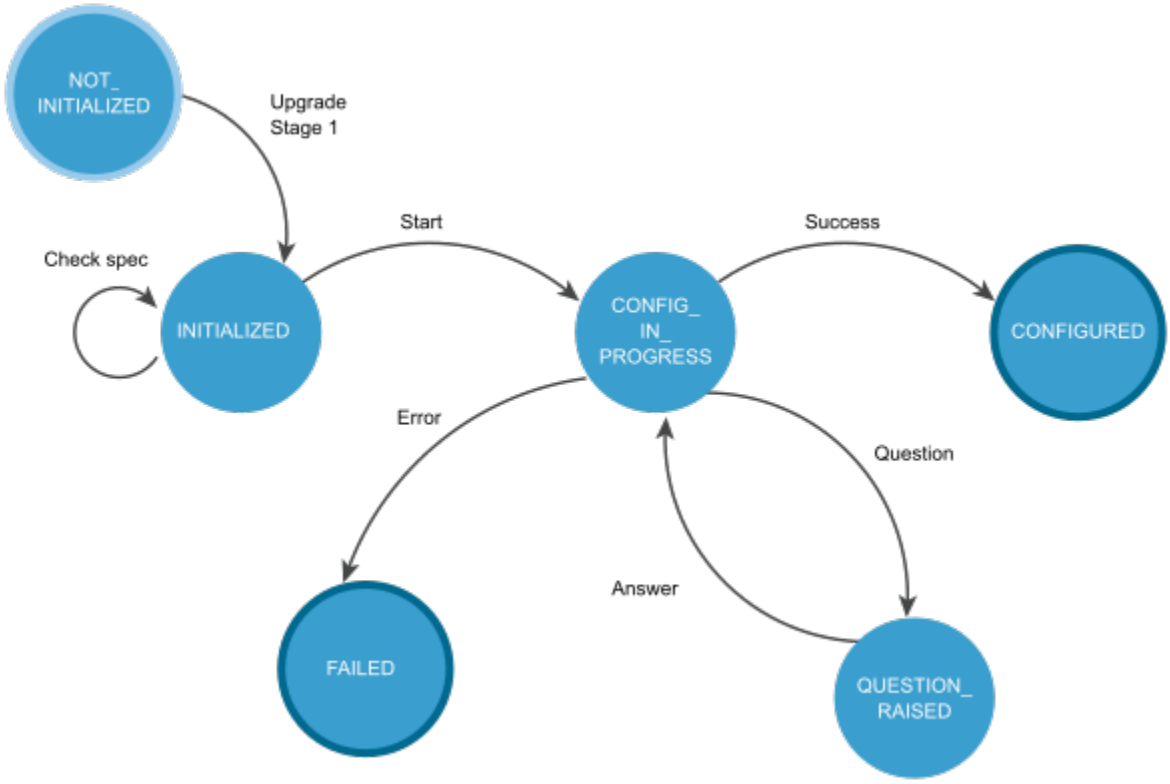


Table 80: vCenter Server Instance States During Upgrade Stage 2

State	Description
`NOT_INITIALIZED`	The upgrade stage 1 phase is in progress, not started, or failed.

Table continued on next page

Continued from previous page

State	Description
INITIALIZED	The vCenter Server instance is deployed and ready for upgrading.
CONFIG_IN_PROGRESS	The upgrade process is in progress.
QUESTION_RAISED	You must answer the question to continue the upgrade process. The vCenter Server instance stays in the QUESTION_RAISED state until it receives the correct answer.
FAILED	Errors appeared during the upgrade process. You can check the errors, warnings, and info data structures.
CONFIGURED	The vCenter Server instance is upgraded or configured successfully.

FAILED and CONFIGURED are final states.

You can roll back a vCenter Server instance upgrade by using the GUI. For information about how to roll back a vCenter Server instance, see the *vCenter Server Upgrade* documentation.

After the upgrade, you can check the vCenter Server instance type, domain registration, services, their state and health status by using the API. For information about how to verify whether the upgrade of your vCenter Server instance is successful, see the *vCenter Server Upgrade* documentation.

Table : User Operations shows operations that you can perform to upgrade your vCenter Server instance.

Table 81: User Operations

Operation	Description
Operations for upgrading	
Get the state of the vCenter Server instance	You can get the state of the vCenter Serverinstance before, during and after the upgrade process.
Check	You can validate the upgrade spec before you run the upgrade process. If the vCenter Server instance is in the INITIALIZED state, you can run the validation. The operation runs upgrade pre-checks. You can check the errors, warnings, and status data structures before you run the upgrade process.
Start	If the vCenter Server instance is in an INITIALIZED state, you can run the upgrade process. If errors appear during the upgrade, you can download the vCenter Server support bundle.
Get	If the vCenter Server instance is in a CONFIGURED state, you can get the spec that is used for upgrading.
Operations for getting and answering a question	
Get	You can get the raised question. If you set the Upgrade.auto_answer to true, the upgrade process will be in a silent mode and the vCenter Serverinstance does not generate questions. It uses default answers and you should not provide an answer.

Table continued on next page

Continued from previous page

Operation	Description
Answer	You can provide an answer to the raised question. The available answers for the upgrading are `OK`, `CANCEL`, `YES`, `NO`, `ABORT`, `RETRY`, and `IGNORE`. The answer depends on the type of the question. If you set the `Upgrade.auto_answer` to `true`, the upgrade process will be in a `silent` mode and the vCenter Serverinstance does not generate questions. It uses default answers and you should not provide an answer.

For information about the available operations in the API, see the `vcenter deployment`, `vcenter deployment upgrade`, `vcenter services`, and `vcenter system-config deployment type` services in the *`API reference`* documentation.

Workflows for Upgrade Stage 2

You can use the `vcenter deployment` API to run the upgrade stage 2 process of your vCenter Server instance.

Figure: Upgrade Workflow and Figure: Upgrade Stage 2 Workflow show example upgrade workflows.

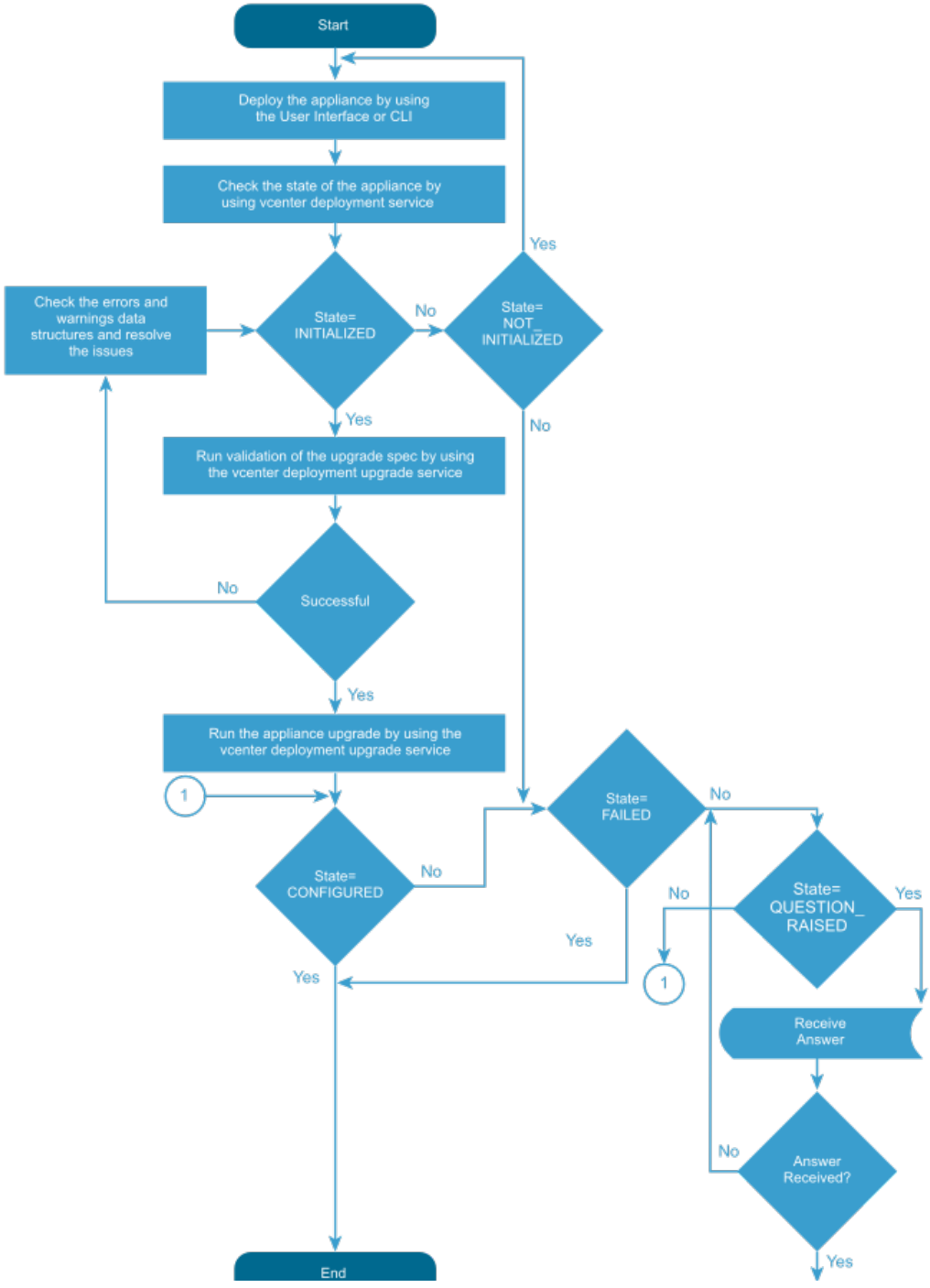
During stage 1, the vCenter Server instance is in a `NOT_INITIALIZED` state. After a successful deployment, the vCenter Server instance enters in an `INITIALIZED` state. If there are errors during stage 1, the vCenter Server instance stays in a `NOT_INITIALIZED` state and you must redeploy it.

You can check the state of the vCenter Server instance before, during, and after the upgrade process. You can run the upgrade stage 2 process if the vCenter Server instance is initialized. You can check the upgrade configuration before you run the upgrade by running pre-checks. If errors or warnings appear during the validation of the upgrade specification, you must remove the causes and correct the specification.

During the upgrade process, the vCenter Server instance can enter in a `FAILED` or `QUESTION_RASED` state. If a question appears during the upgrade, the vCenter Server instance enters in a `QUESTION_RAISED` state and stays in it until you provide an answer. You can run the upgrade in silent mode, in which the vCenter Server instance does not generate questions, and uses default answers.

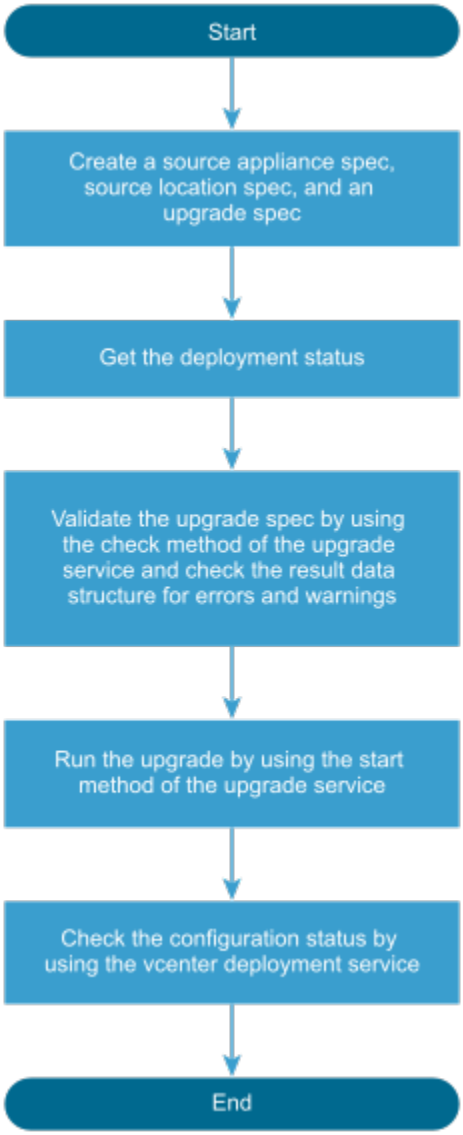
If errors appear during the upgrade, the vCenter Server instance enters in a `FAILED` state and you must remove the causes, redeploy the instance and restart the upgrade. If stage 2 of the upgrade process is successful, the vCenter Server instance enters in a `CONFIGURED` state. If the vCenter Server instance is configured, you can check its services and pause the historical data transfer.

Figure 113: Upgrade Workflow



For information about the states of the vCenter Server instance and available operations, see Upgrading a Instance.

Figure 114: Upgrade Stage 2 Workflow



You can run the upgrade pre-checks and the upgrade stage 2 process by creating and passing an `UpgradeSpec`. In `UpgradeSpec`, you define the upgrade configuration and specify the source vCenter Server instance and the source ESXi host in `SourceApplianceSpec` and `LocationSpec`. See Figure: Upgrade Stage 2 Workflow . You can run the upgrading in silent mode by setting the `UpgradeSpec.auto_answer` to `true`. The default value of `UpgradeSpec.auto_answer` is `false` and the upgrading is in interactive mode, in which you must provide answers to the raised questions.

For information about the classes, variables, and default values, see the *`API reference`* documentation.

Historical Data Transfer

If you migrate vCenter Server for Windows, you can transfer the historical data of your source vCenter Server instance together with the core configuration data.

Deferred Import

The deferred import is a process of historical data transfer after the successful migration of a vCenter Server instance with an external database. The historical data includes statistics, events, and tasks.

By using the deferred import feature, you can postpone the historical data transfer after the migration process completes, so that you manage the downtime of your environment. You can select whether all historical data, or only events and tasks, will be migrated with the core data during the migration. The historical data transfer and deferred import of historical data are deactivated by default. You can enable and configure the historical data transfer by using the API, Key definition for "vami" not found in the DITA map., vCenter Server installer, or CLI installer. A vCenter Server super administrator can run and control the migration and deferred import processes.

If you use the deferred import feature, the historical data is migrated with the core data and the historical data import process starts automatically after a successful migration and when the vCenter Server instance is running. You can pause the historical data import and resume it later.

For information about how to configure and run the migration and deferred import processes by using the Key definition for "vami" not found in the DITA map., see the *vCenter Server Upgrade* documentation.

By using the API, you can configure, control, and monitor the data transfer process. If you use the API to enable the deferred import feature, you must create a history migration spec and set `defer_import` to `true`. For information about how to configure the deferred import by using the API, see the *API reference*.

The data import process has five states that you can check. If the historical data migration and the deferred import are configured, the historical data import starts automatically after a successful migration.

Figure 115: Deferred Import State Diagram

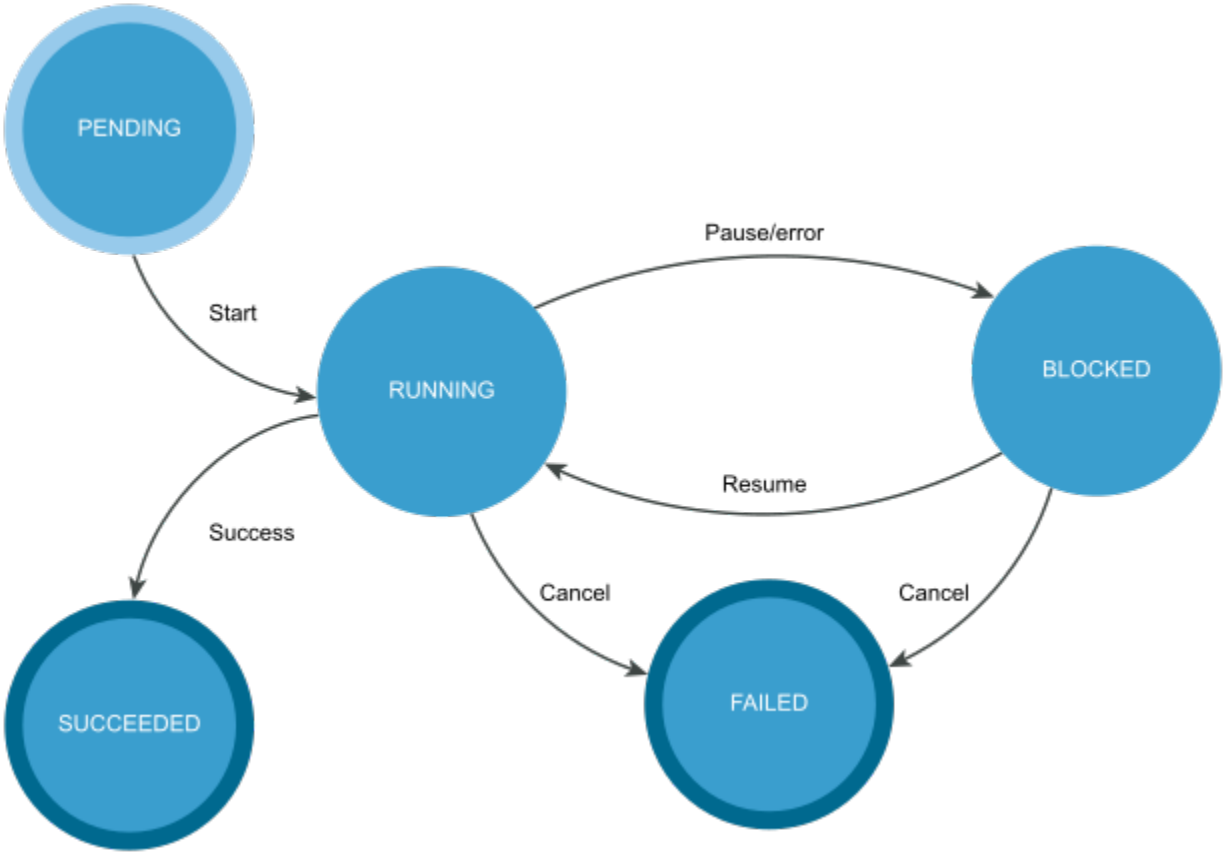


Table 82: User Operations

Operation	Description
Pause	You can pause the running data transfer process. If the data transfer is paused, you can resume or cancel it.
Cancel	You can cancel the data transfer process if it is in a `RUNNING` or `BLOCKED` state. **NOTE** If you cancel the data transfer, the process enters in a final `FAILED` state and you cannot resume the transfer.
Resume	You can resume the stopped data transfer.
Get status	You can retrieve the status of the data transfer process. There are five states. **`PENDING`** The transfer is not started. **`RUNNING`** The transfer is started or resumed. **`BLOCKED`** The transfer is paused or there was a recoverable error, such as not enough disk space, during the import. **`SUCCEEDED`** The transfer is successful. **`FAILED`** The transfer is canceled.

You can run the deferred import operations by using the API or sending an HTTP request.

NOTE
When you send the requests, you must use an authentication.

If you pause the data transfer by using the API or an HTTP request, you can resume or cancel the process by using the API or the Key definition for "vami" not found in the DITA map..

IMPORTANT
If you cancel the transfer process, and want to transfer the historical data later, you must restart the migration process.

Historical Data Import Errors

If an error appears during the data import, the import stops and the process enters in a `BLOCKED` state. You can resume the data import after you eliminate the cause.

You can check the errors, warnings, and info messages by reading the `info`, `status`, and `notifications` data structures.

If the information in the data structures is not enough, you can download the vCenter Server support bundle from `<vcenter_ip_address_or_fqdn>:443/appliance/support-bundle` and check the log files.

Table 83: Log Files

Log File	Path
API log file	`/var/log/vmware/applmgmt/applmgmt.log`
Backend log file	`/var/log/vmware/upgrade/upgrade-post-import.log`
Upgrade Runner log file	`/var/log/vmware/upgrade/deferredimport-upgrade-runner.log`
Deferred import log file	`/var/log/vmware/upgrade/DeferredImport_com.vmware.vcdb_<date_time>.log`

vCenter Server Configuration

You can use the API to perform operations related to health monitoring and capacity monitoring of vCenter Server. You can also manage the global FIPS compliance and perform infrastructure profile management operations.

Health Monitoring of vCenter Server

The vCenter Server API offers health status indicators for several key components of the system. These indicators can be polled periodically to monitor the components for problems.

The health status indicators report graded values from green to red. The general meanings of the grades are as follows.

green
: The component is healthy.

yellow
: The component is healthy, but may have some problems.

orange
: The component is degraded, and may have serious problems.

red
: The component is unavailable, or will stop functioning soon.

gray
: No health data is available.

Check Overall System Health of vCenter Server

vCenter Server provides a composite health indicator that enables you to test a single value that represents the health of all system components. This procedure shows how to test the composite health indicator.

Verify that you have an active authenticated session with vCenter Server. This procedure assumes that the session ID is present in the security context of a stub configuration.

The value of the overall system health indicator reflects the strongest trouble indication among the vCenter Server components. If any component has a `red` indicator, the overall system health indicator is `red`, else if any component has an `orange` indicator, the overall system health indicator is `orange`, and so on.
A `gray` value for any component indicates that data for the subsystem is unknown. If one or more components have a `gray` value, but all other subsystems are `green`, then the overall system health indicator is `gray` rather than `green`. However, if any component has a definite trouble indication, the overall system health indicator reflects the strongest trouble indication among the components.

1. Create an interface stub or REST path that uses the stub configuration.

2. Invoke the `health.system` method.
3. Format and display the resulting value.

Capacity Monitoring ofvCenter Server

vCenter Server keeps a history of statistics that you can use to monitor resources used by thevCenter Server instance.

You can use the statistics to spot peak usage demands or to monitor trends for advance warning of potential resource exhaustion.

Frequency and Retention of Statistics Collection invCenter Server

vCenter Server collects statistics from the guest operating system at regular intervals and stores them in a database. Users can query the statistics in the database by selecting a time period and a roll-up function thatvCenter Server applies to the statistics before returning them to the client.

After the monitoring service starts up, it begins requesting statistics from the guest operating system periodically, at a frequency that depends on the type of statistic. The service requests storage statistics once every 10 minutes, while it requests memory, CPU, and networking statistics once per minute. The collection times are fixed relative to the startup time of the monitoring service, rather than to clock time.

The monitoring service retains statistics approximately 13 months, by default. Older statistics are deleted by the service, creating a 13-month moving window within which you can query statistics. You can choose to delete statistics as needed to conserve storage resources.

Nature of Statistics invCenter Server

vCenter Server supplies statistics of several types.

The guest operating system computes statistics either as rates, such as CPU cycles per second, or as snapshots of size, such as KB used for storage. Statistics stored as size snapshots are collected at the end of their sample periods. Statistics stored as rates are computed as averages of values sampled frequently during each sample period.

When you query the statistics database, the units are not returned with the data, but you can determine the units for any metric by requesting metadata for the metric with the `get()` method.

Requesting Statistics fromvCenter Server

To request statistics, you must construct an appropriate request structure to filter statistics from the database.

To request data or metadata for a metric, you must supply the ID of the metric. You can get a list of metric IDs by using the `list()` method, which returns information on all available metrics.

When you query statistics, you provide a list of IDs to specify the metrics in which you are interested. You also supply a start time, an end time, a roll-up interval, and a roll-up function. These values interact as follows to determine the data returned to you.

- The response contains a list of data points for each metric ID you specified in the request.
- The start time and end time control the limits for the data you want in the response. The response contains data points only for statistics that have timestamps between those limits, inclusive of the endpoints. However, the start time is adjusted to a round number, in some cases.
- The roll-up interval enables you to control the granularity of the data points in the response. Rather than a response with a data point for every statistic between the start time and end time, you get a response with a number of data points equal to the number of intervals between the start and end times. Generally, you should specify a time period that is an even multiple of the interval, so that each data point in the response represents the same number of statistics.

- The roll-up function specifies how the response summarizes the statistics that fall within each interval. The resulting data point can be the maximum statistic value within collection interval, or the mean of the statistics values within the interval, and so on.

Statistics Collection Times

The actual time that a statistic was collected is not readily predictable.

The API does not enable you to determine the exact time that a statistic was collected. Furthermore, some statistics, such as those for storage metrics, might take seconds or minutes to collect, so that they are not available immediately at the time a request is made to the guest operating system.

However, because statistics are collected at regular intervals, and roll-up intervals for a request generally all have the same size, each data point in the response represents the same number of statistics as the others.

Statistics Interval Adjustment invCenter Server

When you make a request for statistics, the monitoring service might adjust the specified roll-up interval times to improve the appearance of statistics graphs in a graphical interface.

The monitoring service adjusts the start time of a data collection request when it is not an exact multiple of the interval length. In these instances, the start time is rounded downward to the previous UTC time that is a multiple of the interval. All subsequent intervals of the data collection are also adjusted to align with the new start time.

For example, if the start time is 10:31 and the interval length is 1 hour, the monitoring service adjusts the start time to 10:00 and the roll-up intervals have the following continuous pattern.

- 10:00 to 10:59:59.999
- 11:00 to 11:59:59.999
- 12:00 to 12:59:59.999

The monitoring service does not adjust the end time of a data collection. Consequently, the response to a statistics query might contain one more data value than expected, or an incomplete final interval might be lengthened.

Empty Data Values

In some instances, you might encounter a response that reports an empty data value, or even a series of empty data values. This might manifest as a list of data values containing some numeric values alternating with empty values.

- Empty data values can happen when the report time period is too short to be certain of containing any statistics. For instance, a time period of 30 seconds is half the length of the sample period for network metrics, so you have only a 50% chance of finding a network statistic during any 30-second reporting period.
- Empty data values can also happen when the interval is shorter than the sample period for a metric you have requested. In this case, some data points are present in the list, while others are empty because no statistic was collected during those intervals. For instance, an interval of 5 minutes is only half the length of the sample period for storage metrics, so every second data value is empty.
- Empty data values can also happen when the monitoring service has not finished collecting and writing the last sample to the database, even if the nominal sample timestamp falls within the report time period. For example, calculation of storage used can delay writing a storage statistic to the database. A request for the statistic during that delay time produces an empty data point in the response.

When a response contains an empty data value, this indicates that no statistics were collected during a collection interval. An appropriate action for the client in such a case depends on how the client is using the data. For example, if you are graphing a resource usage trend, you might choose to interpolate for the missing value to produce a smooth line.

Check Database Usage in vCenter Server

vCenter Server contains a database of all objects managed by the vCenter Server instance. In addition to inventory objects, the database includes vCenter Server statistics, events, alarms, and tasks. You can calculate the database storage consumption by adding the sizes of all data categories.

This task assumes you have previously authenticated and created a client session.

You need to monitor storage consumption in vCenter Server.

1. Prepare a request for database usage statistics.

   Include metric IDs both for `vcdb_core_inventory` and `vcdb_seat`. The name `vcdb_seat` refers to Statistics, Events, and Tasks in the vCenter Server database.

2. Issue the request to the API endpoint.
3. Process the resulting data points as needed.
4. Format and print the results.

The result of this procedure shows the storage used in the vCenter Server database, which includes storage overhead used for indexes and other optimizations beyond the actual size of the data.

List Storage Consumption By Data Type in vCenter Server

vCenter Server provides statistics on several types of storage.

Verify that you have authenticated and created a client session.

For example, you can query statistics about inventory storage, transaction log, and vCenter Server tasks. Many of these statistics are available both for storage consumed and storage available.
This task provides data for system administrators who need to monitor storage consumption in the guest operating system of vCenter Server.

1. Prepare a request for database usage statistics.

   Include metric IDs for each data type you want to monitor.

2. Issue the request to the API endpoint.
3. Process the resulting data points as needed.
4. Format and print the results.

You retrieved information about the storage consumption in your vCenter Server system.

Managing the Global FIPS Compliance

You can retrieve information about the current FIPS (Federal Information Processing Standards) settings of vCenter Server. You can also enable or deactivate the global FIPS compliance.

FIPS 140-2 is a U.S. and Canadian government standard that specifies security requirements for cryptographic modules. vSphere uses FIPS-validated cryptographic modules to match those specified by the FIPS 140-2 standard. The goal of vSphere FIPS support is to ease the compliance and security activities in various regulated environments.

The following table lists the operations that you can perform to manage the FIPS settings of your vCenter Server system.

Table 84: User Operations

Operation	Description
Get FIPS status	You can check whether the global FIPS compliance is currently enabled on the vCenter Server system.
Manage FIPS status	You can enable or deactivate the global FIPS compliance on the vCenter Server system.

NOTE
When you enable FIPS compliance, some components might present functional constraints. For more information, see *vSphere Security*.

You can run FIPS management operations by using the vSphere Automation SDK or sending an HTTP request.

Performing Infrastructure Profile Management Operations

You can export an existing vCenter Server configuration and import it to other vCenter Server instances.

You can export multiple configuration profiles at once. The exported data can contain general configuration settings and user content. You can replicate the same configuration across all vCenter Server instances in your environment by importing the same data package. You can also use the exported data as a backup if you need to revert to the last known good configuration. To avoid configuration issues, you can validate the exported data before importing it to a vCenter Server instance.

The following table lists the operations that you can perform to manage the configuration profiles in your infrastructure.

Table 85: User Operations

Operation	Description
List configuration profiles	You can retrieve a list of all configuration profiles that are registered with vCenter Server.
Export configuration profiles	You can export specific vCenter Server configuration profiles.
Validate configuration profiles	You can validate the exported vCenter Server configuration profiles. The validation process examines the configuration file for possible errors and conflicts and returns output. This operation can help avoid configuration issues or loading the wrong configuration file.
Import configuration profiles	You can import specific vCenter Server configuration profiles into another vCenter Server instance.

You can run infrastructure profile management operations by using the vSphere Automation SDK or sending an HTTP request.

Patching and Updating vCenter Server Deployments

You can use the API to perform operations related to the planning and installing of vCenter Server software updates.

Planning vCenter Server Updates

The vCenter Server API provides operations that can help you plan the life cycle of vCenter Server instances in your environment.

You can use the operations to discover VMware products that can be associated with vCenter Server, list associated products, and manage product associations. You can also get details about available vCenter Server updates, perform pre-checks, and produce reports. The reports can contain interoperability or pre-check information. Interoperability reports

contain information about the interoperability between the associated products and a specific vCenter Server version. Pre-check reports contain information about the compatibility of the current vCenter Server version with a pending update version. You can plan to perform vCenter Server updates based on the information gathered in the produced reports.

Performing Discovery and Planning Operations

You can retrieve information about VMware products associated with vCenter Server, list available vCenter Server updates, and produce reports. The findings can help you plan vCenter Server updates in your environment.

The life cycle management API provides operations that are grouped in the discovery, update, and reports categories. The discovery functionality of the API consists of the operations in the discovery category. The planning functionality of the API consists of the operations in the update category. Both functionalities can produce reports that you can download by using the reports category.

The discovery category provides operations for listing VMware products that can be associated with vCenter Server, managing products associations, and creating interoperability reports. The update category provides operations for listing all available updates and upgrades for vCenter Server and generating a pre-check compatibility report. The reports category provides an operation for downloading reports generated by interoperability and pre-check operations. By using the retrieved information, you can select one of the available patches and can plan an actual patch or upgrade for a specific vCenter Server version.

The following table lists the operations that are available in the discovery category.

Table 86: Discovery User Operations

Operation	Description
Get product catalog	You can retrieve a list of all VMware products that can be associated with vCenter Server.
List associated products	You can retrieve a list of all VMware product deployments in the environment that are associated with vCenter Server. **NOTE** The list contains both product deployments discovered automatically and deployments registered manually through the API.
Get associated product information	You can retrieve detailed information about a product associated with vCenter Server.
Create product association	You can manually associate a VMware product with vCenter Server.
Update product association	You can modify a manually added VMware product that is associated with vCenter Server. **NOTE** You cannot modify VMware products that are discovered automatically.
Delete product association	You can delete or dissociate a manually added VMware product that is associated with vCenter Server. **NOTE** You cannot delete or dissociate VMware products that are discovered automatically.
Create interoperability report	You can create an interoperability report between a vCenter Server release version and all products registered with the vCenter Server instance.

The following table lists the operations that are available in the update category.

Table 87: Update User Operations

Operation	Description
List updates	You can retrieve a list of all available vCenter Server updates. The list can contain minor, in-place, updates and major, migration-based, upgrades.
Get update info	You can retrieve detailed vCenter Server information about a specific update or upgrade.
Create pre-check report	You can create a vCenter Server pre-update compatibility check report for a pending update version. **NOTE** You can export and download the report in CSV format.

The following table lists the operations that are available in the reports category.

Table 88: Reports User Operations

Operation	Description
Get report	You can download the report generated by the interoperability and pre-check operations. For information about downloading the report, see the *API reference* documentation.

You can run life cycle management operations by using the vSphere Automation SDK or sending an HTTP request.

List Available Products and Manage Associated Products

You can automate the management of VMware products associated with vCenter Server by using the API.

- Verify that you have an active authenticated session with vCenter Server.

This procedure includes the operations that you can use to manage the product catalog and associated products.

1. Create a stub configuration.
2. Retrieve the product catalog.
3. Retrieve a list of VMware products associated with vCenter Server.
4. Associate a VMware product with vCenter Server.
5. Update a VMware product associated with vCenter Server.
6. Delete a product form the list of VMware products associated with vCenter Server.

List Available Updates

You can retrieve a list of available vCenter Server updates, details about the updates, and pre-check information by using the API.

- Verify that you have an active authenticated session with vCenter Server.

1. Create a stub configuration.
2. Retrieve a list of available vCenter Server updates.

   If there are available updates, you can retrieve details about the updates.
3. Retrieve pre-check information.

Retrieve a Report

You can retrieve a report generated by the interoperability and pre-check operations by using the API.

- Verify that you have an active authenticated session with vCenter Server.

1. Create a stub configuration.
2. Retrieve the report details.

Updating vCenter Server

vCenter Server provides interfaces to perform software updates.

Before applying updates, you must make sure that your environment is prepared for the vCenter Server software update process.

ApplyingvCenter Server Software Updates

You can automate the installation ofvCenter Server software updates to ensure that your system is stable and protected. Software updates can include security fixes, performance optimizations, and new features.

Security patches usually address vulnerabilities in third-party components and do not affect thevCenter Server functionality.vCenter Server bug fixes can introduce changes to the functionality without affecting the data format or database schema of the system.

Each update contains metadata with information about the updated content, for example, whether high-priority OS updates are included. The update metadata includes a list of components to be updated, the release date of the update, a list of fixed issues, time and disk space requirements, and information whether a reboot is required. The metadata can also contain a newvCenter Server version number, including a build number. In addition to the metadata, an update can contain optional components such as update scripts, new versions ofvCenter Server software components, and new versions of OS components.

vCenter Server can obtain software updates from either a URL or an ISO image. The URL can either point to the VMware Web repository or to a custom repository in which you upload the updates in ZIP format. To perform an update by using an ISO image, attach the image to the CD/DVD drive of thevCenter Server instance.

If you want to prevent issues related to the possibility of update installation failures, you should create a backup or take a snapshot of yourvCenter Server instance before you start the update process. A backup can also be useful when an update is successfully installed. For example, you might decide to revert to the previous version if you encounter any undesired system behavior related to functional changes in the new software version.

Table 89: User Operations

Operation	Description
Get state information	You can retrieve information about the update state.
Check for update	You can check whether a new update is available.
Get update information	You can retrieve information about the available updates.
Get update requirements	You can retrieve information about the update requirements.
Stage	You can initiate the download of the update. **NOTE** The check phase must have completed successfully before you can stage the update.
Get staging status	You can retrieve information about the status of the stage operation.

Table continued on next page

Continued from previous page

Operation	Description
	NOTE You must provide the task ID value that you received as a response when you initiated the stage operation.
Install	You can initiate the installation of the update. **NOTE** The update must be staged before you can install it.
Get installation status	You can retrieve information about the status of the install operation. **NOTE** You must provide the task ID value that you received as a response when you initiated the install operation.
Stage and install	You can initiate the download of the update and the installation starts when the download completes.

You can run software update operations by using thevSphere Automation SDK or sending an HTTP request.

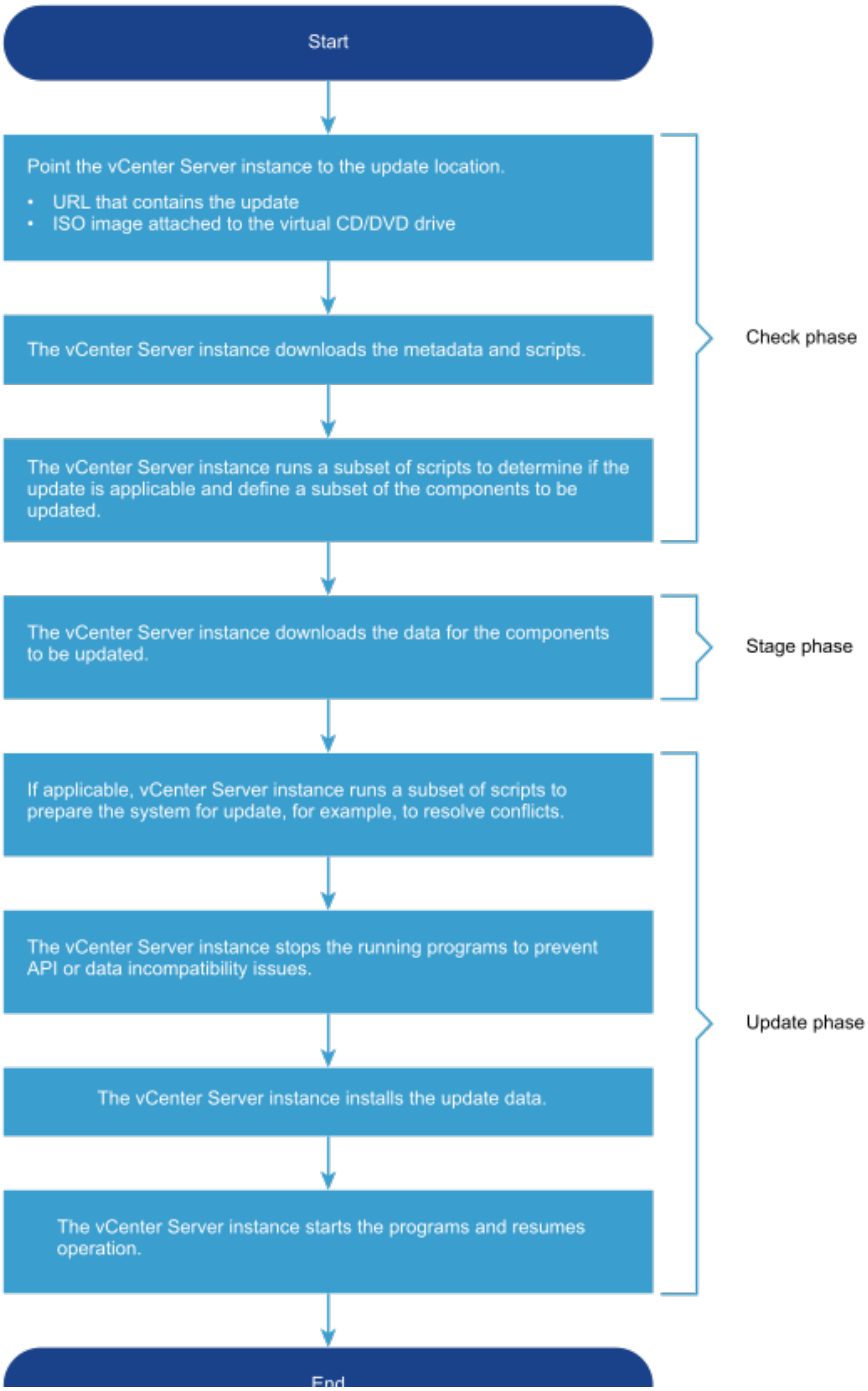
vCenter Server Software Update Workflow

The vCenter Server software update process consists of three major phases. In the first phase, the vCenter Server instance performs various checks, in the second phase it stages the update, and applies the update in the final phase.

To initiate the update process, you must choose whether the vCenter Server instance should obtain software updates from a URL or an ISO image. If you use an ISO image to update the vCenter Server instance, the image must remain attached to the CD/DVD drive of the instance during the stage and install operations.

The workflow in Figure: Update Process Workflow describes the standard steps of the update process.

Figure 116: Update Process Workflow



You can automate checks for new updates and staging of updates by using an update policy. For example, you can set an update policy to make the vCenter Server instance perform automatic checks for new updates at midnight every day. If there are new updates available, the vCenter Server instance can stage them automatically. Using an update policy reduces the waiting time by automating the first two phases and giving you the option to initiate only the update phase manually.

Developing Remote Plug-ins with the vSphere Client SDK

Developing Remote Plug-ins with the vSphere Client SDK provides information about developing and deploying HTML-5 extensions to the VMware vSphere® Client user interface.

VMware provides various APIs and SDKs for different applications and goals. This documentation provides information about the extensibility framework of vSphere Client for developers who are interested in extending the client application with custom functionality.

The vSphere Client SDK

The vSphere Client provides a means for connecting to VMware vCenter Server® systems and managing objects in the vSphere infrastructure. The vSphere Client is an HTML5-based web application with a modular architecture that supports plug-in extensions. The vSphere Client SDK provides tools and examples that help you create custom plug-ins to extend the functionality of the vSphere Client.

VMware once offered local plug-ins for the vSphere Client SDK, but these are replaced by remote plug-ins. Remote plug-ins are compatible with VMware cloud, easier to upgrade, more secure, perform better, and support a wide variety of configurations.

Intended Audience

This information is intended for anyone who wants to extend the vSphere Client with custom functionality. Users typically are software developers who use HTML and JavaScript to create graphical user interface components that work with VMware platforms.

Revision History

This book, *Developing Remote Plug-ins with the vSphere Client SDK*, is updated with each release of the product or when necessary.

The table below provides an update history of this book.

Revision	Description
18 JUN 2024	Release with vSphere 8.0 U3: • Dynamic actions can be hidden or disabled. • Add vCenter version parameters and per-vCenter extensions. • Improvements to extension registration tool. • Fix functionality of multi-manifest and describe its backward compatibility. • New feature to define the size of summary view cards.
21 SEP 2023	Release with vSphere 8.0 U2: • Users can monitor progress of solution installation in the Tasks console. • No longer necessary to have `plugin.json` for multi-manifest support. • Additional properties for vCenter Servers injected into OVF. • Developers without `vim25.jar` must obtain it from vSphere Management SDK. • Plug-in servers can be registered with certificate in addition to or instead of thumbprint.

Table continued on next page

Continued from previous page

Revision	Description
18 APR 2023	Release with vSphere 8.0 U1: • New type "separator" can define dividers between menu actions. • Ability to define multiple summary cards (three later increased to five). • New property `requiredVcVersion` available with integrated solution installer.
29 MAR 2023	Update: • Added `solutioninstall` API to integrated solution installer chapter.
14 JAN 2023	Corrections: • Corrected specification for app.ClientViewNavigationOptions. • Added vCenter Server thumbprint to registration example. • Removed obsolete material in Best Practices and Troubleshooting chapters. • Corrected statement about resizing Summary views.
11 OCT 2022	vSphere 8.0 release. • Change occurrences of 'disable' to 'deactivate'. • Update documentation to reflect the change to Clarity cards in place of portlets. • Add advice to label the primary server with type MANIFEST_SERVER in the registration record. • Add a chapter about the integrated solution installer. See Integrated Solution Installer for the vSphere Client. • Add material about dynamic Summary views. See Dynamic Extensions for Remote Plug-ins. • Add material about using auxiliary servers for dynamic filter queries. See Dynamic Extensions for Remote Plug-ins. • Refresh screenshots in Choosing Extension Points for vSphere Client Plug-ins and Viewing the vSphere Client Remote Plug-in Sample.
14 OCT 2021	Streamlined description of remote plug-in sample directory structure.
05 OCT 2021	vSphere Client SDK 7.0 Update 3 release. • Expanded material on server back end authentication to explain more flexible token exchange and access feature. • Improved diagram of authentication communicatons. • Added MacOS examples to Remote Sample Starter. • Updated information about SHA-1 deprecation. • Removed troubleshooting topic about OSGi (inapplicable for remote plug-ins).

Table continued on next page

Continued from previous page

Revision	Description
	• Removed obsolete reference to `updateOnBrowserRefresh`. • Noted Administrator role is required for Redeploy feature. • Removed obsolete references to external Platform Services Controller.
09 MAR 2021	vSphere Client SDK 7.0 Update 2 release. • Expand information about auxiliary plug-in servers. • Expand material about plug-in instance handling. • Corrected final step in using `cloneSession()` method. • Improvements to the chapter about using the Remote Plug-in Sample. Added Remote Plug-in Sample Starter instructions. • Added screen shots to illustrate UI context for extension points.
06 OCT 2020	vSphere Client SDK 7.0 Update 1 release. • Dynamic extension support. • Multiple targets for plug-in actions. • Add plug-in server logging configuration to server startup command. • Add production build option to SDK sample. • Add chapter for best practices. • Multi-manifest capability to support different versions. • Expanded material concerning extension points. • At VMware, we value inclusion. To foster this principle within our customer, partner, and internal community, we are replacing some of the terminology in our content. We have updated this guide to remove instances of non-inclusive language.
04 MAY 2020	Cosmetic improvements.
02 APR 2020	vSphere Client SDK 7.0 release. • Extended and updated JavaScript API. • SHA-256 thumbprint support. • Support for auxiliary plug-in servers. • Support for theme-dependent icons. • Improved discovery of new plug-ins or updates. • Plug-in compatibility support. • Redeploy button in development mode.
20 AUG 2019	vSphere Client SDK 6.7U3 release. • Corrected MOB URL in Troubleshooting chapter. • Added Additional Resources chapter.
20 JUN 2019	Minor corrections to Troubleshooting chapter.
30 MAY 2019	Minor updates and corrections. Added Troubleshooting chapter.

Table continued on next page

Continued from previous page

Revision	Description
11 APR 2019	Changes for vSphere Client SDK 6.7U2 release. • Replaced Virgo server with Tomcat server. • Updated JavaScript API to handle UI themes. • Added chapter about UI themes. • Added information about plug-in caching. • Added information about plug-in handling in linked mode. • Expanded and improved chapter about running remote plug-in sample. • Improved information about plug-in deployment.
12 FEB 2019	Minor updates and corrections. Expanded material about Deployment.
16 OCT 2018	Initial release.

Using the vSphere Client Remote Plug-in Samples

For an introduction to the vSphere Client Remote Plug-in SDK, you can install and run the remote plug-in sample and the remote plug-in sample starter. The samples illustrate several key features that you can adapt to develop your own plug-ins.

This chapter assumes that you have access to a vCenter Server instance and a development machine where you build and run the sample plug-in server.

Try the Remote Plug-in Sample Starter

Build and run the remote plug-in sample starter for a quick introduction to the process of deploying and viewing a remote plug-in for the vSphere Client.

The plug-in sample starter demonstrates a very simple plug-in that serves only static HTML content. This is the easiest way to be introduced to the remote plug-in environment, including only a few components that extend the vSphere Client user interface with a full-screen view available for plug-in content.

What To Do First

Check the dependencies on your development machine:
- Java Development Kit 8
- Maven 3

Install the vSphere Client SDK:

1. Download the SDK from https://code.vmware.com/sdk/client
2. In a shell window, unzip the SDK file:
   PowerShell:

```
PS C:\> Get-Item vsphere-client-sdk-*zip |
 Foreach-Object { Expand-Archive -Path $_ -DestinationPath C:\ }
```

   MacOS:

```
Downloads % unzip vsphere-client-sdk-*zip -d ~
```

Generate a Plug-in Server Certificate

1. Generate an SSL certificate and a JKS key store for the remote plug-in server using Java's **`keytool`** certificate management utility.
   PowerShell example:

   ```
   PS C:\> keytool -genkeypair -alias plugin-server -keyalg RSA -keysize 4096 `

   -storetype JKS -keystore keystore.jks -storepass "remote.plugin" -keypass
   "remote.plugin" `

   -dname "CN=mydevbox"
   ```

   MacOS example:

   ```
   Downloads % keytool -genkeypair -alias plugin-server -keyalg RSA -keysize 4096 \

   -storetype JKS -keystore keystore.jks -storepass "remote.plugin" -keypass
   "remote.plugin" \

   -dname "CN=mydevbox"
   ```

   NOTE
   Do not change the values of arguments to the **`keytool`** utility, except the value *`mydevbox`* in the `-dname` "CN=mydevbox" argument. The value *`mydevbox`* must be replaced by the IP address or the DNS name of the machine on which the remote plug-in server will run. The value is significant because vCenter Server will check whether the Common Name (CN) field value of the remote plug-in server certificate matches the host through which remote plug-in resources are served.

2. Replace the default *`keystore`* in the Remote Plug-in Sample Starter code with the newly generated one:
   PowerShell example:

   ```
   PS C:\> Move-Item -Path "keystore.jks" -Destination `

    "html-client-sdk\samples\remote-plugin-sample-
   starter\src\main\resources\keystore.jks" `

    -Force
   ```

   MacOS example:

   ```
   Downloads % mv -f keystore.jks \

    ~/html-client-sdk/samples/remote-plugin-sample-starter/src/main/resources/
   keystore.jks
   ```

Build the Remote Plug-in Sample Starter

1. Navigate to the sample starter directory:
   PowerShell example:

   ```
   PS C:\> cd html-client-sdk/samples/remote-plugin-sample-starter
   ```

   MacOS example:

```
Downloads % cd ~/html-client-sdk/samples/remote-plugin-sample-starter
```

2. Issue the Maven build command:
   PowerShell example:

   ```
   PS C:\html-client-sdk\samples\remote-plugin-sample-starter> mvn clean install
   ```

   MacOS example:

   ```
   remote-plugin-sample-starter % mvn clean install
   ```

Start the Remote Plug-in Sample Starter

1. Navigate to the `target` directory:
   PowerShell example:

   ```
   PS C:\html-client-sdk\samples\remote-plugin-sample-starter> cd target
   ```

   MacOS example:

   ```
   remote-plugin-sample-starter % cd target
   ```

2. Issue the Java command to run the plug-in server:
   PowerShell example:

   ```
   PS C:\html-client-sdk\samples\remote-plugin-sample-starter\target> `
   java -jar (Get-ChildItem remote-plugin*.jar -Name | Select-Object -first 1)
   ```

   MacOS example:

   ```
   target % java -jar remote-plugin-sample-starter-*.jar
   ```

3. Use a browser to verify that the remote plug-in server can supply its manifest. Enter the following URL:
   `https://localhost:8443/sample-ui/plugin.json`

   Leave the browser window open for the next step.

Save the Certificate or Fingerprint of the Plug-in Server

1. Find the security certificate and/or fingerprint of the plug-in server.
   In Chrome browser for example, click the padlock icon next to the URL at the top of the browser window. Then click **Connection is secure** > **Certificate is valid**. You can copy the SHA-256 Fingerprint or Thumbprint (depending on the browser) from there. For the certificate, click **Details** > **Export**.
2. Paste the SHA-256 fingerprint into a file, or save the certificate as a PEM file with `.cer` suffix.
   vCenter Server expects the fingerprint (thumbprint) to contain colon-separated character pairs. As of vSphere 8.0 U3, the registration script changes unseparated or space-separated character pairs into the expected format.

   For PEM, select base64-encoded ASCII, single certificate. In the plug-in server certificate, check that the server host name is specified according to subjectAltName extension (RFC 2818, Section 3.1 Server Identity).

Register the Remote Plug-in Sample Starter

1. Navigate to the directory containing the registration script:
   PowerShell example:

```
PS C:\html-client-sdk\samples\remote-plugin-sample-starter\target> `
cd 'C:\html-client-sdk\tools\vCenter plugin registration\prebuilt'
```

MacOS example:

```
target % cd ~/html-client-sdk/tools/vCenter\ plugin\ registration/prebuilt
```

2. Run the registration script for the plug-in. Use the name of your vCenter Server instance in the URL and use valid credentials to authenticate. Enter the password for `administrator@vsphere.local` when prompted.
   PowerShell example:

```
PS C:\html-client-sdk\tools\vCenter plugin registration\prebuilt> `
 .\extension-registration.bat `
 -action registerPlugin -remote `
 -url https://myvcenter.example.com/sdk `
 -username administrator@vsphere.local `
 -key com.vmware.sample.remote.starter -version 1.0.0 `
 -pluginUrl https://mydevbox:8443/sample-ui/plugin.json `
 -serverThumbprint 19:FD:2B:0E:62:5E:0E:10:FF:24:34:7A:81:F1:D5:33:`
19:A7:22:A0:DA:33:27:07:90:0F:8E:8D:72:F1:BD:F1 `
 -serverCertificateFile C:\plugin-server.cer `
 -c 'Example, Inc.' -n 'Remote Plug-in' -s 'This is a sample plug-in'
```

   MacOS example:

```
prebuilt % ./extension-registration.sh \
 -action registerPlugin -remote \
 -url https://myvcenter.example.com/sdk \
 -username administrator@vsphere.local \
 -key com.vmware.sample.remote.starter -version 1.0.0 \
 -pluginUrl https://mydevbox:8443/sample-ui/plugin.json \
 -serverThumbprint 19:FD:2B:0E:62:5E:0E:10:FF:24:34:7A:81:F1:D5:33:\
19:A7:22:A0:DA:33:27:07:90:0F:8E:8D:72:F1:BD:F1 \
 -serverCertificateFile /tmp/plugin-server.cer \
 -c 'Example, Inc.' -n 'Remote Plug-in' -s 'This is a sample plug-in' -insecure
```

When registering the plug-in, it's highly recommended to provide the SSL certificate of the plug-in server in addition to the thumbprint of the SSL certificate of the plug-in server. Performing a full SSL certificate check during SSL handshake is more secure than performing an SSL certificate thumbprint check. Furthermore, in future vSphere releases, support for SSL certificate thumbprints will be dropped and full SSL certificates will be required.

View the Plug-in User Interface

1. Connect a web browser to the vCenter Server instance and log in to the vSphere Client. Example URL: https://myvcenter.example.com/ui/
2. Navigate to the Home screen (for example, use **ctrl-alt-home**) or the Shortcut screen.
3. In the object navigator pane on the left of the home screen, click on the remote plug-in sample starter. The vSphere Client displays a plug-in welcome screen.

Remote Plug-in Sample UI and Server

The remote plug-in sample demonstrates a secure, efficient, remote plug-in that was developed according to recommended practices. The sample supports a Global View extension to the vSphere Client. The user interface component creates a modal dialog, views that extend vSphere inventory objects, and examples of using a card to extend a Summary view.

The user interface code also shows how to:
- Initialize the Client API.
- Retrieve session authentication information and pass it to the plug-in server.
- Perform a data retrieval request to the plug-in server.
- Define a context action for a VirtualMachine object.

The plug-in server code shows how to:
- Respond to a data retrieval or modification request.
- Clone and cache a user session with a vCenter Server.

The remote plug-in sample does the following actions:
- Creates a global view.
- Opens a modal dialog.
- Authenticates to vCenter Server.
- Performs a data retrieval call.
- Creates a relation between a Chassis object and a HostSystem object.
- Creates several views that extend context object views.
- Defines an action on a VirtualMachine context object.

Components in the Remote Plug-in Sample

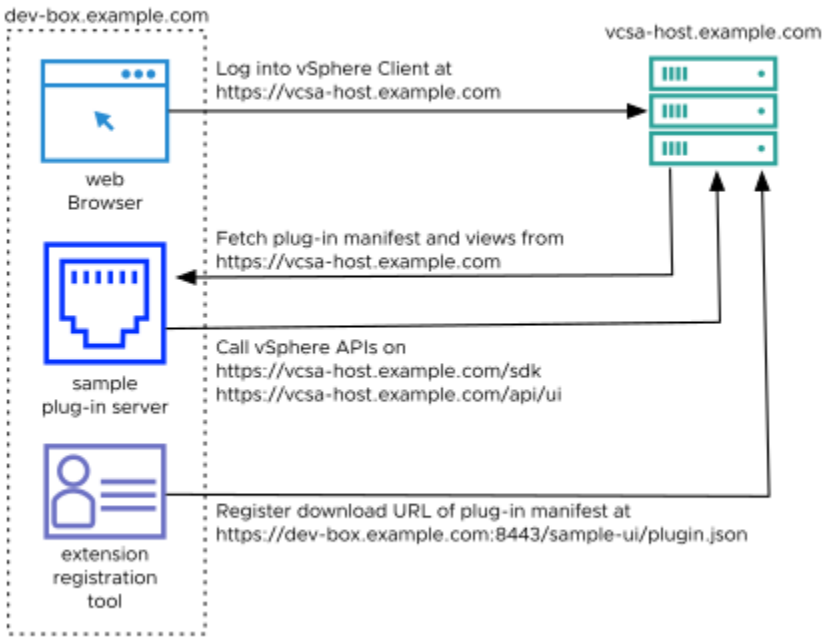
The remote plug-in sample in the vSphere Client SDK shows how to design and implement, deploy and register a remote plug-in. The sample is functionally simple, to focus on displaying the infrastructure rather than the business logic.

The sample remote plug-in package contains several components:

- The user interface is written in JavaScript and Angular, using Clarity design components to maintain compatibility with the vSphere Client user interface.
- The sample plug-in server is written in Java, but Java is not a requirement. The server includes the following:
  - In-memory data storage for fictitious Chassis objects.
  - Controller logic to handle user interface requests for Chassis objects and vSphere HostSystem objects.
  - Service interfaces for operations on both kinds of objects.
  - A library layer to interface to vCenter Server, including logic to handle delegated authentication.
- The `plugin.json` file specifies the vSphere Client extension points that the plug-in extends.
- The `spring-context.xml` file contains the Spring bean definitions for the plug-in server.
- The `pom.xml` file specifies how Maven will install dependencies and build the plug-in deliverable.
- The `application.properties` file specifies properties that the Spring application server uses to deploy the plug-in server.

The following diagram illustrates the basic architecture of the remote plug-in sample, when installed in a simple vSphere environment.

Figure 117: Remote Plug-in Sample Environment



Build the vSphere Client Remote Plug-in Sample

You build the remote plug-in sample with a Maven build directed by the sample's `pom.xml` file.

Before building the remote plug-in sample, you must have the following:

- You need Java 8 to compile the code for this sample.
- You need Maven 3 to build the plug-in package.
- Download the SDK and unzip it. See Remote Plug-in Sample Directory Structure for information about the structure of the SDK archive.

After you download the vSphere Client SDK, you must build the remote plug-in sample before you can run it.

1. Generate an SSL certificate and a JKS key store for the remote plug-in server using Java's **`keytool`** certificate management utility:
   PowerShell example:

   ```
   PS C:\> keytool -genkeypair -alias plugin-server -keyalg RSA -keysize 4096 `

   -storetype JKS -keystore keystore.jks -storepass "remote.plugin" -keypass
   "remote.plugin" `

   -dname "CN=mydevbox"
   ```

   MacOS example:

```
Downloads % keytool -genkeypair -alias plugin-server -keyalg RSA -keysize 4096 \

-storetype JKS -keystore keystore.jks -storepass "remote.plugin" -keypass
"remote.plugin" \

-dname "CN=mydevbox"
```

NOTE
Do not change the values of the arguments of **`keytool`** utility except the value *`mydevbox`* in the `-dname` "CN=mydevbox" argument. The value *`mydevbox`* must be replaced with the IP address or the DNS name of the machine on which the remote plug-in server will run. The value is significant because vCenter Server will check whether the Common Name (CN) field value of the remote plug-in server certificate matches the host through which remote plug-in resources are served.

2. Replace the default key store in the Remote Plug-in Sample code with the newly generated one.

   PowerShell example:

   ```
   PS C:\> Move-Item -Path "keystore.jks" -Destination \

   "html-client-sdk\samples\remote-plugin-sample\src\main\resources\keystore.jks" -Force
   ```

   MacOs example:

   ```
   Downloads % mv -f keystore.jks \

    ~/html-client-sdk/samples/remote-plugin-sample/src/main/resources/keystore.jks
   ```

3. Navigate to the sample directory:
   PowerShell example:

   ```
   PS C:\> cd html-client-sdk/samples/remote-plugin-sample
   ```

   MacOS example:

   ```
   Downloads % cd ~/html-client-sdk/samples/remote-plugin-sample
   ```

4. Install the Web Services API library into the local Maven repository:
   ```
   mvn validate
   ```
5. Build the sample:
   ```
   mvn clean install
   ```

   For production builds, substitute **`mvncleaninstall-Dproduction.build=true`** in step 5.

These steps install the Web Services API library into the local Maven repository, download and build the sample executable, download the Clarity design system, and build the JAR file that contains the sample components. The sample is a Spring Boot Application that will start an embedded Tomcat server when you run the sample in a command shell.

Choose a production build for the client code to run more efficiently. Choose a development build for a better debugging experience.

After you build the remote plug-in sample, you need to locate authentication information, start the plug-in server, and register the plug-in with vCenter Server.

Find the SSL Thumbprint and GUID of vCenter Server

Before you start the remote plug-in server, you need to find the certificate thumbprint (fingerprint) and GUID of the vCenter Server where you want to register the plug-in.

vCenter Server must be running while you do this procedure.

1. Connect a browser to the vCenter Server.
   The URL for vCenter Server looks similar to this: `https://vcenter-1.example.com`
   The browser displays a launch screen, with a small padlock icon in the address field.
2. Click **LAUNCH VSPHERE CLIENT (HTML5)** and log in to the vSphere Client.
   The browser displays the default Hosts and Clusters view.
3. If you connected to a vCenter Server instance in an extended linked mode environment, you must select the chosen vCenter Server instance in the navigation pane on the left of the vSphere Client screen.
   The URL in the browser address box contains an embedded managed object reference, similar to the following: `Folder:group-d1:56d373bd-4163-44f9-a872-9adabb008ca9`. This is an extended managed object reference that ends with the GUID of the vCenter Server instance. The GUID is a string of 32 hexadecimal digits, organized in groups of 4, 8, or 12 digits, separated by hyphens.
4. Copy the 32 hexadecimal digits of the GUID, along with the inset hyphens, and save this into a shell variable or a text file.

   You will use the GUID when you start the plug-in server.
5. Click the padlock icon in the browser address field to access a certificate information window.
   The browser displays a brief summary of browser properties.
6. Click **Details** to display more certificate information.
   The browser displays full details of the vCenter Server certificate.
7. Scroll through the certificate details to find the SHA-256 fingerprint.

   The SHA-256 fingerprint is a string of 64 hexadecimal digits, usually in pairs separated by spaces or other non-alphanumeric delimiters.
8. Select the fingerprint string and copy it to a text file.
9. Edit the text file to remove all spaces or other delimiters from the fingerprint string.
   You now have a string of 64 contiguous hexadecimal digits. This is the *`thumbprint`* of the vCenter Server instance.

The thumbprint and GUID of the vCenter Server instance are needed to start the remote plug-in server. After starting you will determine the plug-in server's certificate thumbprint.

Start the Remote Plug-in Server

The remote plug-in sample has an embedded Spring Boot application server. You can start the server by using a Java command with arguments that configure the server to communicate with a vCenter Server instance. This is the vCenter Server instance with which you will register the plug-in.

- Before you can run the remote plug-in sample, you must have built the SDK sample code.
- Before you run the remote plug-in sample, choose the vCenter Server where you will register the plug-in.
- Collect the following information about the vCenter Server instance:
  - Fully qualified domain name
  - Port number for Web Services API access
  - Certificate thumbprint

1. In a shell window, change to the root directory of the remote plug-in sample.
   `cd sdk/samples/remote-plugin-sample`
2. Run the JAR file in the `target` directory.

   The command to run the plug-in JAR file requires several arguments, including the thumbprint, GUID, DNS name, and HTTPS port number of the vCenter Server instance. You can also specify `--logging.path`, which creates a subdirectory (if it does not already exist) and stores server log files in the subdirectory. Use a command similar to the following example, but substitute the details that pertain to your vCenter Server:

   ```
   java -jar target/remote-plugin-sample-7.0.1.00000.jar \
     --logging.path=logdir \
     --vcenter.guid=223b94f2-af15-4613-5d1a-a278b19abc09 \
     --
   vcenter.thumbprint=274172e07a754b9811a4fb5fc45384a79a5c258d13fa1667185016f28685fc54 \
     --vcenter.fqdn=vcenter-1.example.com --vcenter.port=443
   ```

The plug-in application server runs. It might take a few minutes to initialize, and the console displays a number of output lines. When the server is ready, the console displays two lines saying `Tomcatstarted` and `StartedSpringbootApplication`.

```
2021-02-25 04:36:49.442  INFO 76 --- [           main]
 o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8443
 (https) with context path '/sample-ui'
2021-02-25 04:36:49.442  INFO 76 --- [           main]
 c.v.sample.remote.SpringBootApplication : Started SpringBootApplication in
 34.752 seconds (JVM running for 36.127)
```

If you have not yet registered the plug-in with a vCenter Server, find the plug-in server thumbprint and use it to register the plug-in. Registration enables the plug-in to render global views and object-specific views for objects managed by the vCenter Server instance.

Get Thumbprint or Certificate of Remote Plug-in Server

The remote plug-in sample has an embedded application server with a self-signed certificate that is used for encrypted communications. The certificate and its thumbprint are stored in a Java keystore file.

- Start the server.
- Find the server's port number. The default port number is 8443. You can configure a different port number in the `application.properties` file.

To register a remote plug-in with vCenter Server, you need to determine the thumbprint of the plug-in application server, or save its security certificate, which you then use in arguments to the registration command.

1. Connect a browser to the application server, for example using the URL of the plug-in manifest.

The default URL for the manifest file is `https://localhost:8443/sample-ui/plugin.json`.

2. Examine the certificate presented by the application server.

   This step depends on the browser. In Firefox for example, you click the padlock icon next to the URL, then select **Connection secure** > **More information** > **View Certificate**. The thumbprint is the field labelled **SHA-256** and the certificate can be downloaded by the **PEM (cert)** link.

   NOTE
   A SHA-1 fingerprint is also supported, but SHA-1 is deprecated in favor of SHA-256.

3. Save the thumbprint in a text file, or save the PEM certificate in a file with `.cer` suffix.

   vCenter Server expects the thumbprint (fingerprint) to contain colon-separated character pairs. As of vSphere 8.0 U3, the registration script changes unseparated or space-separated character pairs into the expected format.

Use the application server thumbprint or PEM certificate when you register the plug-in with vCenter Server.

Register the vSphere Client Remote Plug-in Sample

Before you can view the remote plug-in sample in the vSphere Client, you must register it with a vCenter Server instance to which you want to connect. The vSphere Client SDK contains a vCenter Server plug-in registration tool that registers a plug-in with a vCenter Server ExtensionManager.

- Start the server.
- If needed, start vCenter Server.
- If needed, change permissions on the plug-in registration tool to allow execute access.
- Collect the following information to prepare the registration command:
  - The host name or IP address of the machine where you are running the sample plug-in server. This address must be accessible to the vCenter Server instance so that it can download the plug-in manifest file.
  - The port number where the plug-in server receives HTTPS requests. The sample serves port 8443 by default. The port must be open on the firewall of your development machine and accessible to the vCenter Server.
  - The host name or IP address of the vCenter Server where you want to register the remote plug-in sample.
  - The username and password of a vSphere user who has permission to access the ExtensionManager on the vCenter Server where you want to register the remote plug-in sample. For example, `administrator@vsphere.local` normally has the necessary `Extension.Register` permission.
  - The SHA-256 thumbprint or SSL certificate of the plug-in server, so that vCenter Server can retrieve the plug-in manifest file. With a certificate, check that the host name in the plug-in server certificate is specified according to subjectAltName extension (RFC 2818, Section 3.1 Server Identity).

    NOTE
    vCenter Server expects the thumbprint (fingerprint) to contain colon-separated character pairs. As of vSphere 8.0 U3, the registration script changes unseparated or space-separated character pairs into the expected format.

  - The version number of the remote plug-in sample.
  - The key of the remote plug-in sample, defined in the plug-in manifest, `plugin.name` by default.
  - Path from plug-in web server root to the plug-in manifest file, `/sample-ui/plugin.json` by default.
  - Location of JSON files for `taskList`, `faultList`, `privilegeList`, and `resourceList`.

For more information about the registration tool, see vSphere Client Plug-in Registration Tool.

You have installed the SDK, built the sample, and are ready to run the remote plug-in.

1. In a command terminal or shell, change directory to the `prebuilt` folder, which contains an `extension-registration.bat` script for Windows, and an `extension-registration.sh` script for Unix or Linux.
   `cd html-client-sdk/tools/*plugin*/prebuilt`
2. Run the extension-registration script appropriate for your operating system, specifying the prerequisite parameters.
   For a Unix or Linux shell, use this syntax:

```
path2json=../../../samples/remote-plugin-sample/registration/resources

./extension-registration.sh  -action registerPlugin  -remote \

-url https://myvcenter/sdk \

-username administrator@vsphere.local  -password 'mysecret' \

-key com.vmware.sample.remote  -version 1.0.0 \

-pluginUrl https://mydevbox:8443/sample-ui/plugin.json \

-serverThumbprint 19:FD:2B:0E:62:5E:0E:10:FF:24:34:7A:81:F1:D5:33:\

19:A7:22:A0:DA:33:27:07:90:0F:8E:8D:72:F1:BD:F1 \

-serverCertificateFile /tmp/plugin-server.cer \

-c 'Example, Inc.'  -n 'Remote Plug-in'  -s 'This is a sample plug-in' \

--faultList $path2json/faultList.json --privilegeList $path2json/privilegeList.json \

--resourceList $path2json/resourceList.json --taskList $path2json/taskList.json
```

   For a DOS command shell, use this syntax:

```
set path2json=..\..\..\samples\remote-plugin-sample\registration\resources

.\extension-registration.bat  -action registerPlugin -remote ^

-url https://myvcenter/sdk ^

-username administrator@vsphere.local  -password "mysecret" ^

-key com.vmware.sample.remote  -version 1.0.0 ^

-pluginUrl https://mydevbox:8443/sample-ui/plugin.json ^

-serverThumbprint 19:FD:2B:0E:62:5E:0E:10:FF:24:34:7A:81:F1:D5:33:^

19:A7:22:A0:DA:33:27:07:90:0F:8E:8D:72:F1:BD:F1 ^

-serverCertificateFile C:\plugin-server.cer ^

-c "Example, Inc."  -n "Remote Plug-in"  -s "This is a sample plug-in" ^

--faultList %path2json%\faultList.json --privilegeList %path2json%\privilegeList.json
^

--resourceList %path2json%\resourceList.json --taskList %path2json%\taskList.json
```

   For a PowerShell prompt, use this syntax:

```
Set-Variable path2json ..\..\..\samples\remote-plugin-sample\registration\resources

.\extension-registration.bat  -action registerPlugin  -remote `
```

```
-url https://myvcenter/sdk `
-username administrator@vsphere.local  -password 'mysecret' `
-key com.vmware.sample.remote  -version 1.0.0 `
-pluginUrl https://mydevbox:8443/sample-ui/plugin.json `
-serverThumbprint 19:FD:2B:0E:62:5E:0E:10:FF:24:34:7A:81:F1:D5:33:`
19:A7:22:A0:DA:33:27:07:90:0F:8E:8D:72:F1:BD:F1 `
-serverCertificateFile C:\plugin-server.cer `
-c 'Example, Inc.'  -n 'Remote Plug-in'  -s 'This is a sample plug-in' `
--faultList $path2json\faultList.json --privilegeList $path2json\privilegeList.json `
--resourceList $path2json\resourceList.json --taskList $path2json\taskList.json
```

If a password contains special characters, use appropriate escape sequences for your shell.

> **NOTE**
> When registering the plug-in, it's highly recommended to provide the SSL certificate of the plug-in server in addition to the thumbprint of the SSL certificate of the plug-in server. Performing a full SSL certificate check during SSL handshake is more secure than performing an SSL certificate thumbprint check. Furthermore, in future vSphere releases, support for SSL certificate thumbprints will be dropped and full SSL certificates will be required.

The registration script displays a message that the plug-in is "successfully registered in vCenter."

An `Extension` record is added in the ExtensionManager of the vCenter Server instance.

In a web browser, connect to the vCenter Server URL and verify that the remote plug-in displays a Global View.

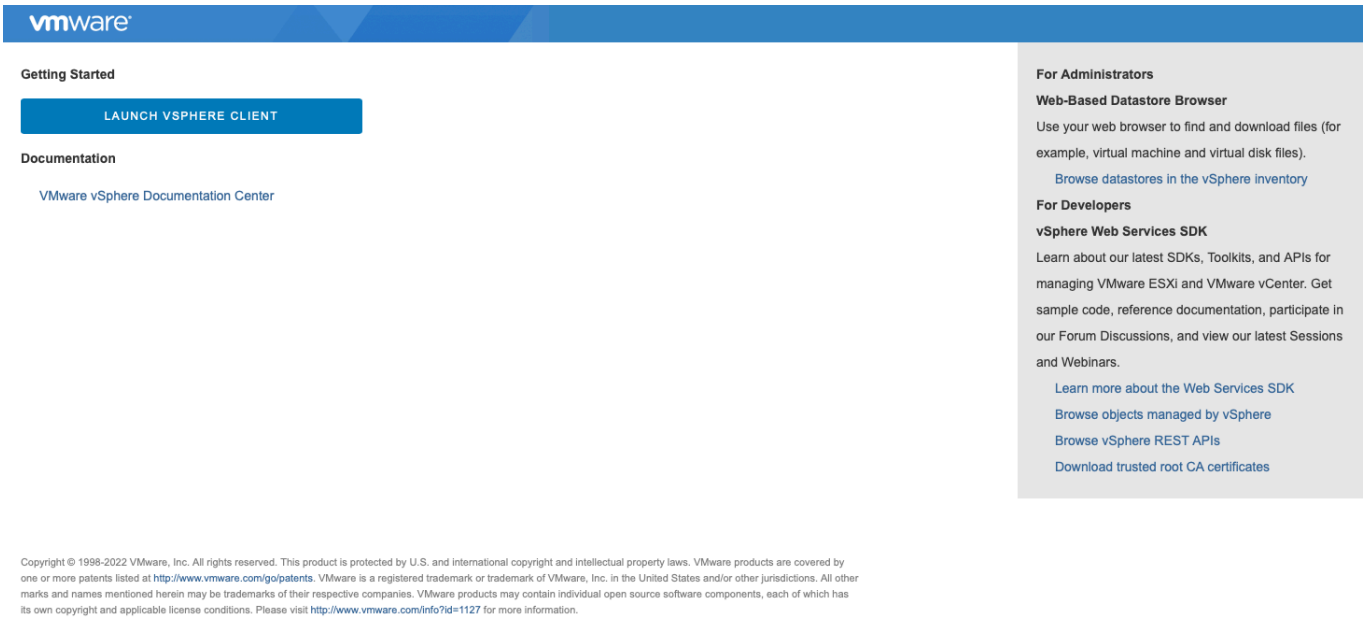
Viewing the vSphere Client Remote Plug-in Sample

After you build and register the remote plug-in sample, you can view it in the vSphere Client.

To view the remote plug-in sample, you must run the plug-in server and register it with the vCenter Server instance to which you connect your browser and run the vSphere Client.

To view the plug-in user interface, open a browser window and connect to the vCenter Server instance where you registered the plug-in. Use the fully qualified domain name of the vCenter Server to match the server certificate. From the welcome screen you can launch the vSphere Client.

Figure 118: vCenter Server Welcome Screen



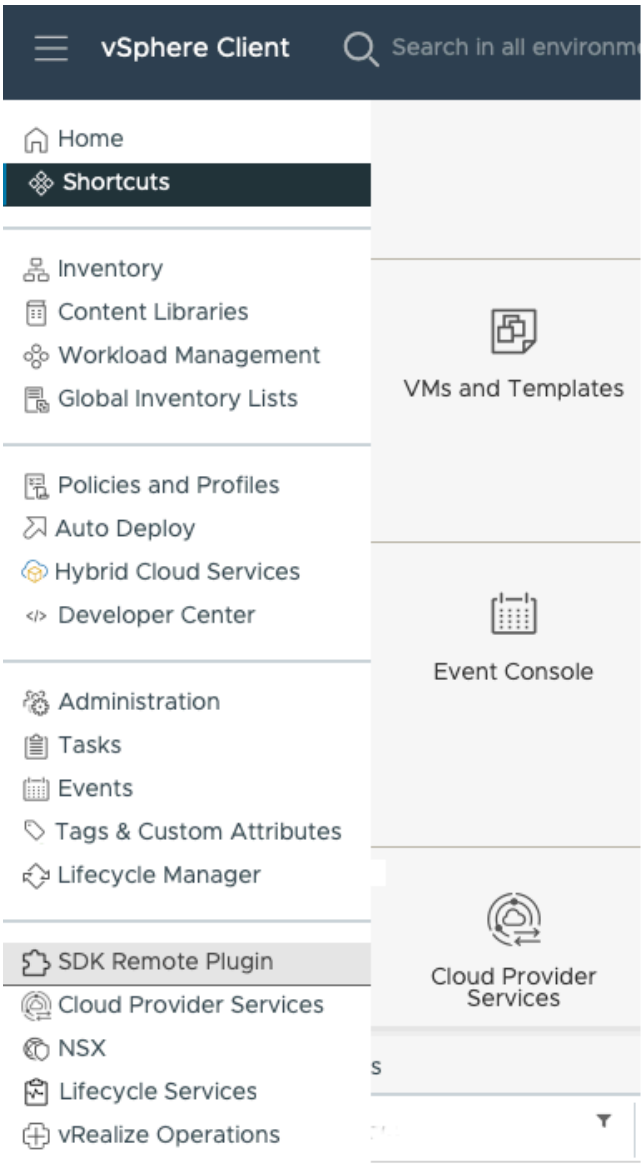
After you select **LAUNCH VSPHERE CLIENT** and authenticate with vCenter Server, the vSphere Client displays the home page with an **SDK Remote Plugin** link in the object navigator pane on the left. The link leads to the remote plug-in global view.

The plug-in global view includes the following subviews:

- A welcome page is the default page for the global view.
- A settings page allows you to change the number of items displayed in the Chassis List page.
- A Chassis List page displays summary information about the Chassis objects currently in the store. The Chassis store is initialized with several random Chassis objects for display. On the Chassis List page you can do the following actions:
  - Edit the Chassis object properties in a modal dialog.
  - Display related Host objects.
  - Display the Monitor subview or the Configure subview.

The code that supports these actions demonstrates how to use a modal dialog, how to create, delete, and update Chassis objects, and how to make calls to the plug-in server and the vCenter Server. The following illustrations show some of the features of the global view.

Figure 119: Selecting the Sample Plug-in Global View in the Navigation Pane



The Global View extension point is specified by the following lines from the manifest file, `plugin.json`:

```
"global": {
   "view": {
      "navigationId": "entrypoint",
      "uri": "index.html#/entry-point",
      "navigationVisible": false
   }
}
```

The Global View content is specified in the file `entry.point.component.html`.

Figure 120: Sample Plug-in Global View Welcome Page

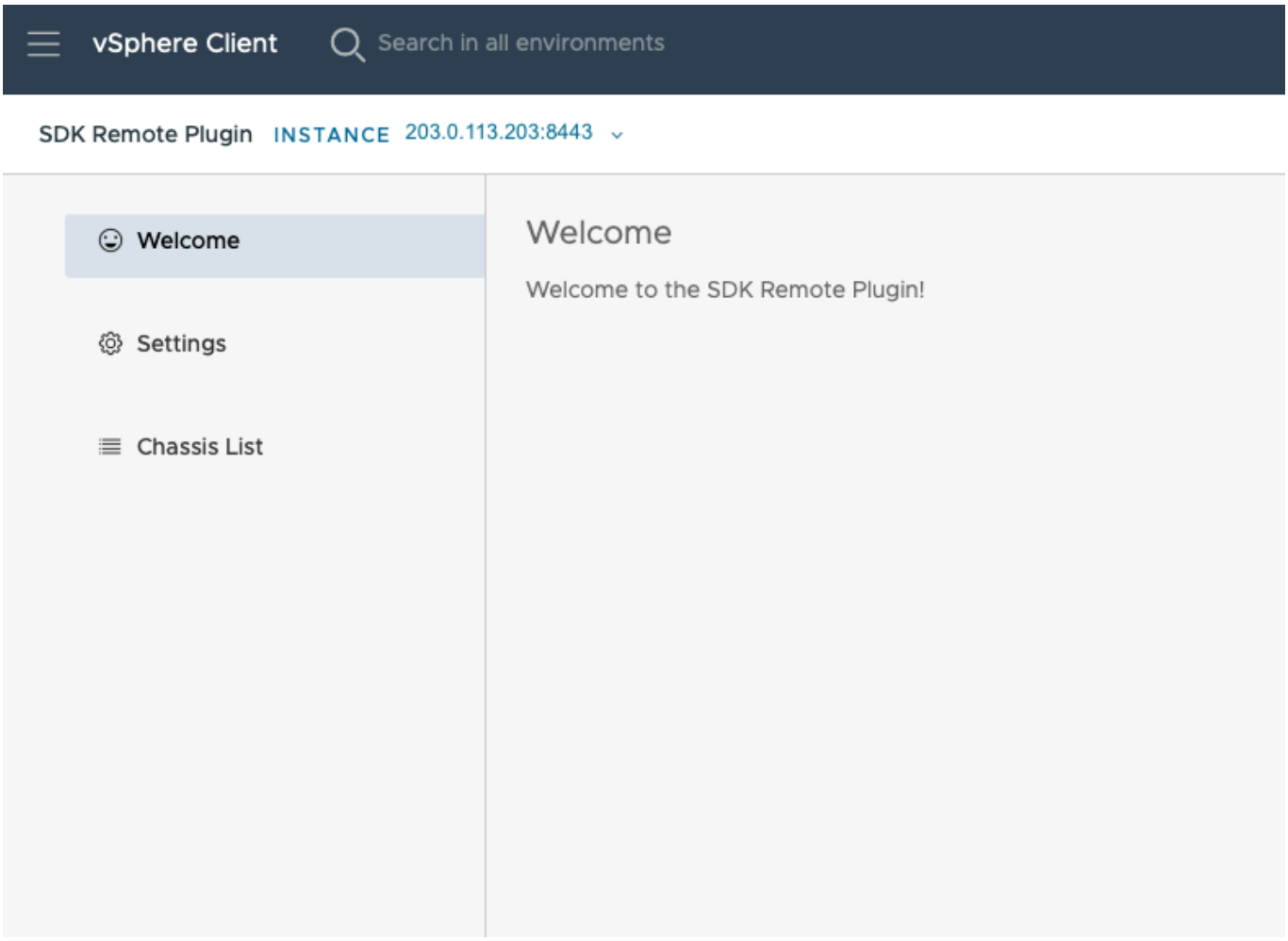


Figure 121: Sample Plug-in Global View Chassis List Page

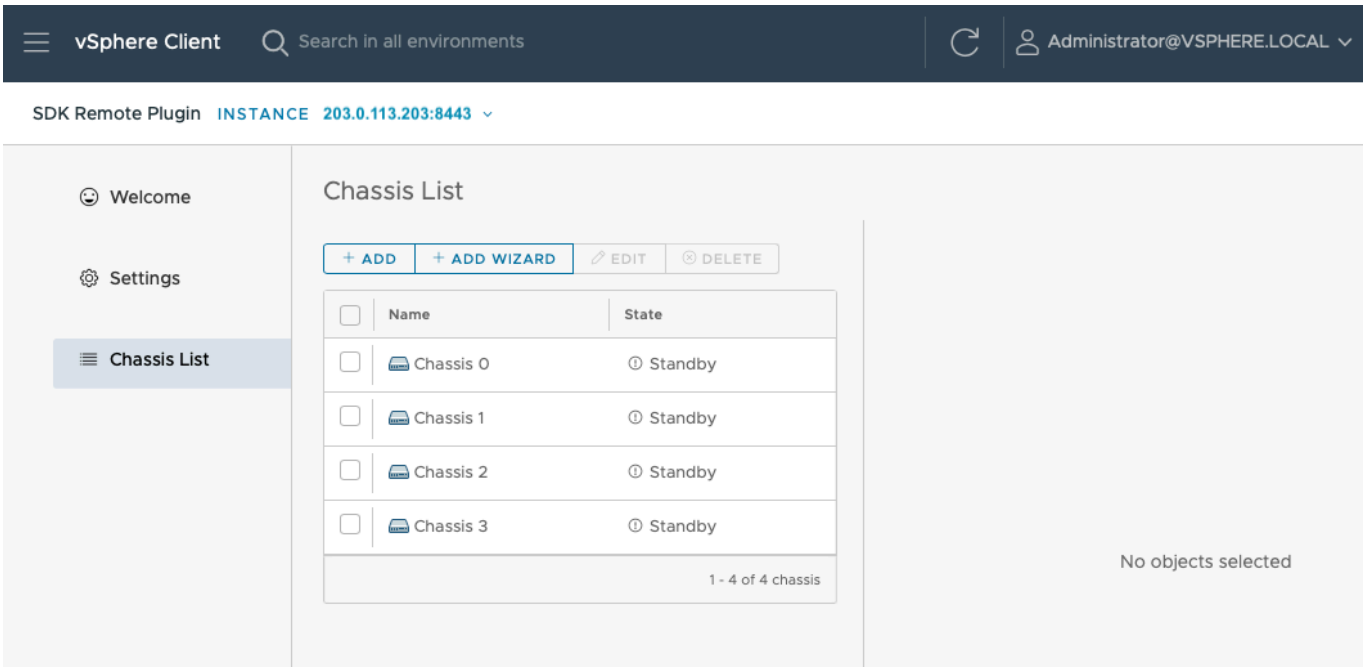


Figure 122: Sample Plug-in Edit Chassis Dialog

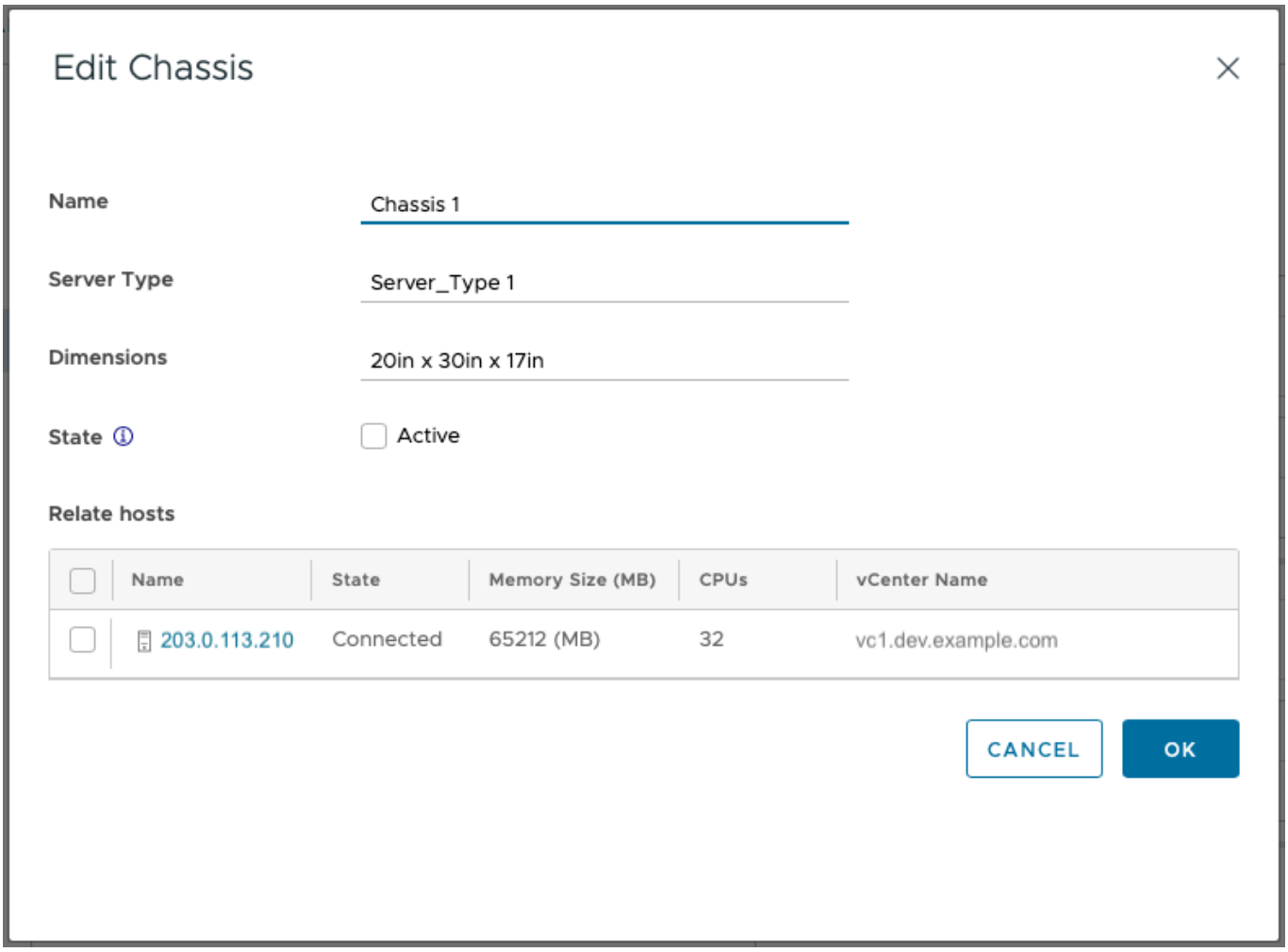
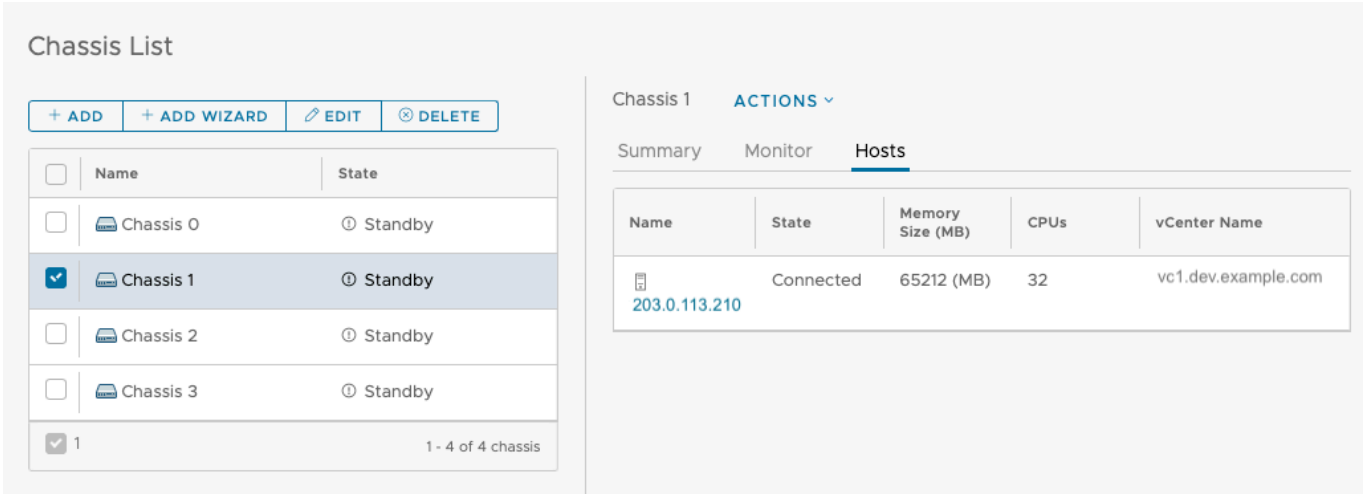
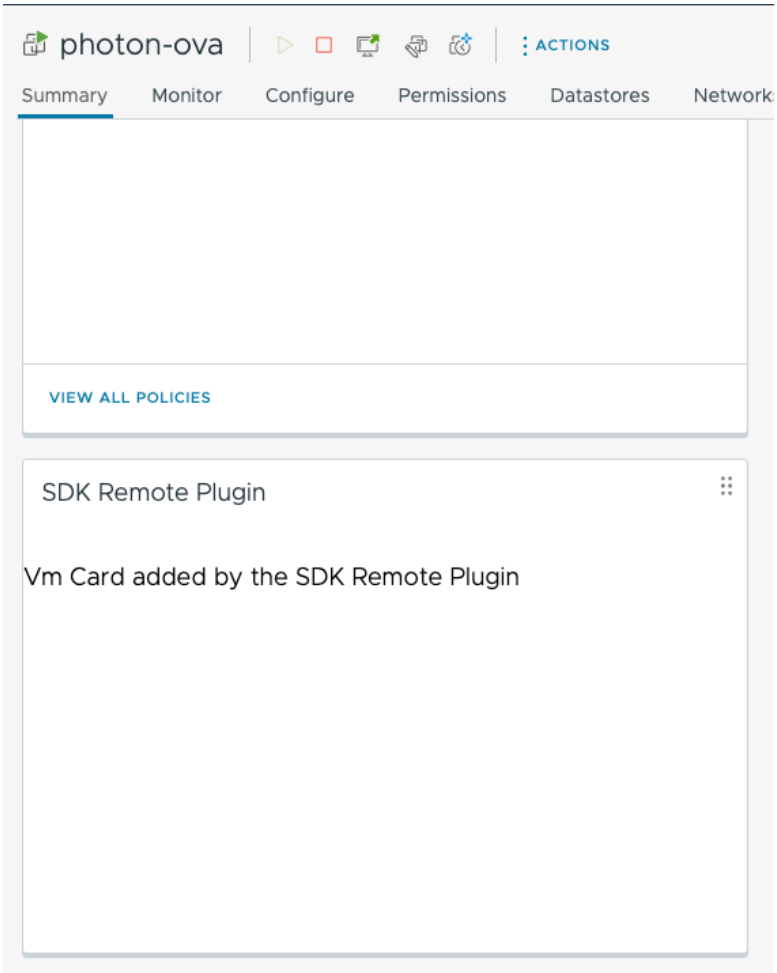


Figure 123: Sample Plug-in Chassis with Related Host



In addition to the global view and its subviews, the remote plug-in sample provides views that show how to extend the vSphere VirtualMachine object. It also shows the use of a card within the Summary view.

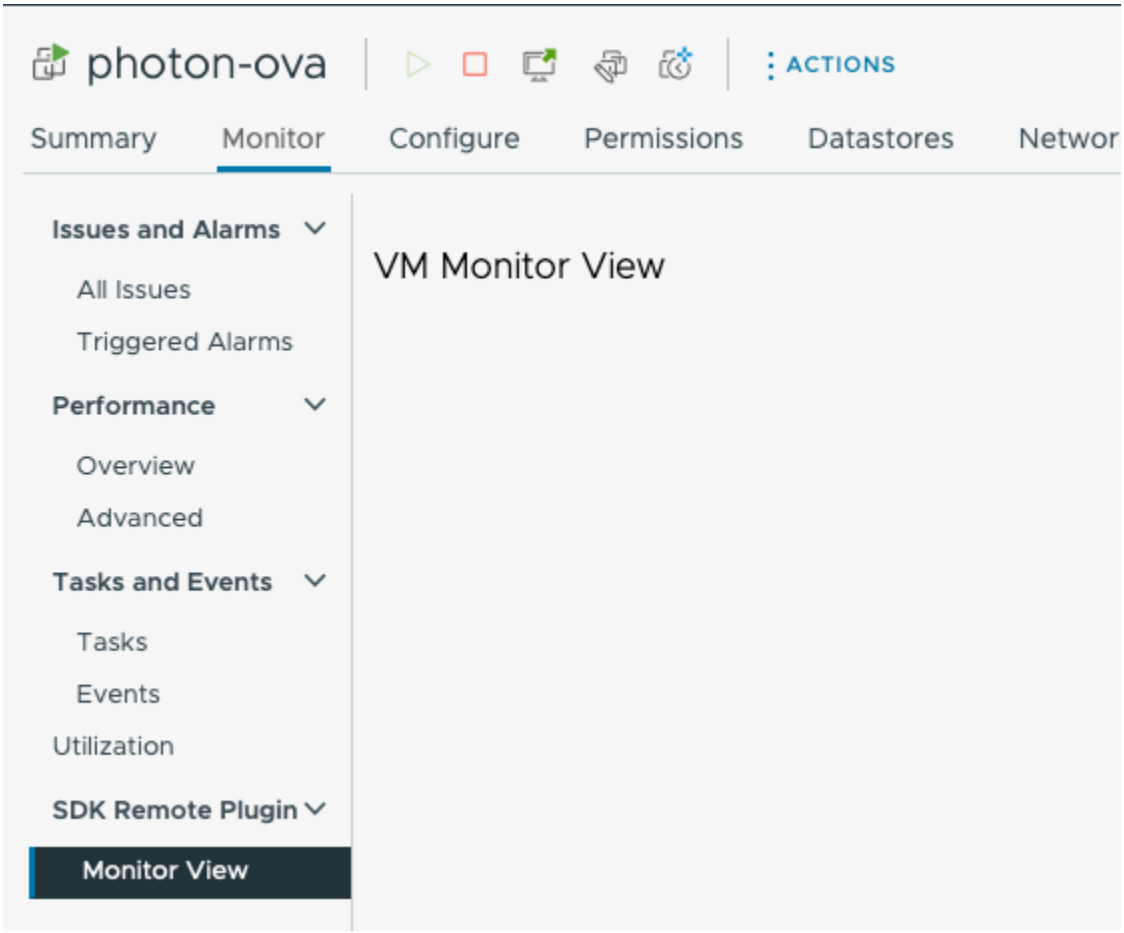
Figure 124: Card Extending the VirtualMachine Summary View



The VirtualMachine Summary View extension point is specified by the following lines from the manifest file, `plugin.json`:

```
"objects": {
   "VirtualMachine": {
      "summary": {
         "view": {
            "uri": "index.html#/vm-card",
         }
      },
      ...
   }
}
```

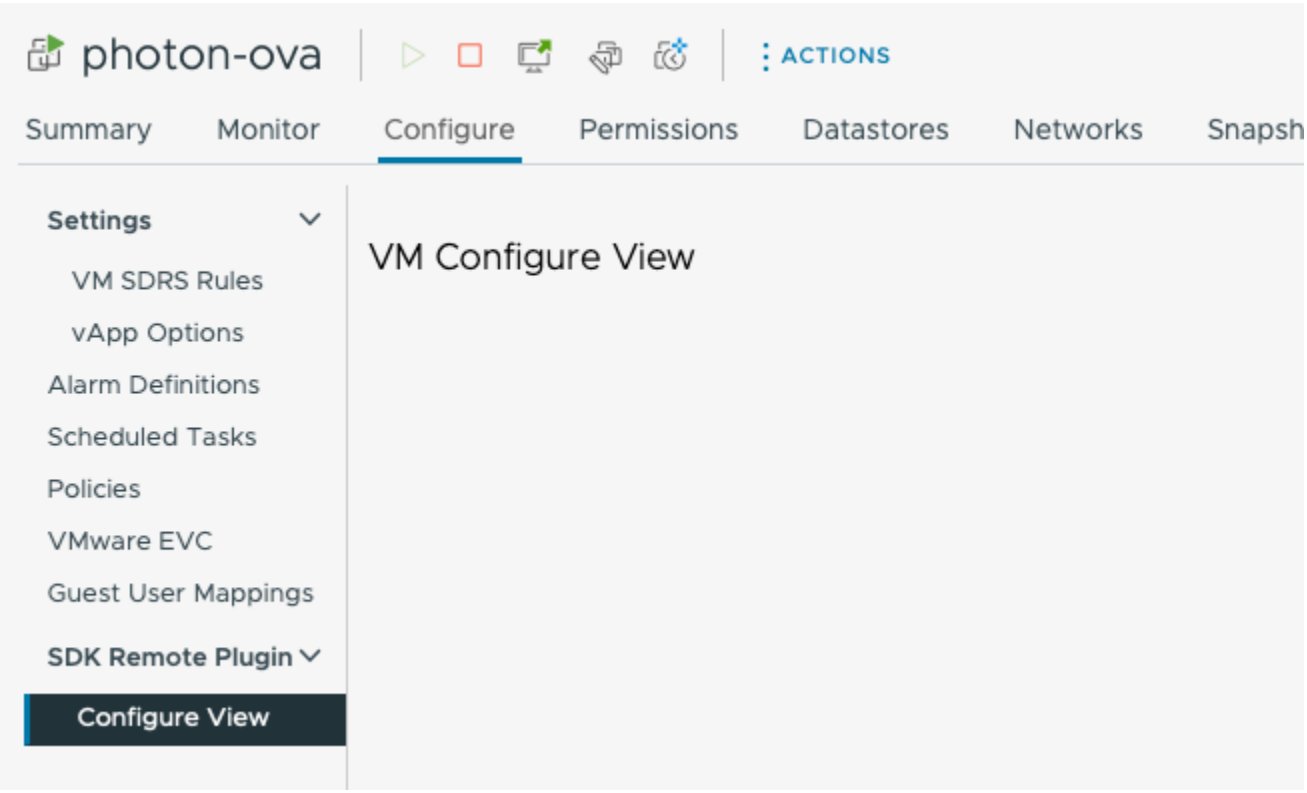
Figure 125: Extending the VirtualMachine Monitor View



The VirtualMachine Monitor View extension point is specified by the following lines from the manifest file, `plugin.json`:

```
"objects": {
   "VirtualMachine": {
      ...
      "monitor": {
         "views": {
            "navigationId": "vmMonitor",
            "labelKey": "vm.monitor.view.title",
            "uri": "index.html#/vm-monitor",
         }
      }
   }
}
```

Figure 126: Extending the VirtualMachine Configure View



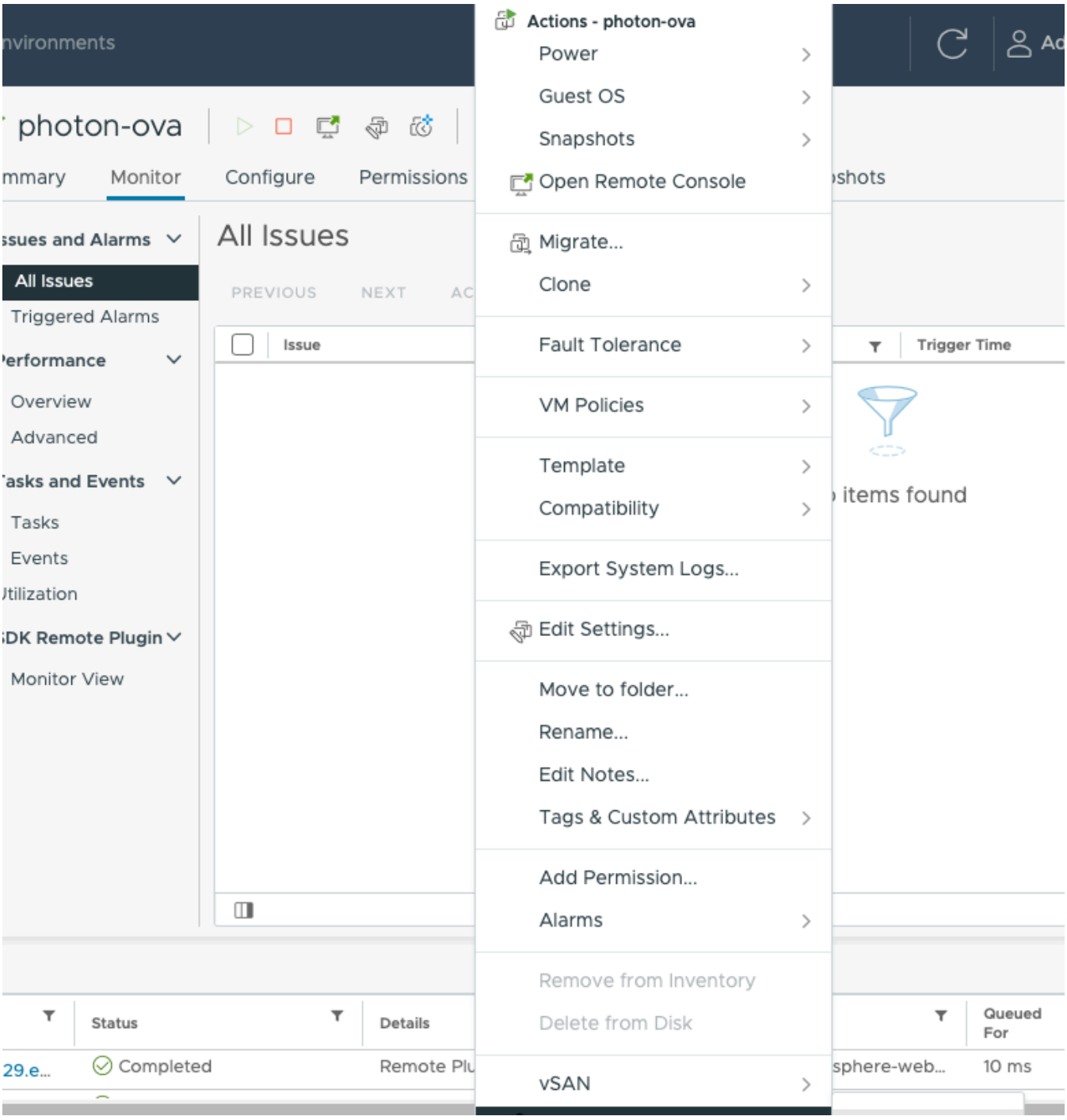
The VirtualMachine Configure View extension point is specified by the following lines from the manifest file, `plugin.json`:

```
"objects": {
   "VirtualMachine": {
      "configure": {
         "dynamicUri": "dynamicItems/vm/configure",
         "views": {
            "navigationId": "vmConfigureView",
            "labelKey": "vm.configure.view.title",
            "uri": "index.html#vm-congfigure",
            "dynamic": true
         }
      }
   }
}
```

NOTE
The Configure View is dynamic, which means that its appearance is conditional. For more information about dynamic views, see Dynamic Extensions for Remote Plug-ins.

The remote plug-in sample shows how to add an action item to a context menu for a VirtualMachine object. You can see how this appears in the following illustration.

Figure 127: Action Added to VirtualMachine Context Menu

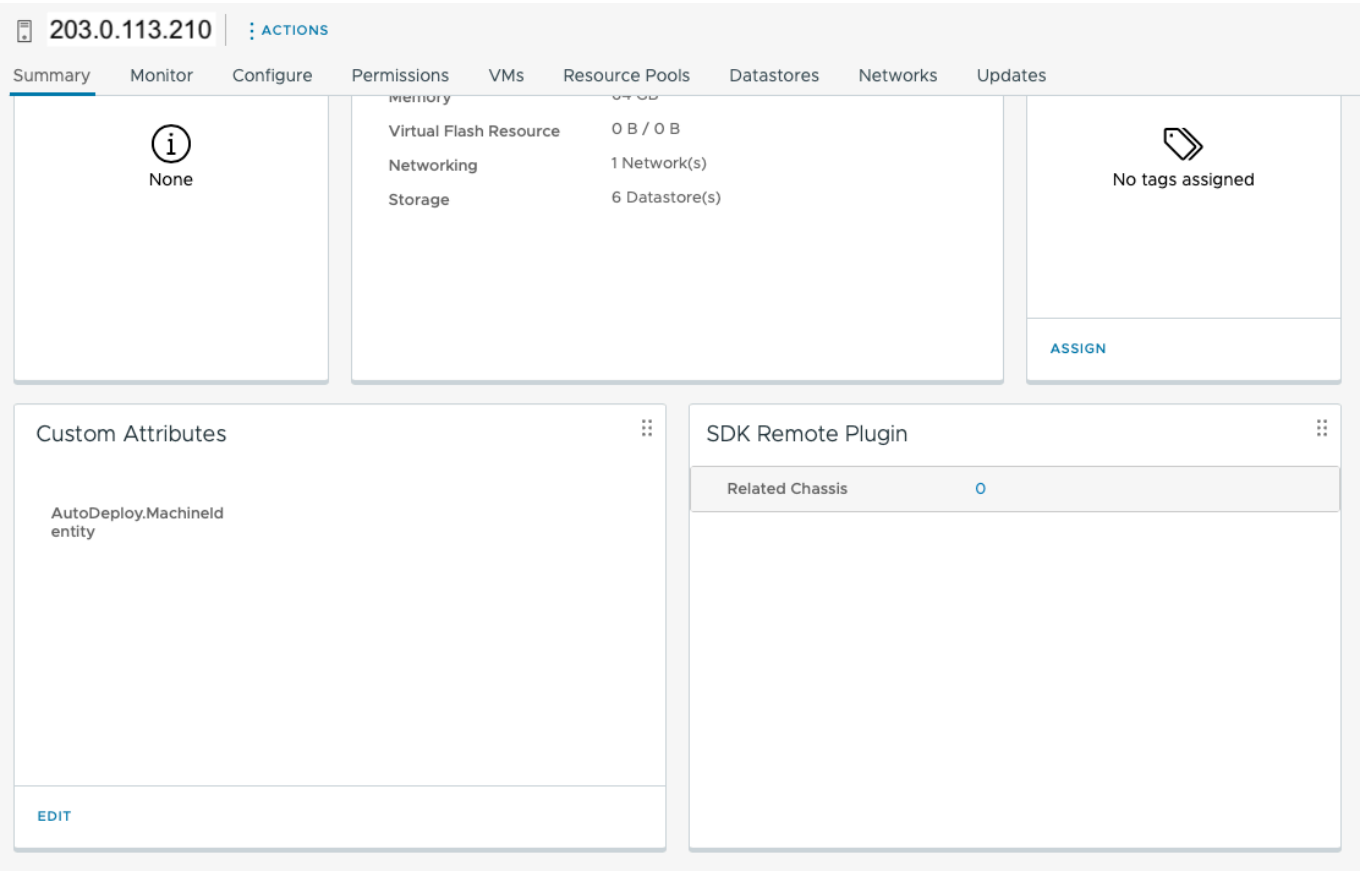


The VirtualMachine context menu extension point is specified by the following lines from the manifest file, `plugin.json`:

```
"objects": {
   "VirtualMachine": {
      "menu": {
         "dynamicUri": "dynamicItems/vm/actions",
         "actions": {
            "labelKey": "RemoteSample:vm.action.label",
            "icon": {
               "name": "action-vm"
            },
            "trigger": {
               "type"; "modal",
               "uri": "index.html#/vm-action-modal",
               "titleKey": "vm.action.model.title",
               "size": {
                  "width": 600,
                  "height": 205
               }
            },
            "acceptMultipleTargets": true,
            "dynamic": true
         }
      }
   }
}
```

The remote plug-in sample displays a HostSystem Summary view and a HostSystem Monitor view also. The Summary view shows the number of related Chassis objects. You click the number to reach the Monitor view. The Monitor view displays a datagrid listing all available Chassis objects. To create a relation between a HostSystem and a Chassis object, select the check box beside the Chassis object and click **Update**.

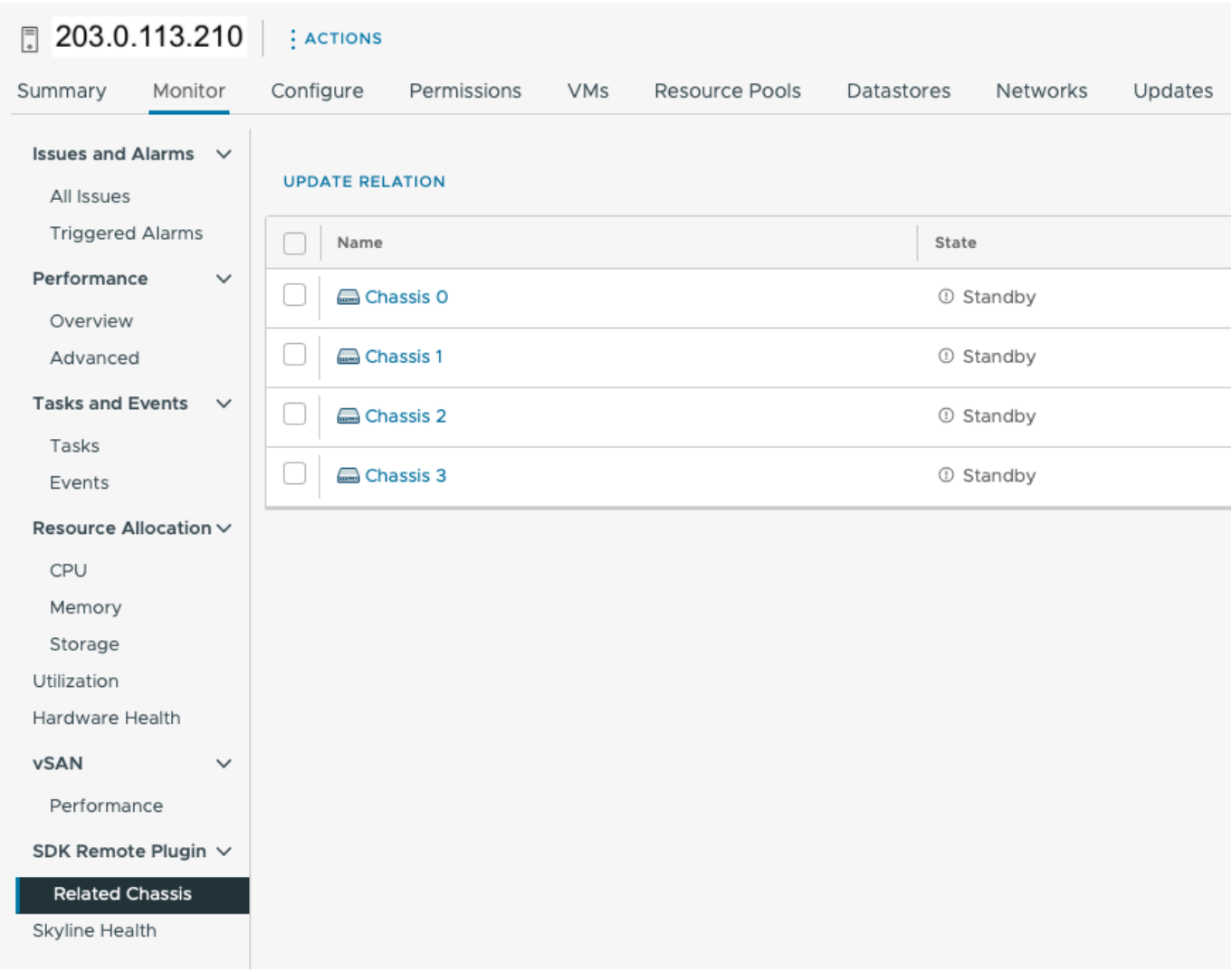
Figure 128: Card Extending the HostSystem Summary View



The HostSystem Summary View extension point is specified by the following lines from the manifest file, `plugin.json`:

```
"objects": {
   "HostSystem": {
      "summary": {
         "view": {
            "uri": "index.html#/host-card"
         }
      }
   }
}
```

Figure 129: Extending the HostSystem Monitor View



The HostSystem Monitor View extension point is specified by the following lines from the manifest file, `plugin.json`:

```
"objects": {
   "HostSystem": {
      "monitor": {
         "views": [
            {
               "navigationId": "hostMonitor",
               "labelKey": "host.monitor.view.title",
               "uri": "index.html#/host-monitor"
            }
         ]
```

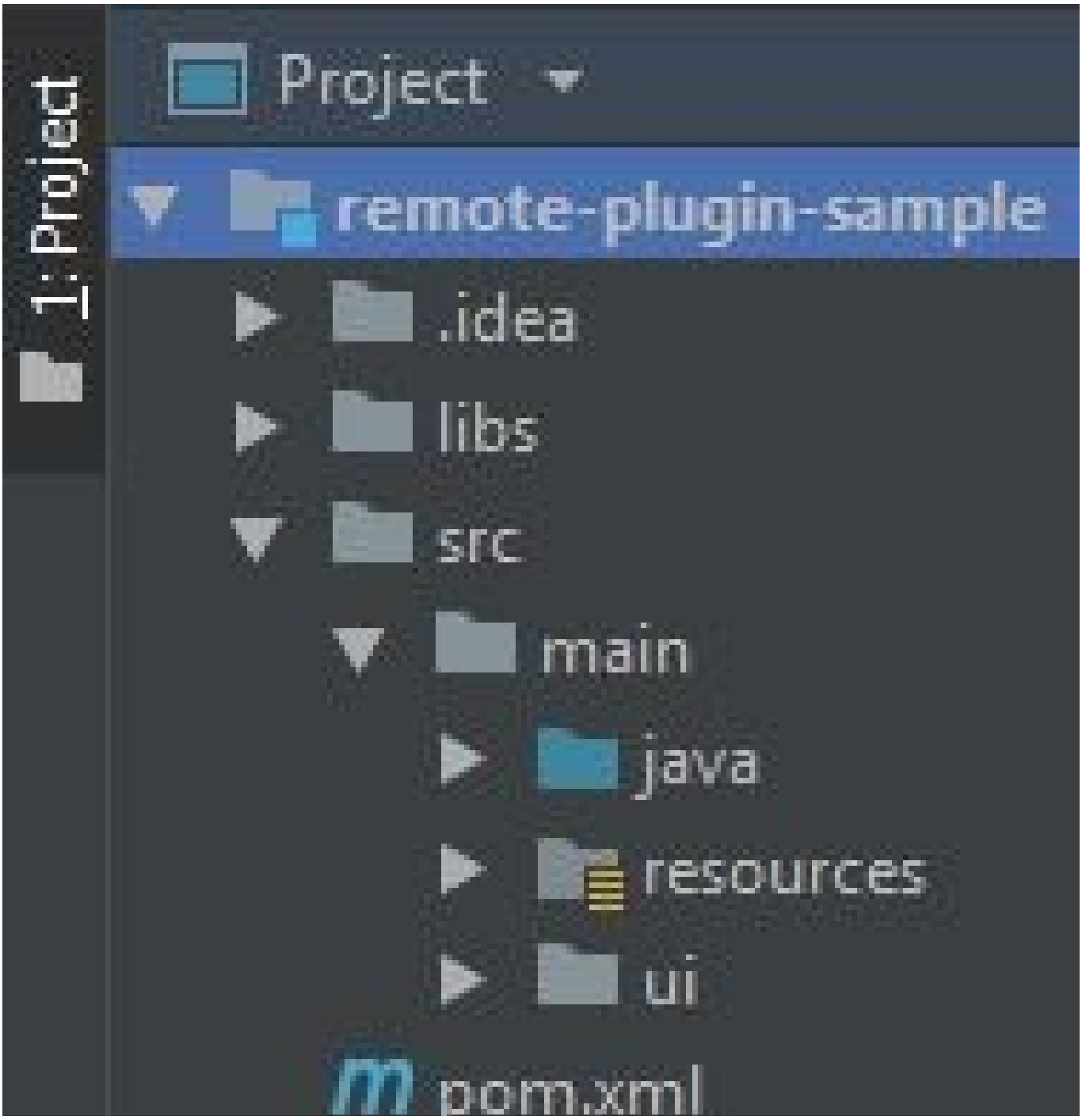
```
            }
        }
}
```

Remote Plug-in Sample Directory Structure

The remote plug-in sample code is organized as a Maven project. You can locate certain important files in the project as follows.

Top-Level Directory

Figure 130: Remote Plug-in Sample Top-Level Directory



The top-level directory, `remote-plugin-sample`, is located within the `html-client-sdk/samples` directory of the SDK zip file.

- The `libs` subdirectory contains the external library `vim25.jar`, which must be installed as a local dependency during the project build and deploy process. The library allows access to vSphere managed objects. In vSphere 8.0

U2 the `vim25.jar` library was deleted from the Client SDK, so developers who don't have it must find it in the vSphere Management SDK.
- Nested inside the `src` directory you can find a `java` directory, which contains Java source files for the sample plug-in server, at `java/com/vmware/sample/remote/*`.
- Also nested inside the `src` directory you can find a `ui` directory, which contains source code for the sample plug-in UI, at `ui/src/app/{model, service, views}/*`.
- To build the project, use the `pom.xml` file in the top-level directory of the remote plug-in sample, `html-client-sdk/samples/remote-plugin-sample/pom.xml`.

Remote Plug-in Architecture in the vSphere Client

The vSphere Client remote plug-in architecture is designed to integrate plug-in functionality into the vSphere Client without the need to run inside vCenter Server. This provides plug-in isolation and enables scale-out of plug-ins that operate in large vSphere environments. The remote plug-in architecture provides the following benefits:

- Your plug-in is protected from interference by unstable or compromised plug-ins loaded in the same vSphere Client.
- An incompatible plug-in does not interfere with vCenter Server operation.
- Plug-in compatibility is robust across vCenter Server upgrades.
- You can deploy a number of plug-in versions within the same vSphere environment. Your remote plug-in user interface needs to communicate with only a single back-end server.
- The topology of deployed plug-ins is well defined and easy to troubleshoot.

Components of the vSphere Client Architecture

The vSphere Client architecture enables administrators to manage vSphere environments of varying scale and complexity with a single user interface. It supports environments ranging from a single vCenter Server to a number of vCenter Server instances in Enhanced Linked Mode or Hybrid Linked Mode.

The user interface component runs in a browser, managing and displaying HTML5 views. It communicates with the vsphere-ui service, requesting HTML and Javascript files with vSphere inventory data, authenticating as needed. The user interface also manages a sandbox for each active plug-in and provides client library services to the plug-in user interface.

The vsphere-ui service is an OSGi Java application server that runs on every vCenter Server node. The vsphere-ui service communicates with all of the services provided by vCenter Server by using a variety of API styles and protocols. The vsphere-ui service maintains an authenticated session connection as a client of each of the services.

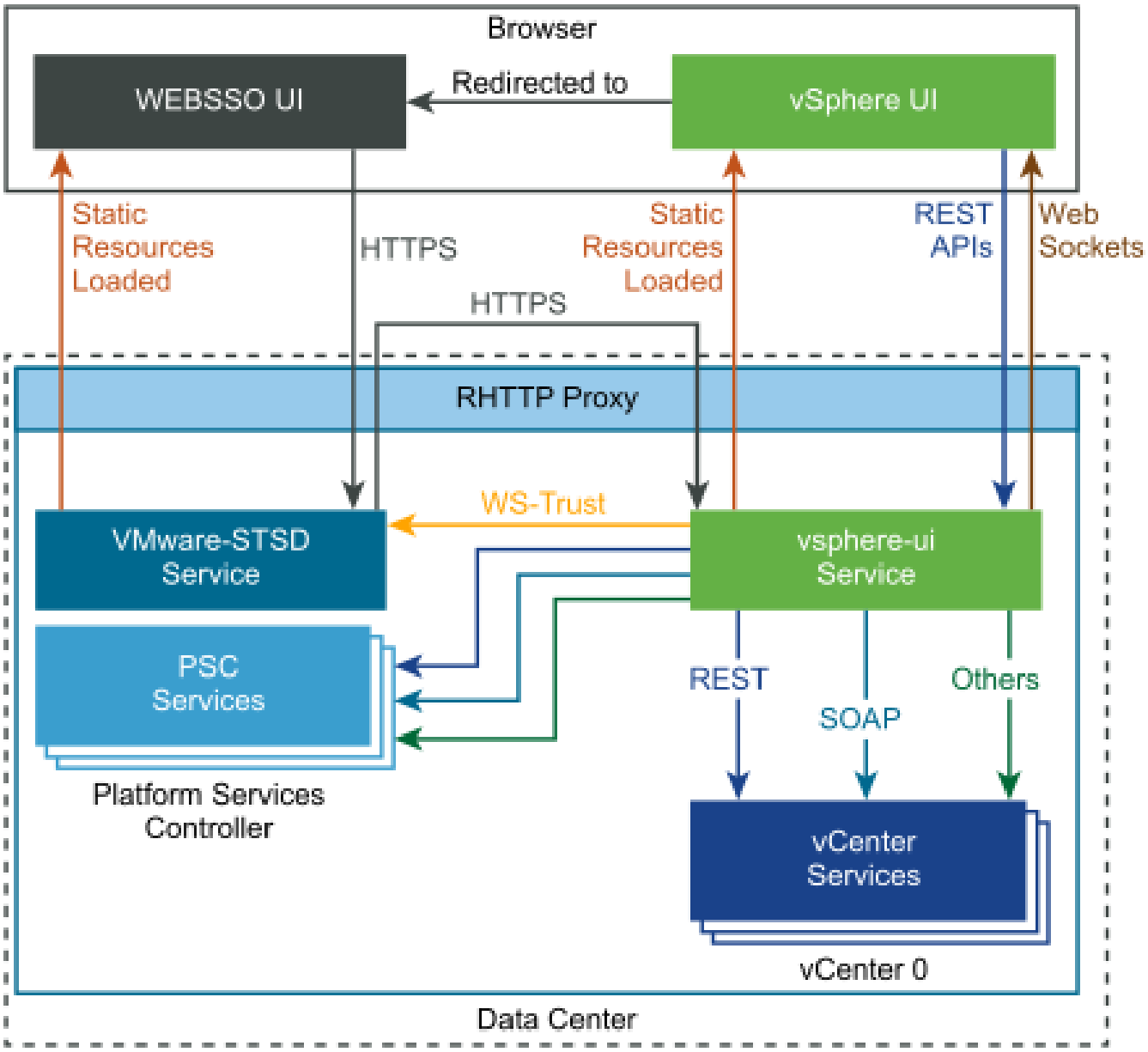
The vsphere-ui service also provides REST and Web Sockets APIs to the vSphere Client user interface running in the browser. The service supports authentication for users of the vSphere Client by redirecting the browser to a login user interface provided by the VMware vCenter Single Sign-on service.

vCenter Server Configurations

Remote plug-ins can operate in linked mode configurations of vCenter Server as well as in single vCenter Server instances. Illustrations in this chapter show the components and communication paths for both kinds of configuration.

The following illustration shows a single vCenter Server communicating with a web browser.

Figure 131: vCenter Server and Web Browser Communicatiions



In a linked mode configuration, the vsphere-ui service handles delegation of requests to the linked vCenter Server instances, using the same protocols that it uses when communicating with its own services..

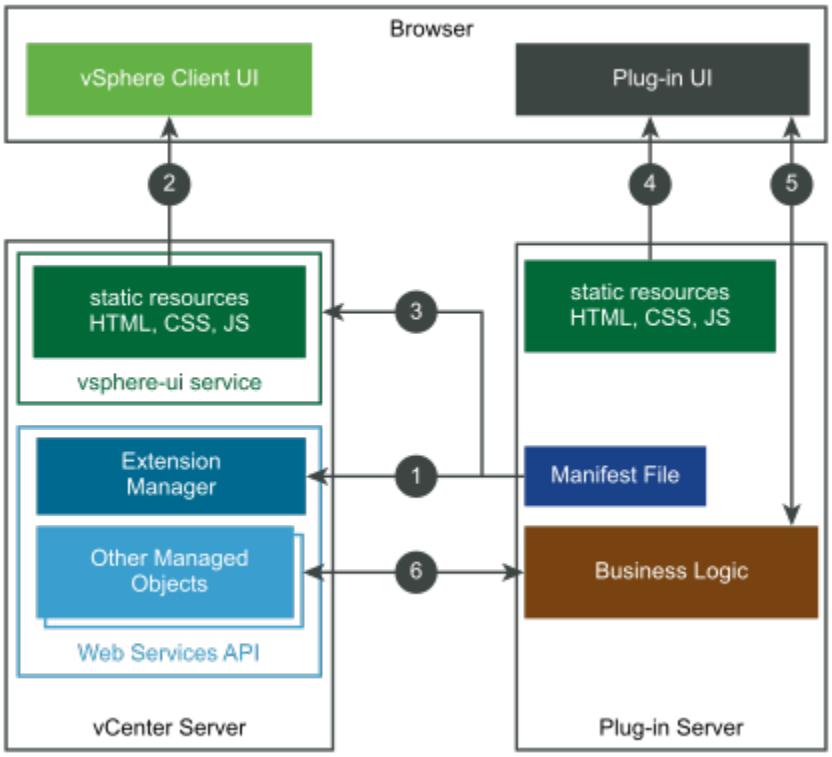
Communication Paths in Remote Plug-in Architecture

This diagram shows some of the communication paths between a plug-in, its user interface, and the vCenter Server to which the user interface is connected.

On the front end, the vsphere-ui service downloads and parses the plug-in manifest and serves UI components to the browser, including references to plug-in components, which are served by the plug-in server. These paths use simple HTTPS communications.

The back-end communication paths between the plug-in server and the vCenter Server Web Services API use SOAP messages over HTTPS. These communications are described in more detail in Communication Paths for Authentication in the Remote Plug-in Server.

Figure 132: A Simplified View of Remote Plug-in Architecture



The circled numbers identify the following data paths:

1. The remote plug-in installer registers the plug-in manifest file with the vCenter Server Extension Manager, by using the Web Services API.

2. A web browser downloads user interface elements of the vSphere Client from the vsphere-ui service in vCenter Server.

3. The vsphere-ui service downloads and parses the plug-in manifest file to determine where the plug-in extends the user interface.

4. The browser downloads user interface elements of the plug-in from the plug-in server.

5. The plug-in user interface requests data from the plug-in server.

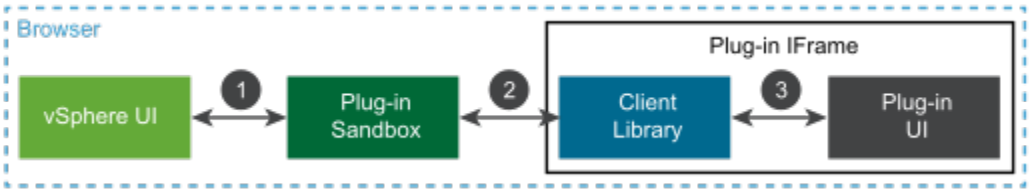
6. The plug-in server uses the Web Services API to interact with vCenter Server.

Communications Among UI Components in the vSphere Client

The vSphere Client user interface loads both its own components and the components belonging to the plug-in user interface. The vsphere-ui service reads the plug-in manifest file to determine where it should insert plug-in components in the user interface.

The user interface components loaded in the browser are organized as shown in the following diagram.

Figure 133: User Interface with a Sandboxed Plug-in



Paths:

1. Internal JavaScript methods
2. `window.postMessage()` method in browser
3. Public JavaScript methods

The plug-in user interface operates within its own iFrame, isolated from other plug-ins. The plug-in loads a copy of the vSphere Client JavaScript API client library, which is its sole connection to other client code. The plug-in code communicates with the client library code using JavaScript method calls.

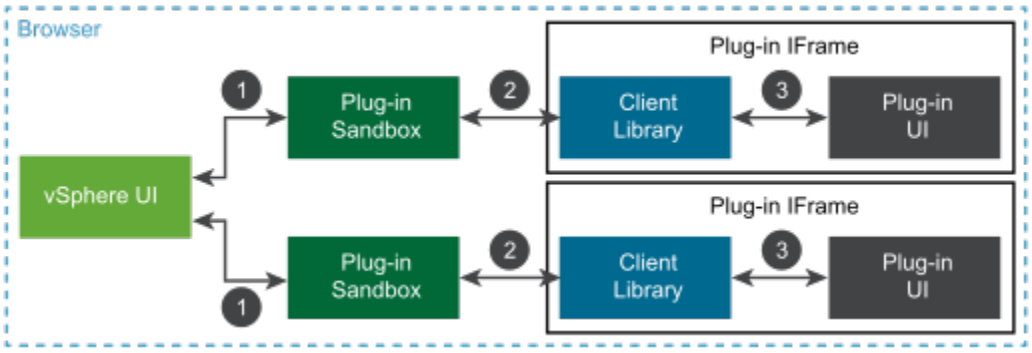
The client library communicates with the sandbox component that the vSphere Client provides to interface with the plug-in UI components. The communication with the sandbox uses the browser's `window.postMessage()` API. This makes it possible for the iFrame and its parent window to be loaded from different origins.

> **NOTE**
> In the vSphere 6.7 Update 1 release of the vSphere Client, the iFrame and its parent window share the same origin. Do not depend on this to remain the same in future releases. In cloud environments, the iFrame is loaded from a secondary host.

The plug-in sandbox communicates with other components of the vSphere Client user interface by using internal Javascript APIs.

If the vSphere Client has more than one plug-in active, each plug-in is allocated its own sandbox, and operates within its own iFrame, as shown in the following illustration.

Figure 134: User Interface with Two Plug-ins in Separate Sandboxes



Paths:

1. Internal JavaScript methods
2. `window.postMessage()` method in browser
3. Public JavaScript methods

In this case, each plug-in UI communicates with its own back-end server.

Client-Server Communications with Remote Plug-ins

Client requests to back-end services are handled in a similar way for plug-in user interface components and vSphere Client user interface components.

A plug-in sandbox runs outside the plug-in iFrame. The sandbox component, along with the rest of the vSphere Client user interface components, sends requests to the vsphere-ui service of the vCenter Server instance that the browser connected to. Service requests use REST and Web sockets over HTTPS. All requests pass through the vCenter Server reverse proxy, which routes them to the correct server components.

A plug-in user interface sends service requests to the plug-in back end, using any RPC style on top of the HTTPS transport protocol. All requests pass through the vCenter Server reverse proxy, which routes them to the plug-in server. The proxy routing is configured at the time the plug-in is installed; it simplifies dealing with self-signed certificates, and avoids problems with cross-origin resource access.

Security Concepts for Remote Plug-ins

Remote plug-ins typically use the HTTP protocol as a transport for requests, whether using REST or SOAP requests. Authentication methods vary, depending on the target endpoint.

Client-side sessions with REST endpoints are tracked with a session token passed in a custom HTTP header named `webClientSessionId`. Server-side sessions with SOAP endpoints are tracked with a cookie-based session ID.

A plug-in developer can choose what form of authentication suits the plug-in server component. A best practice is to authenticate by using the session token that the plug-in user interface can get from the client library.

To use the client-side session token, the plug-in server sends the token to a specific REST endpoint of the vsphere-ui service. The vsphere-ui service verifies the authenticating token, and then returns a session clone ticket. The plug-in server uses the clone ticket with the vSphere Web Services API to obtain a SOAP session. The authentication process is described in more detail in Remote Plug-in Server Components for the vSphere Client.

Creating a Remote Plug-in for the vSphere Client

This chapter outlines how to create your own remote plug-in for the vSphere Client. The goal is to illustrate the entire process using simple components. Other chapters deal with some of the steps in more detail.

To create a remote plug-in, you need both front-end components and back-end components. The front-end components constitute a user interface based on HTML 5. The back-end components include a web server and business logic in support of the front end.

For the front end of your plug-in, you can choose any JavaScript-compatible language. The Remote Plug-in Sample in the vSphere Client SDK uses Angular and Clarity. A best practice is to use Clarity elements to harmonize with the look and feel of the vSphere Client.

For the back end, you can choose any language and any web server. The SDK samples are written in Java and built with Maven, but there is no restriction on the language or tools that you use for back-end development. The SDK samples use Tomcat as a web server because it is bundled with Spring.

Code Components for a vSphere Client Remote Plug-in

A remote plug-in for the vSphere Client includes several code components that you create.

To create code for a complete plug-in, you must do the following:

- Choose the extension points where your user interface views will extend the vSphere Client.
- Create front-end views that provide the user interfaces for data access.
- Create back-end controllers that interface between services and user interface views.
- Create models for your logical data objects.

- Create back-end services that translate between data models and the objects in storage that back the models.
- Import `vim25.jar`, which as of vSphere 8.0 U2 is available only in the vSphere Management SDK, to access the Web Services SDK. The `vim25` library provides RPC access to the PropertyCollector, used to retrieve properties of vCenter Server managed objects.

Deployment Requirements for a vSphere Client Remote Plug-in

To prepare a remote plug-in for deployment, you must prepare at least one file and launch one or more processes. The simplest plug-in can be implemented with a single back-end server and a plug-in manifest file that describes the extension views and other details about the plug-in.

Deployment of a remote plug-in takes place at run time, but you must make preparations in advance. You need to do the following to prepare for plug-in deployment:

- Run a web server that provides plug-in components on demand.
- Run your plug-in server binary. The plug-in server runs on a virtual or physical machine of your choice, but it must be the same machine as the web server.
- Prepare a plug-in manifest file, `plugin.json`, that specifies the plug-in components. See also . The manifest file must be accessible by HTTPS, on the same machine as the web server.
- Register your plug-in with a vCenter Server instance. You can register either by using the registration script in the SDK or by writing your own registration tool. In either case, you need the URL and credentials to access the vCenter Server instance. You also need the security certificate or thumbprint of your plug-in manifest server, and the URL of your plug-in manifest file. The connection should be secure HTTP (HTTPS).

  NOTE
  For production installations, a best practice is to create a registration tool that does not expose vCenter Server credentials on the command line.

  NOTE
  When registering the plug-in, it's highly recommended to provide the SSL certificate of the plug-in server in addition to the thumbprint of the SSL certificate of the plug-in server. Performing a full SSL certificate check during SSL handshake is more secure than performing an SSL certificate thumbprint check. Furthermore, in future vSphere releases, support for SSL certificate thumbprints will be dropped and full SSL certificates will be required.

To register a plug-in by using the script in the SDK, you use the `registerPlugin` function of the `extension-registration` script. The script arguments are demonstrated in the following example.

```
./extension-registration.sh  -action registerPlugin  -remote \

-url https://myvcenter/sdk \

-username administrator@vsphere.local  -password mysecret \

-key com.mycompany.myplugin  -version 1.0.0 \

-pluginUrl https://mydevbox:8443/myplugin/plugin.json \

-serverThumbprint 19:FD:2B:0E:62:5E:0E:10:FF:24:34:7A:81:F1:D5:33:\

19:A7:22:A0:DA:33:27:07:90:0F:8E:8D:72:F1:BD:F1 \

-serverCertificateFile /tmp/plugin-server.cer \

-vCenterServerThumbprint 2A:0E:3C:1F:73:6F:1F:21:00:35:45:8B:92:02:E6:44:\

2A:B8:33:B1:EB:44:38:18:A1:10:9F:9E:83:02:CE:02 \
```

```
-c 'Example, Inc.'  -n 'Remote Plug-in'  -s 'This is a remote plug-in'
```

To register a plug-in by means of your own tool, you must create an `Extension` type record in the `ExtensionManager` managed object. The `Extension` record must describe the plug-in manifest server in two places: `client[0]` and `server[0]`. These two array entries are similar in several properties, but have important differences:

- The `client` array, which has type `ExtensionClientInfo[]`, contains the plug-in version in the first element. This is required so that the vSphere Client can manage plug-ins correctly.
- The `client` array must have a `type` property with a value of `vsphere-client-remote` in the first element.
- The `server` array, which has type `ExtensionServerInfo[]`, must have a `type` property with any value you choose in the first element. A best practice is to assign the value `MANIFEST_SERVER` to the first element of the array, to identify the manifest server for the plug-in.
- To support HTTPS connections, the `server` array must have either `serverThumbprint` or `serverCertificateFile` or both in the first element. Property `serverThumbprint` takes the value of the SHA-256 hash of the server certificate, while `serverCertificateFile` takes the value of the PEM-encoded server certificate (string).

  NOTE
  HTTP connections are not recommended for production use with the vSphere Client.

Both the `client` array and the `server` array must have the same value in the `url` property of the first array element. The value must be the URL of the plug-in manifest server.

For information about the plug-in manifest, see Sample Manifest File for a Remote Plug-in. For information about auxiliary plug-in processes, see Using Auxiliary Plug-in Servers. For information about the plug-in registration script, see vSphere Client Plug-in Registration Tool.

Using Auxiliary Plug-in Servers

A remote plug-in package must include a primary server that serves the plug-in manifest file. A plug-in can also include auxiliary servers that serve other plug-in resources.

Plug-in packages must be downloaded with secure HTTP (HTTPS) protocol.

You might want to design a plug-in with auxiliary servers for several reasons:

- To provide load balancing
- To separate performance tiers
- To manage security risks
- To localize database access
- To take advantage of different coding languages or frameworks
- To facilitate re-implementation

For more information about auxiliary servers, see Advanced Considerations for Remote Plug-in Servers.

vSphere Client Plug-in Registration Tool

The vSphere Client SDK contains a plug-in registration tool that manages plug-in extension registration records in the vCenter Server ExtensionManager. The tool registers, unregisters, and updates the registration record of a plug-in.

Plug-in Registration Script

To register a plug-in, use this registration script in the SDK. It's a wrapper for a Java tool, which opens a session with a vCenter Server instance using the VMware Web Services API.

```
tools/vCenter plugin registration/prebuilt/extension-registration.sh
```

Plug-in Registration Script Syntax

The arguments of the registration script function as follows:

- `-action` (required) can be one of:
  - `registerPlugin`
  - `updatePlugin`
  - `unregisterPlugin`
  - `isPluginRegistered`
- `-c` or `-company` is the name of the plug-in vendor.
- `-insecure` bypasses security checks on the vCenter Server certificate. This is not recommended for a production environment.
- `-k` or `-key` (required) is an identification string for the plug-in. The plug-in registration record in the vCenter Server ExtensionManager contains this identification.
- `-local` (default) is used to register or update a local plug-in. See also `-remote`.
- `-n` or `-name` is a user-friendly identification string for the plug-in.
- `-p` or `-password` (required) authenticates the vCenter Server user account. See also `-username`.
- `-ps` or `-pluginServers` is a string that contains a JSON array of objects that specify endpoints for auxiliary services belonging to the plug-in. Each endpoint object must contain a `url` property to enable connections to the endpoint. The scheme must specify HTTP or HTTPS as the protocol. To support HTTPS connections to an endpoint, at least one of the fields `serverThumbprint` or `serverCertificateFile` must be set in the endpoint object. HTTP connections are not recommended for production use with the vSphere Client. The `type` property, required for auxiliary servers, is a service registration identifier string by which the plug-in front end can discover a needed endpoint.

  Choice and optional properties that can also be present in the object are:

  - `serverThumbprint` is the thumbprint (fingerprint) of the auxiliary server's SSL certificate. See constraints for the `-st` argument below.
  - `serverCertificateFile` is the path of the auxiliary server's SSL certificate on the local filesystem. See constraints for the `-scf` argument below. The certificate of the auxiliary server must contain the host name specified according to subjectAltName extension (RFC 2818, Section 3.1 Server Identity).
  - `label` (not currently used)
  - `summary` is a brief description of the auxiliary server.
  - `company` is the name of the plug-in server vendor.
  - `adminEmail`
- `-pu` or `-pluginUrl` (required to register a plug-in) is the URL of the plug-in manifest served by the plug-in back end. The scheme must specify HTTP or HTTPS as the protocol. The path segment of the `-pluginUrl` must be specified relative to the directory where you run the plug-in server. **
- `-remote` (required for a remote plug-in) is used to register or update a remote plug-in. See also `-local`.
- `-s` or `-summary` is a brief description of the plug-in.
- `-show` or `-showInSolutionManager` specifies that the plug-in will appear in the Solutions list of the Administration panel.
- `-st` or `-serverThumbprint` is the SHA-256 signature of the plug-in back-end server certificate. Character pairs should be separated by colon separators.
- `-scf` or `-serverCertificateFile` is the path on the local filesystem of the plug-in manifest server's SSL certificate file, in PEM or DER format. The certificate of a plug-in server record must contain the host name of the plug-in server in its `subjectAltName` extension as per RFC 2818, Section 3.1 Server Identity.
- `-u` or `-username` (required) identifies a vCenter Server user account that has permission to write to the vCenter Server ExtensionManager. See also `-password`.

- `-url` (required) is the URL of the `/sdk` resource of the vCenter Server. Use the fully qualified domain name of the vCenter Server instance. For example: `https://my-vcsa.example.com/sdk`
- `-v` or `-version` (required) identifies the plug-in version.
- `-vct` or `--vcenterServerThumbprint` supplies the certificate thumbprint for the vCenter Server instance. You can use this in development environments when the certificate is self-signed or otherwise not recognized by the browser.
- `-eventList` - path to the event list JSON file, relative to the file system root of the manifest server. The file should contain a JSON formatted array of event infos. Each object in the array must specify the *`eventId`* of the event. In addition, an optional XML descriptor for the `EventType` can be specified.
- `-faultList` - path to the fault list JSON file, relative to the file system root of the manifest server. The file should contain a JSON formatted array of fault infos. Each object in the array must specify the *`faultId`* of the fault.
- `-privilegeList` - path to the privilege list JSON file, relative to the file system root of the manifest server. The file should contain a JSON formatted array of privilege group objects. Each object in the array must specify the *`groupId`* of the privilege group and the privileges in that group. Each object in the `privileges` array must specify an *`privilegeId`* of the privilege.
- `-resourceList` - path to the resource list JSON file, relative to the file system root of the manifest server. The file should contain a JSON formatted object, where the keys are locales (for example 'en', 'fr', 'de' ..) and the values are objects with a key to be localized and the value is the localizable message.
- `-taskList` - path to the tasks list JSON file, relative to the file system root of the manifest server. The file should contain a JSON formatted array of task infos. Each object in the array must specify the *`taskId`* of the task.

** To support HTTPS connections to the plug-in manifest server, at least one of the fields `serverThumbprint` or `serverCertificateFile` must be set in the endpoint object. Insecure HTTP connections are not recommended for production use with the vSphere Client.

> **NOTE**
> When registering a plug-in, it's highly recommended to provide SSL certificates of the plug-in servers in addition to the thumbprints of the SSL certificates of the plug-in servers. Performing a full SSL certificate check during SSL handshake is more secure than performing an SSL certificate thumbprint check. Furthermore, in future vSphere releases, support for SSL certificate thumbprints will be dropped and full SSL certificates will be required.

Sample Manifest File for a Remote Plug-in

The following sample manifest file contains the minimum elements required for a simple plug-in.

```
{
   "manifestVersion" : "1.0.0",
   "requirements": {
      "plugin.api.version": "1.0.0"
   },
   "configuration": {
      "nameKey": "plugin.name",
      "icon": {
         "name": "main"
      }
```

```
   },
   "definitions": {
      "i18n": {
         "locales": ["en-US"],
         "definitions": {
            "plugin.name": {
               "en-US": "Hello World"
            }
         }
      }
   },
   "global": {
      "view": {
         "uri": "index.html"
      }
   }
}
```

NOTE
The URIs specified in the manifest file are relative to the location of the manifest file itself. That is, the directory containing `plugin.json` should be considered the server root directory for plug-in resources.

Deploying Remote Plug-ins for the vSphere Client

A remote plug-in must be deployed by the vSphere Client before it can be used. Deployment is the process of preparing components of vCenter Server to accept communications from the plug-in and to display plug-in views in the browser. Developers have a choice of deployment topologies, as explained later in this chapter.

Remote Plug-in Life Cycle

Remote plug-in content becomes visible in the vSphere Client after you prepare and register the plug-in with a vCenter Server instance.

The plug-in displays when you connect a browser to the vSphere Client URL of a vCenter Server instance, or of a vCenter Server linked to the instance where the plug-in is registered, and log in.

A remote plug-in has the following life cycle stages:

Build

The plug-in developer assembles necessary resources and builds the package(s) to implement the remote plug-in.

Run

The plug-in developer starts the plug-in manifest server.

Registration

The plug-in developer registers the plug-in as a vCenter Server extension, using the ExtensionManager managed object of the vCenter Server instance.

Discovery

A vCenter Server instance discovers a new plug-in registration, or a new version registration.

Deployment

A vCenter Server instance downloads the plug-in manifest file, verifies certification and compatibility, and configures its reverse proxy to route server requests.

Use

The browser downloads plug-in resources from a back-end server into an iFrame of the vSphere Client user interface, and the plug-in user interface operates in conjunction with the back-end server or servers.

Uninstallation

The developer unregisters the plug-in, and all linked vCenter Server instances delete the routing configuration.

Remote Plug-in Deployment

When the vSphere Client discovers a remote plug-in, it schedules the plugin for deployment. vCenter Server takes the following steps to deploy a remote plug-in in the vSphere Client:

- Download the remote plug-in manifest, `plugin.json`, from the location specified in the `ClientInfo` property registered in the ExtensionManager. When you use the extension-registration script provided in the SDK this is the value of the `-pu` or `-pluginUrl` argument. vCenter Server downloads your plug-in manifest into `/etc/vmware/vsphere-ui/vc-packages/vsphere-client-serenity/`*`your_plugin_name-your_plugin_version`*.
- Parse the plug-in manifest to determine whether its specified version and environment are compatible with deployment on this vCenter Server instance.
- Parse the plug-in manifest to determine what views will be shown in the user interface, and add them to an internal extension point table.
- Configure the vCenter Server reverse HTTP proxy to route plug-in UI traffic to the remote plug-in server or servers.

After these steps complete successfully, the vSphere Client user interface displays a notification that the remote plug-in is installed.

Plug-In Caching

When the vSphere Client installs a plug-in, it downloads the plug-in manifest file and caches it. The cached copy is re-used whenever the vsphere-ui process restarts.

After a plug-in has been unregistered, the vsphere-ui service detects the change immediately. The cached copy of the plug-in manifest is deleted from the cache.

When a plug-in is upgraded, the vsphere-ui service detects the change as soon as the plug-in registration entry is updated with a new version number. At that time, the old plug-in version is undeployed, the cached copy of the manifest is removed, and the new plug-in version is deployed.

If plug-in resources are upgraded without changing the version number, such as when developing a plug-in, the vsphere-ui service does not detect the change. To deploy plug-in changes during development, use `pluginDeveloperMode` to enable the `Redeploy` button in the user interface. For more information, see Redeploying Plug-ins During Development.

Remote Plugin Uninstallation

Unregistering a plug-in package from vCenter Server causes the vSphere Client service to delete the plug-in from the environment. A vCenter Server instance notifies all linked vCenter Server instances as soon as the plug-in is unregistered. The linked vCenter Server instances undeploy the plug-in, by using the following steps:

- Delete the plug-in from internal extension point tables.
- Delete the plug-in's reverse proxy routing rules.
- Delete the plug-in's manifest from cache.

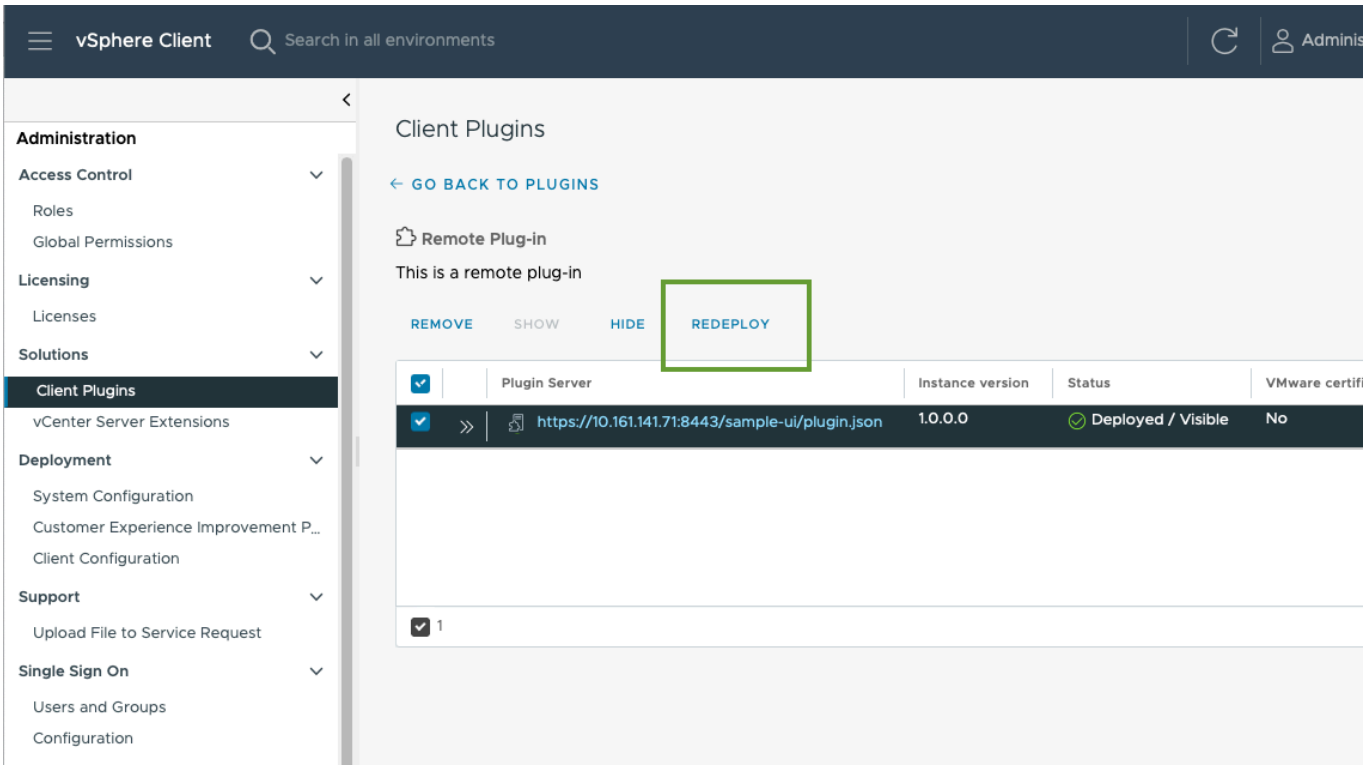
While the process of uninstallation takes place, the vSphere Client UI does not wait for the process to complete. You might need to refresh the browser window after the vSphere Client displays a notification that the uninstallation is complete.

Redeploying Plug-ins During Development

The vSphere Client service discovers new remote plug-ins as soon as they are registered with any linked vCenter Server instance. Some later changes to the registration record can cause redeployment of the plug-in. To bypass the need to change the registration record during plug-in development, you can trigger redeployment of plug-in changes by using the `Redeploy` feature.

When you are developing and testing a plug-in, you can trigger redeployment frequently by using the `Redeploy` button, without the need to make changes to the extension registration record. After the `Redeploy` button is enabled, it is visible in the **Client Plug-Ins** window, where it applies to any selected plug-in. The button causes vCenter Server to undeploy the selected plug-in, and then deploy it again based on the same registration record.

Figure 135: Redeploy feature in Plugin Developer Mode



To enable the `Redeploy` button, set `pluginDeveloperMode` by appending the query parameter `pluginDeveloperMode=true` to any URL loaded into the browser window. For example, the URL could be the following:

`https://`*`vcenter-server-fqdn`*`/ui/?pluginDeveloperMode=true`

When you set `pluginDeveloperMode`, it remains in effect until you browse to a URL that contains the query parameter `pluginDeveloperMode=false`, or until you refresh the browser window while the browser address field does not include `pluginDeveloperMode=true`.

> **NOTE**
> The `Redeploy` button requires the `Plugin.Management` privilege, which is available by default to all users who have the Administrator role.

Specifying Remote Plug-in Compatibility

The vSphere Client provides a way to specify compatibility between remote plug-ins and vCenter Server versions, and also compatibility with public cloud and private cloud environments.

When a remote plug-in is registered with a vCenter Server instance, each linked instance is notified. Each notified instance responds by checking compatibility constraints specified in the plug-in manifest. The constraints determine whether the notified instance will involve the plug-in in future requests that access vSphere resources on the registration instance.

Each notified instance must satisfy client constraints, and the registration instance must satisfy server constraints, to enable plug-in traffic. If both sides meet the constraint conditions, the notified instance deploys the plug-in for future connections to the registration instance.

Deployment includes configuring extension point metadata to deploy plug-in views in the vSphere Client user interface, and setting proxy rules to forward view requests to the plug-in server. This applies to views that access vSphere resources that the vCenter Server manages.

In the plug-in manifest file you can specify two points of control for plug-in compatibility. First, you can control whether any vCenter Server instance is compatible with the plug-in's user interface views. You specify this with the `vsphere.client` property. Second, you can control whether the plug-in is compatible with a vCenter Server instance where it is registered. You specify this with the `vcenter.server` property.

For example, if your plug-in is compatible with only an on-premises vCenter Server, you should specify that limitation in the `vcenter.server` object in the manifest. If the plug-in is registered in a cloud environment by mistake, vsphere-ui will refuse to deploy it. Or if your plug-in is incompatible with vSphere Client versions before 8.0, you should specify a minimum version of 8.0 in the `vsphere.client` manifest object. If a user connects to an earlier version vCenter Server, the client refuses to display any view served by the plug-in.

To specify compatibility constraints, modify the `requirements` object in the remote plug-in manifest file. The following nested objects are available to specify compatibility:

- `vcenter.server`

  Use this object to specify environments and versions of vCenter Server on which the plug-in can be deployed. You can specify these properties:

  - `environments`

    A specifier array for vCenter Servers. If property `environments` is specified in object `vcenter.server`, the plug-in can only be deployed there. If this property is absent, the plug-in can be deployed from any vCenter Server it is registered with. Legal values:
    - `onprem`
    - `cloud`
  - `version`

    A string containing a version or range of versions for vCenter Server instances where the plug-in can be registered and deployed. Legal formats:

- *version* = a single compatible version string
- [*version*, *version*] = an inclusive range of compatible versions
- (*version*, *version*) = an exclusive range of compatible versions
- [*version*,) = a minimum version
- (, *version*) = all prior versions

where *version* is an integer or a series of dot-separated integers: *n*[.*n*[.*n*[.*n*]]]

Although bracket and parenthesis look unmatched, "[" is inclusive and "(" is exclusive. For example [8.0, 9.0) means everything from 8.0 up to but not including 9.0.

A best practice is to specify the form [*version*, *version*) to indicate compatibility that begins at one major release and includes all minor releases up to, but not including, the next major release.

The `version` property in this context applies to a vCenter Server instance with which a remote plug-in is registered. If a version constraint is present in the `vcenter.server` object, `vsphere-ui` processes in all linked vCenter Server instances will verify that the registration instance satisfies the version constraint. If it does not, `vsphere-ui` will not deploy the plug-in for connections to the registration instance of vCenter Server. If the `version` property is absent, any `vsphere-ui` is free to deploy the plug-in if other compatibility constraints are satisfied.

- `vsphere.client`

  Use this object to specify environments and versions of vSphere Client that can display the views served by the plug-in. When the plug-in is registered on one vCenter Server instance, and a user connects to a second vCenter Server instance, the `vsphere-ui` of the second instance will display plug-in views for resources managed by the registration instance only if the second instance satisfies the `vsphere.client` compatibility requirements.

  – `environments`

    A specifier array for vCenter Servers. If property `environments` is specified in object `vcenter.server`, the plug-in can only be deployed there. If this property is absent, the plug-in can be deployed from any vCenter Server it is registered with. Legal values:

    - `onprem`
    - `gateway`
    - `cloud`

  – `version`

    A string containing a version or range of versions for vSphere environments. If the `version` property is present in the `vsphere.client` object, it limits the versions that can deploy the plug-in to access resources on a vCenter Server instance where the plug-in is registered. Legal formats are:

    - *version* = a single compatible version string
    - [*version*, *version*] = an inclusive range of compatible versions
    - (*version*, *version*) = an exclusive range of compatible versions
    - [*version*,) = a minimum version
    - (, *version*) = all prior versions

where *version* is an integer or a series of dot-separated integers: *n*[.*n*[.*n*[.*n*]]]

Remote Plug-in Topologies

Enhanced linked mode (ELM) enables you to manage all linked vCenter Server instances from a single vSphere Client connected to any one of the linked vCenter Server instances. The way that the vSphere Client visualizes the vSphere resources on a vCenter Server instance depends on the set of remote plug-ins installed on that instance. The topology of

remote plug-ins is the overall configuration of plug-ins that determines how the vSphere Client interacts with the resources on different vCenter Server instances.

Remote Plug-in Terminology

The following terms are useful to understand concepts in this chapter.

- Plug-in – A product that extends the vSphere user interface with additional functionality.
- Plug-in Version – The version of the plug-in specified at the time of registration.
- Plug-in Instance – A plug-in server registered with a vCenter Server instance. The plug-in instance is defined by the plug-in product, version, and plug-in manifest URL.
- Plug-in UI – The part of the plug-in loaded inside an iFrame within the vSphere Client UI in the browser.
- Plug-in Server – A back-end process to which the plug-in UI talks.
- Plug-in Manifest Server – The plug-in process that serves the plug-in manifest file.
- Plug-in Auxiliary Server – A plug-in process that serves some plug-in resources, but not the manifest file.

Visibility of Remote Plug-in Views

In an Enhanced Linked Mode (ELM) environment, the vSphere Client is capable of loading views from plug-in instances registered with any of the linked vCenter Server instances. For this example, suppose you are working with three linked vCenter Server instances: VC1, VC2, VC3. You register instances of plug-in A with VC1 and VC2, but not with VC3.

When you register plug-in A with VC1, and again with VC2, all three linked vCenter Server instances discover the registrations. Assume that plug-in A has no configured version or environment constraints, so deployment can proceed freely on all three vCenter Server instances. Plug-in views are visible in your vSphere Client only when appropriate, as defined by these rules:

- Global views are always present in the vSphere Client, regardless of which linked instance of vCenter Server you connect to. A plug-in instance selector enables you to choose between plug-in servers. For example, if you connect to VC3 you can choose between the global views of plug-in A instances registered with VC1 and VC2.
- The visibility of an object-specific plug-in view depends on which vCenter Server instance manages the object. When you select an object managed by a given vCenter Server instance, you have access to object-specific views served by the plug-ins registered with that vCenter Server instance. For example, if you connect to VC3 and select an object managed by VC1, the vSphere Client displays the object-specific view served by plug-in A. However, if you connect to VC1 and select an object managed by VC3, the vSphere Client does not display the object-specific view because VC3 has no instance of plug-in A registered.
- Plug-in visibility can also be limited by version differences between the plug-in and your vSphere Client. For example, suppose you connect to VC3 and select an object managed by VC2, but the instance of plug-in A registered with VC2 uses a new method not recognized by the older version of the JavaScript API served by VC3. Your version of the vSphere Client is unable to display the view served by plug-in A.

  NOTE
  To avoid run-time issues with incompatible software versions, configure compatibility constraints as described in Specifying Remote Plug-in Compatibility.

Remote Plug-in Multi-Instance Support

The remote plug-in architecture allows for multiple instances of the same remote plug-in to be deployed within an ELM environment. Instances of a remote plug-in provide distinct server processes that can be completely equivalent in function or can implement different versions of the plug-in functionality. The illustrations that follow show characteristics of some possible topologies.

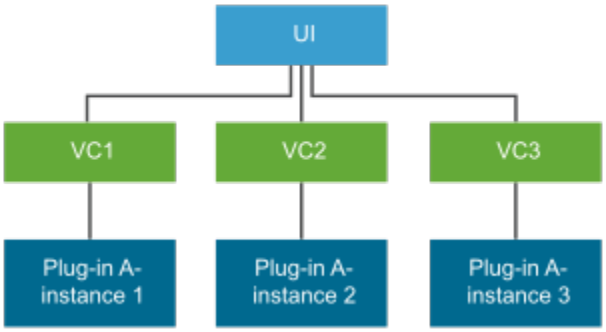
For example, consider an ELM environment with three vCenter Server instances. Suppose that the following extension registration commands are issued:

```
./extension-registration.sh -action registerPlugin -url https://vc1.example.com/sdk -u
"Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'Remote Plugin Test' -s
'A test plugin demonstrating plugin instances' -k com.example.remoteplugin.test -pu
"https://pluginserver1.example.com/path/to/plugin.json" -v "1.0.0" -st
plugin1_server_thumbprint  -remote

./extension-registration.sh -action registerPlugin -url https://vc2.example.com/sdk -u
"Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'Remote Plugin Test' -s
'A test plugin demonstrating plugin instances' -k com.example.remoteplugin.test -pu
"https://pluginserver2.example.com/path/to/plugin.json" -v "1.0.0" -st
plugin2_server_thumbprint -remote

./extension-registration.sh -action registerPlugin -url https://vc3.example.com/sdk -u
"Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'Remote Plugin Test' -s
'A test plugin demonstrating plugin instances' -k com.example.remoteplugin.test -pu
"https://pluginserver3.example.com/path/to/plugin.json" -v "1.0.0" -st
plugin3_server_thumbprint -remote
```

Figure 136: One Plug-in instance per vCenter Server

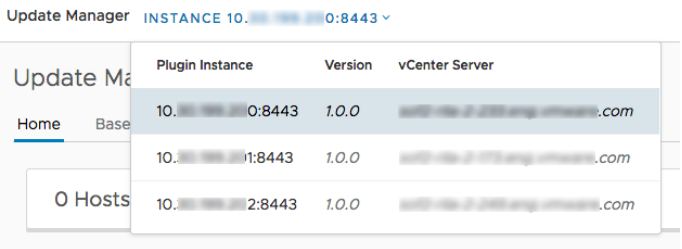


This topology has three instances of PluginA, such that the first plug-in server is registered with VC1, the second is registered with VC2, and the third server is registered with VC3. These plug-in servers are completely equivalent in function: the plug-in manifests that they host and the plug-in specific bits they run are identical. Because each server has a different URL for its manifest file, they are three separate instances of PluginA.

Object views will be loaded from the plug-in instance connected with the vCenter Server instance that the object belongs to. For example, if the plug-in declares a card on the VM summary page, then browsing VMs on VC2 will load the card from the second plug-in server, and browsing VMs on VC3 will load the card from the third plug-in server. Calls to the plug-in back end will be routed to the corresponding plug-in server instance.

Global views, however, will be aggregated in a single global view with an additional instance selector component that allows the user to choose between the global views of the different instances. A sample instance selector is shown below:

Figure 137: Selector for Single Plug-in Instance per vCenter Server



The instance selector shown above displays three back-end servers that represent the three instances of PluginA with version 1.0.0. The **Plugin Instance** column displays the IP address or fully qualified domain name of each plug-in instance, and the **vCenter Server** column displays the vCenter Server that each plug-in instance is connected to. Switching between the items in this drop-down will load the global view of the remote plug-in with version "1.0.0" from the specified plug-in instance.

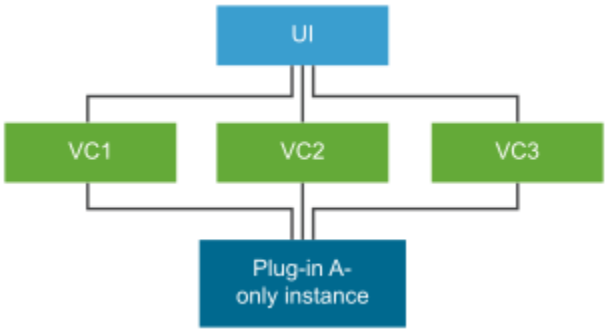
An alternative topology, which could be considered for a data center with lower access volume, might configure a single shared plug-in instance like this:

```
./extension-registration.sh -action registerPlugin -url https://vc1.example.com/sdk -u
"Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'Remote Plugin Test' -s
'A test plugin demonstrating plugin instances' -k com.example.remoteplugin.test -pu
"https://pluginserver1.example.com/path/to/plugin.json" -v "1.0.0" -st
plugin1_server_thumbprint  -remote

./extension-registration.sh -action registerPlugin -url https://vc2.example.com/sdk -u
"Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'Remote Plugin Test' -s
'A test plugin demonstrating plugin instances' -k com.example.remoteplugin.test -pu
"https://pluginserver1.example.com/path/to/plugin.json" -v "1.0.0" -st
plugin1_server_thumbprint -remote

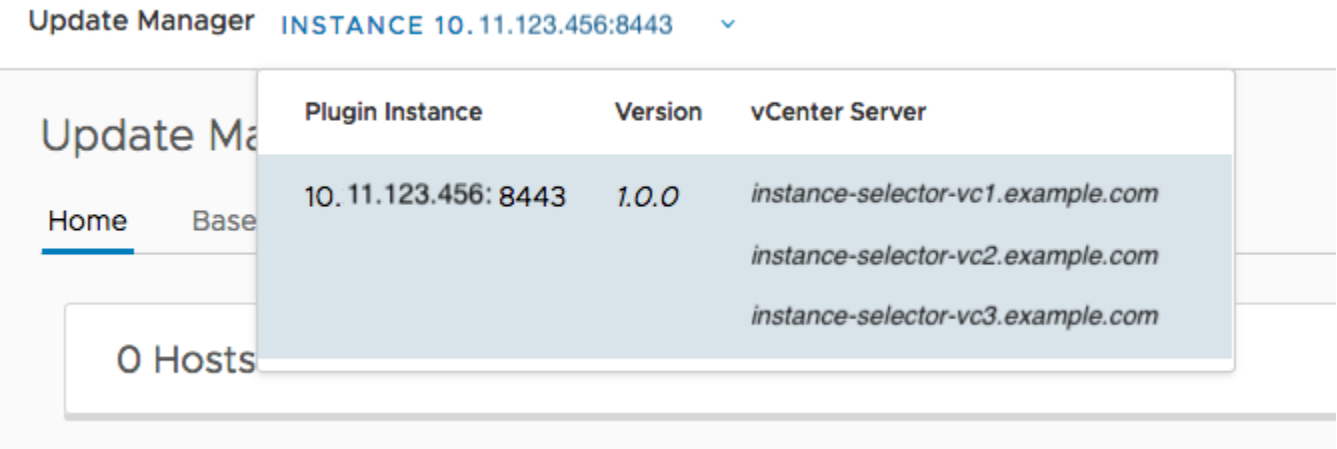
./extension-registration.sh -action registerPlugin -url https://vc3.example.com/sdk -u
"Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'Remote Plugin Test' -s
'A test plugin demonstrating plugin instances' -k com.example.remoteplugin.test -pu
"https://pluginserver1.example.com/path/to/plugin.json" -v "1.0.0" -st
plugin1_server_thumbprint -remote
```

Figure 138: Singleton Plug-in Instance for Link Group



The plug-in is registered with all linked instances of vCenter Server. All registrations have the same plug-in ID, version, and manifest URL, so this is considered a single instance. All plug-in object specific views will be loaded from this one plug-in server. Because no other versions of the same remote plug-in are present in the environment the global view will contain a single item that represents the global view of the singleton instance, as shown in the following example selector:

Figure 139: Selector for Singleton Plug-in



With this topology, the plug-in can be upgraded to a newer version for an entire link group by replacing a single process. However, a singleton plug-in works best in a homogenous environment. In an environment that contain sdifferent versions of vCenter Server, the singleton plug-in might need to handle different API versions.

Similarly, if you upgrade an instance of vCenter Server in a singleton plug-in configuration, the plug-in could become incompatible or could fail entirely. For more flexible version support, consider using multiple-instance topologies instead.

Differentiating Plug-in Instances

When the vSphere Client checks the vCenter Server's Extension Manager for new remote plug-ins it also checks for new instances of a remote plug-in. If the vSphere Client finds two extension registrations in two different vCenter Server instances such that the extension ID and extension version are the same but the plug-in manifest URL (ExtensionClientInfo.url) is different, then these two extensions are considered to be different instances of the same remote plug-in. If the plug-in manifest URL is also the same for both registrations, both vCenter Server instances share the same plug-in instance.

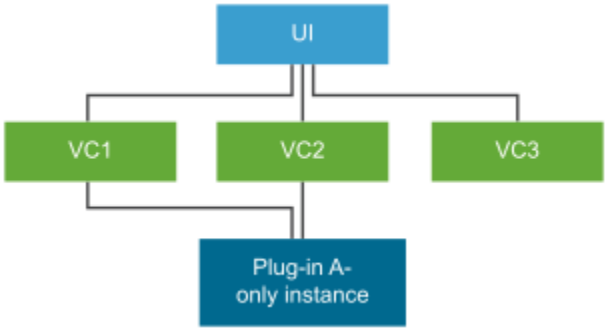
For example, consider an ELM environment with three vCenter Server instances. Suppose that the following extension registration commands are issued:

```
./extension-registration.sh -action registerPlugin -url https://vc1.example.com/sdk -u
"Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'Remote Plugin Test' -s
'A test plugin demonstrating plugin instances' -k com.example.remoteplugin.test -pu
"https://pluginserver1.example.com/path/to/plugin.json" -v "1.0.0" -st
plugin1_server_thumbprint  -remote
```

```
./extension-registration.sh -action registerPlugin -url https://vc2.example.com/sdk -u
"Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'Remote Plugin Test' -s
'A test plugin demonstrating plugin instances' -k com.example.remoteplugin.test -pu
"https://pluginserver1.example.com/path/to/plugin.json" -v "1.0.0" -st
plugin1_server_thumbprint -remote
```

Both VC1 and VC2 have the same plug-in manifest, and thus the same plug-in server. This is considered to be a single plug-in instance, registered with two vCenter Server instances.

Figure 140: Partial Coverage by Singleton Plug-in



In this topology, browsers connected to VC1 or VC2 will see the same object-specific views. In a global view context, a browser connected to any of the linked vCenter Server instances displays a view selector that enables the user to choose a path to the plug-in server. Both VC1 and VC2 provide a proxy route to the same plug-in instance.

Now suppose that we register the following extension in VC3:

```
 ./extension-registration.sh -action registerPlugin -url https://vc3.example.com/sdk -u
"Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'Remote Plugin Test' -s
'A test plugin demonstrating plugin instances' -k com.example.remoteplugin.test -pu
"https://pluginserver2.example.com/path/to/plugin.json" -v "1.0.0" -st
plugin2_server_thumbprint -remote
```

The extension registered in VC3 has the same ID and version as the one registered in VC1 and VC2 but has a different manifest URL. When the extension is registered, the vSphere Client will detect that this is a different instance of the remote plugin with ID `com.example.remoteplugin.test`, version `1.0.0`. The UI will show the following behavior:

- Object-specific views declared by plug-in instance 2 (registered in VC3) will be shown for the corresponding objects from VC3. However, the views declared by instance 1 (registered in VC1 and VC2) will be shown for objects from VC1 and VC2.
- For global views, a single entry point in the object navigator displays a plug-in instance view selector where the user will be able to switch between the global views supplied by either of the two plug-in instances.

Different plug-in instances can also have different plug-in versions. For example, consider an ELM environment with three vCenter Server instances. Suppose that the following extension registration commands are issued:

```
./extension-registration.sh -action registerPlugin -url https://vc1.example.com/sdk -u
"Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'Remote Plugin Test' -s
'A test plugin demonstrating plugin instances' -k com.example.remoteplugin.test -pu
"https://pluginserver1.example.com/path/to/plugin.json" -v "1.0.0" -st
plugin1_server_thumbprint  -remote

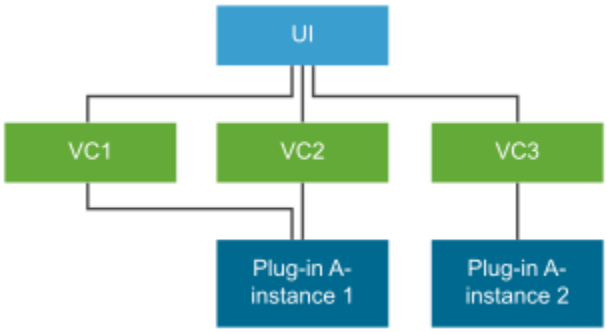
./extension-registration.sh -action registerPlugin -url https://vc2.example.com/sdk -u
"Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'Remote Plugin Test' -s
'A test plugin demonstrating plugin instances' -k com.example.remoteplugin.test -pu
"https://pluginserver1.example.com/path/to/plugin.json" -v "1.0.0" -st
plugin1_server_thumbprint -remote

./extension-registration.sh -action registerPlugin -url https://vc3.example.com/sdk -u
"Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'Remote Plugin Test' -s
'A test plugin demonstrating plugin instances' -k com.example.remoteplugin.test -pu
```

```
"https://pluginserver2.example.com/path/to/plugin.json" -v "2.0.0" -st
plugin2_server_thumbprint -remote
```

In this topology, VC1 and VC2 are running with a single instance of the plug-in, at version **1.0.0**, while VC3 is running with a second instance of the plug-in, at version **2.0.0**.

Figure 141: Remote Plug-in Version Differentiation



The newer version might have support for new features, such as changes in the API. This capability allows a plug-in to support custom features for some managed objects. It also can help to facilitate testing and rolling upgrades.

Custom Topologies

You can design your plug-in with any topology which may be appropriate for your needs. The following limitations apply:

- At most one remote plug-in instance with a given extension ID can be registered per vCenter Server
- Remote plug-in registrations with different extension IDs are considered different plug-ins.
- Remote plug-in registrations with the same extension ID but different manifest file locations are considered different instances of the same plug-in. In the case of multiple instances, any object-specific view is loaded by the plug-in server instance that is registered with the vCenter Server instance that manages the object. The view makes back-end calls to the server instance that loaded it.
- Remote plug-in registrations with the same extension ID and the same manifest file location are considered a single instance. That instance serves all the vCenter Server instances with which it is registered. Any object-specific view is loaded by the single plug-in server instance that is registered with all the vCenter Server instances.
- Remote plug-ins with the same extension ID and different manifest file locations can also have different versions. In that case, any object-specific view is loaded by the instance that is registered with the vCenter Server instance that manages the object. The details of an object-specific view might differ, depending on the version of the plug-in that is registered with the vCenter Server instance. The view makes back-end calls to the server instance that loaded it.
- If an object is managed by a vCenter Server instance that has no instance of the plug-in registered, then no object-specific view is displayed for that plug-in.
- In case of multiple instances, a single global view will be displayed for the remote plug-in, aggregating the global views of all plug-in instances, and a plug-in view selector allows for switching between the global view content from the different instances.

Deploying Auxiliary Plug-in Servers

When you divide plug-in features between several auxiliary servers, you register each server with a string identifier that indicates the service it provides. Any plug-in server can locate a service it needs by using the service identifier to look up the service endpoint URL in the extension registry record.

When vCenter Server deploys a remote plug-in, it caches the plug-in manifest and configures a routing rule for the reverse HTTP proxy, which directs UI traffic to the manifest server process. If the plug-in includes auxiliary servers, vCenter Server also creates rules to route UI traffic to the auxiliary servers.

VCenter Server maintains a mapping of service identifiers to proxy URLs. Plug-in code that runs in the browser can access that mapping by using the `getPluginBackendInfo()` method. The resulting data allows the JavaScript code to determine the proxy URLs for the services it needs. For more information about `getPluginBackendInfo()`, see vSphere Client JavaScript API: Application Interface.

For more information about registerirg auxiliary plug-in servers, see Registering Auxiliary Plug-in Servers.

Remote Plug-in Deployment Example with Simultaneous Users

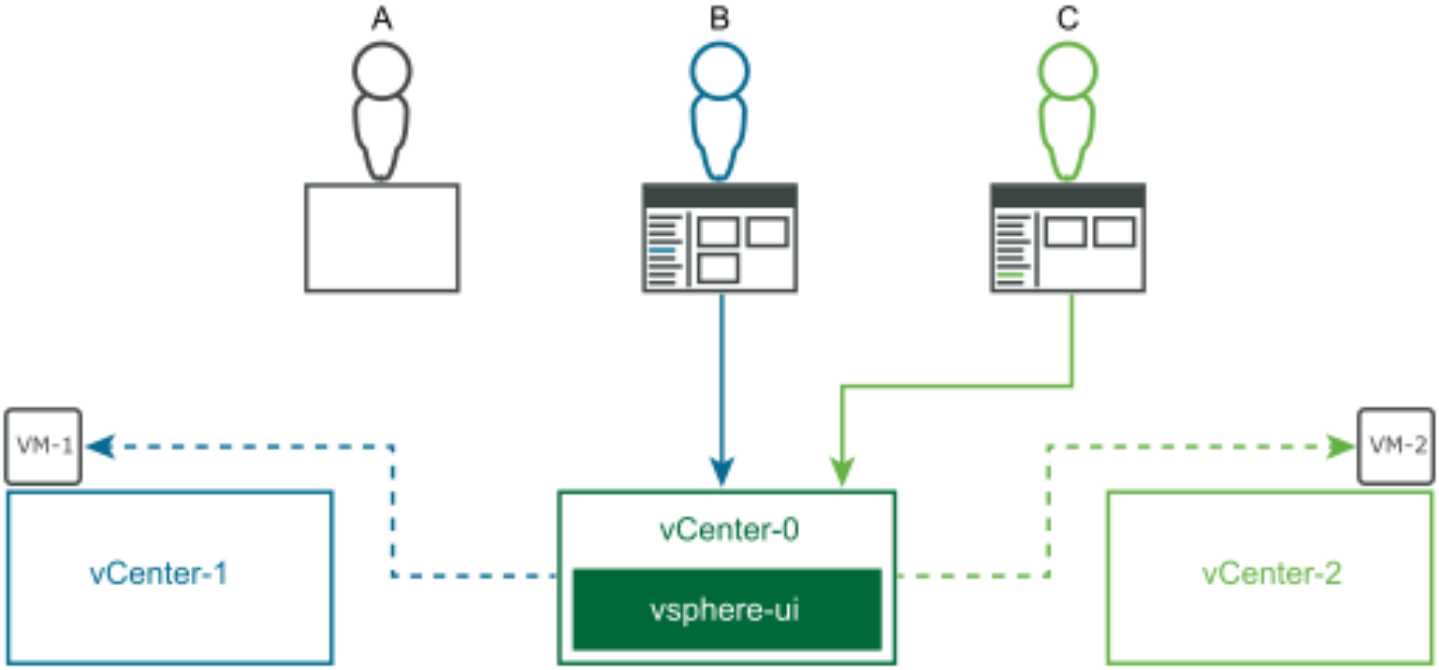
After being detected, a remote plug-in will be scheduled for deployment in the vSphere Client. The deployment of a remote plug-in, on a high level, consists of the following stages:

- vCenter Server downloads the remote plug-in manifest.
- vCenter Server parses the plug-in manifest.
- vCenter Server configures the VMware reverse HTTP proxy to route plug-in UI traffic.

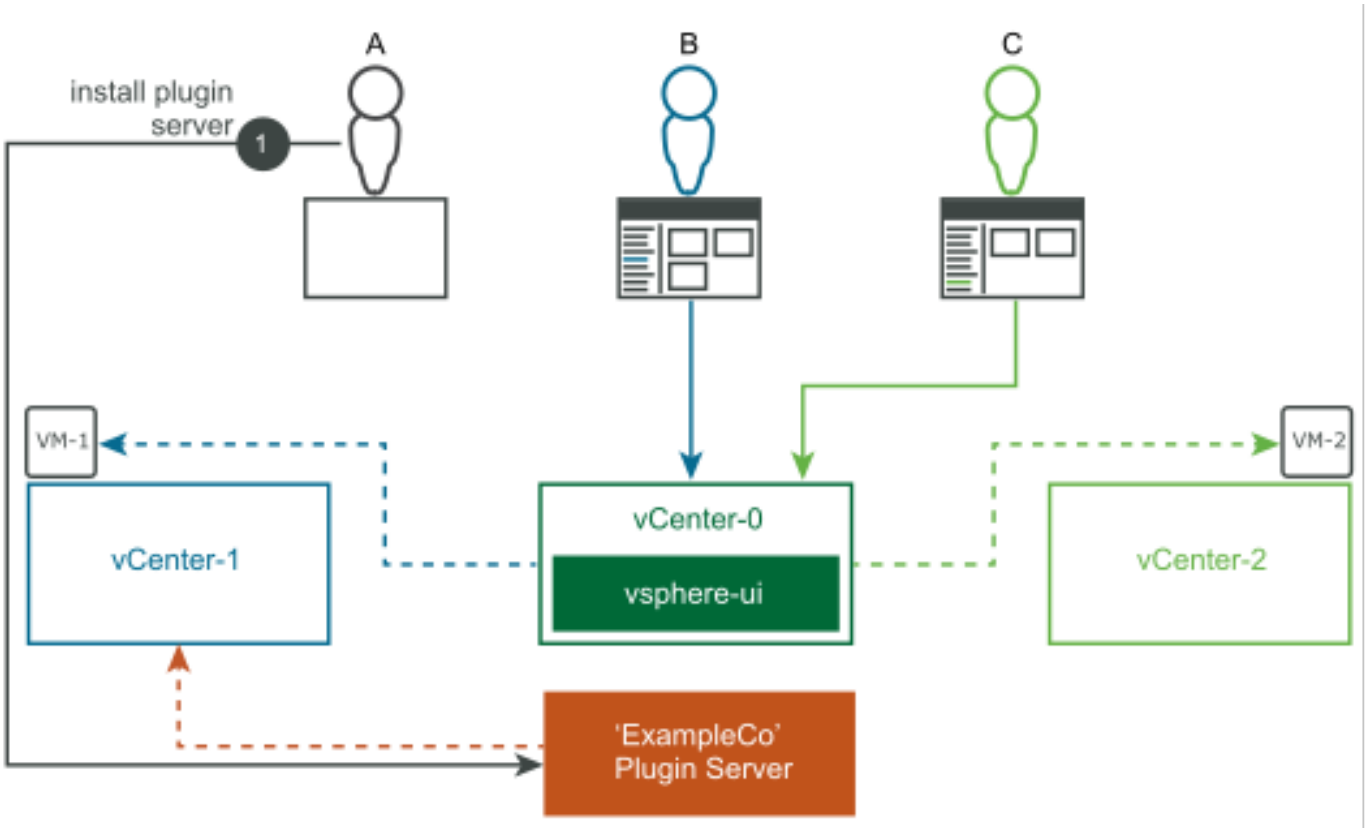
After these stages complete successfully, the vSphere Client UI displays a notification message that the remote plug-in is installed.

This example shows in more detail how the deployment process works, in a situation involving three users simultaneously accessing the data center. The initial state consists of the following:

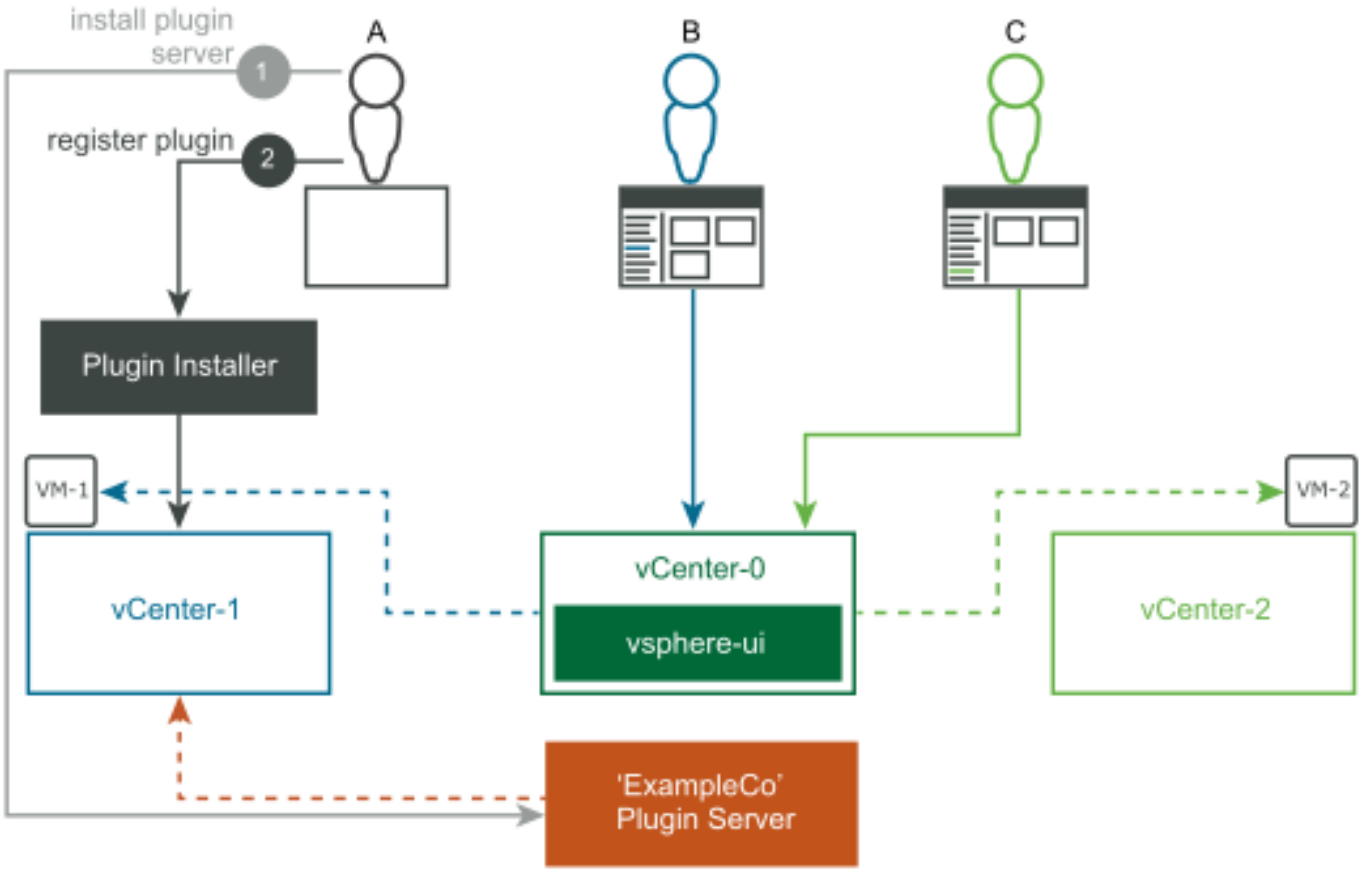
- Three vCenter Server instances in an ELM environment: vCENTER-0, vCENTER-1, and vCENTER-2.
- Three users are accessing the data center: Alpha, Blue, and Claire.
- Blue and Claire are already browsing the vSphere UI loaded from `vCENTER-0`
  - Blue is looking at the summary page of `VM-1` managed by `vCENTER-1`.
  - Claire is looking at the summary page of `VM-2` managed by `vCENTER-2`.
- Alpha is about to install a plug-in from Example Company:



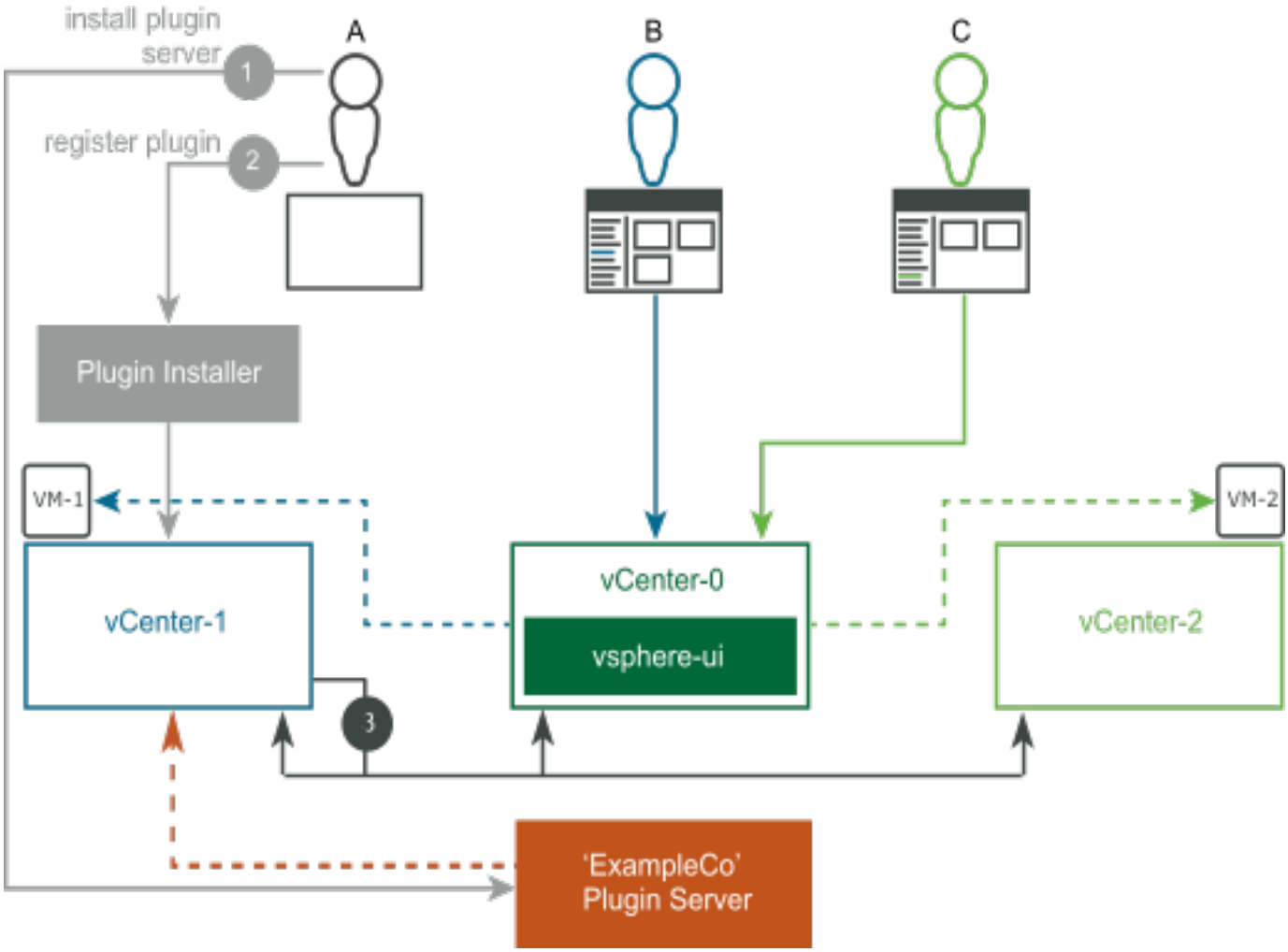
1. Alpha installs and configures the back-end server for the `ExampleCo` plug-in:



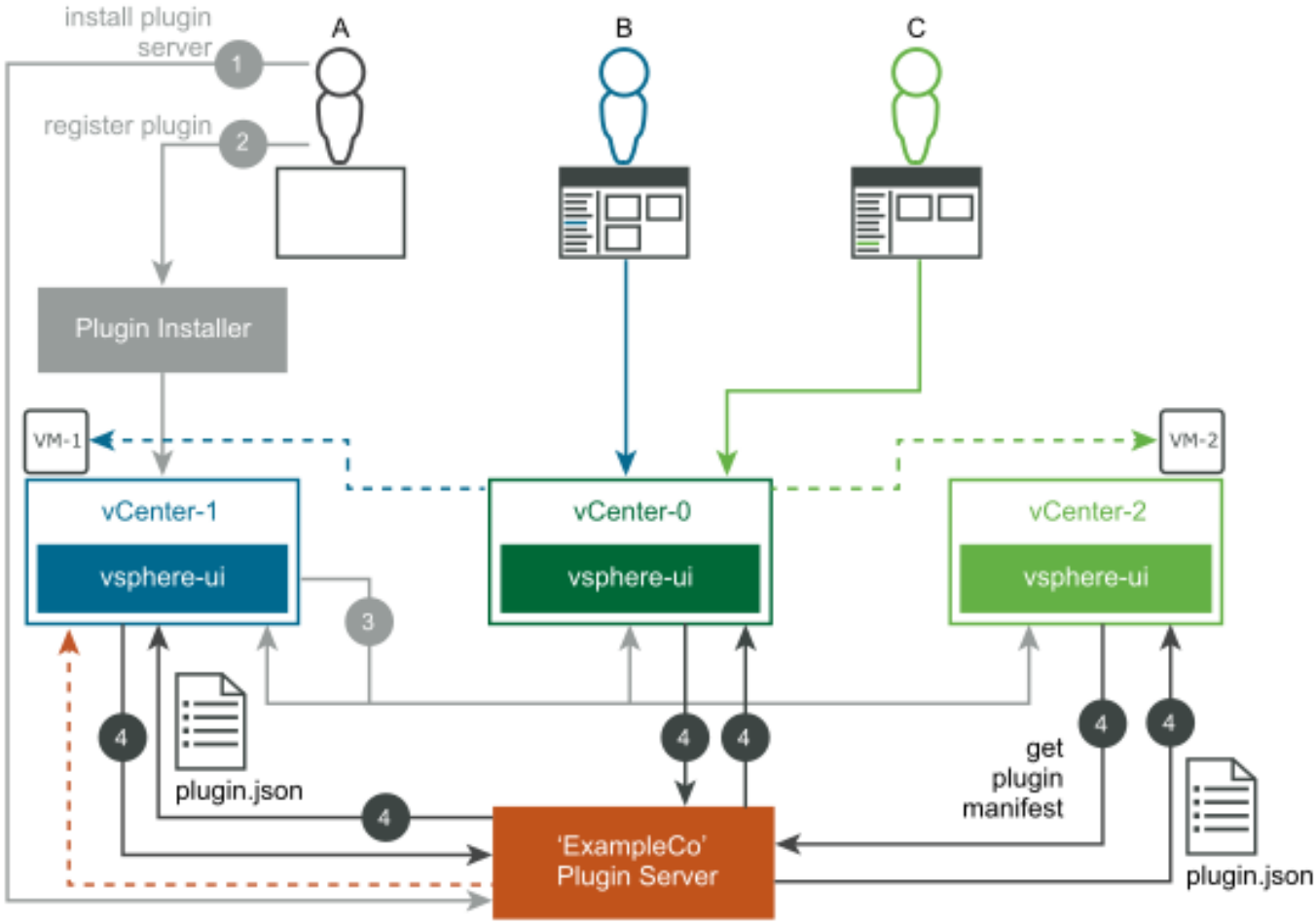
2. Alpha registers the `ExampleCo` plug-in with the `vCENTER-1` ExtensionManager by using Example Company's plug-in installer:



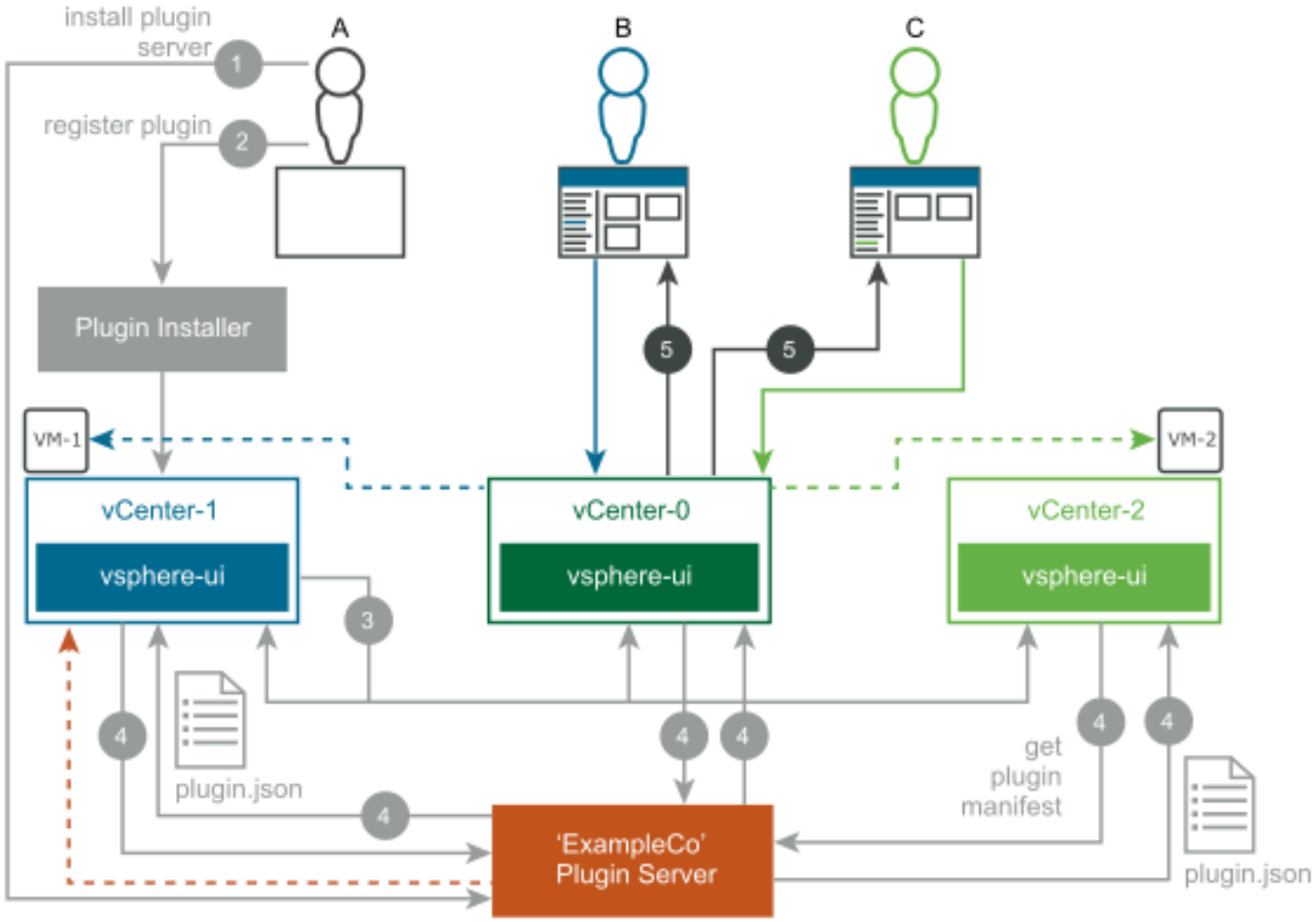
3. The plug-in registration triggers notifications to all linked vCenter Server instances:



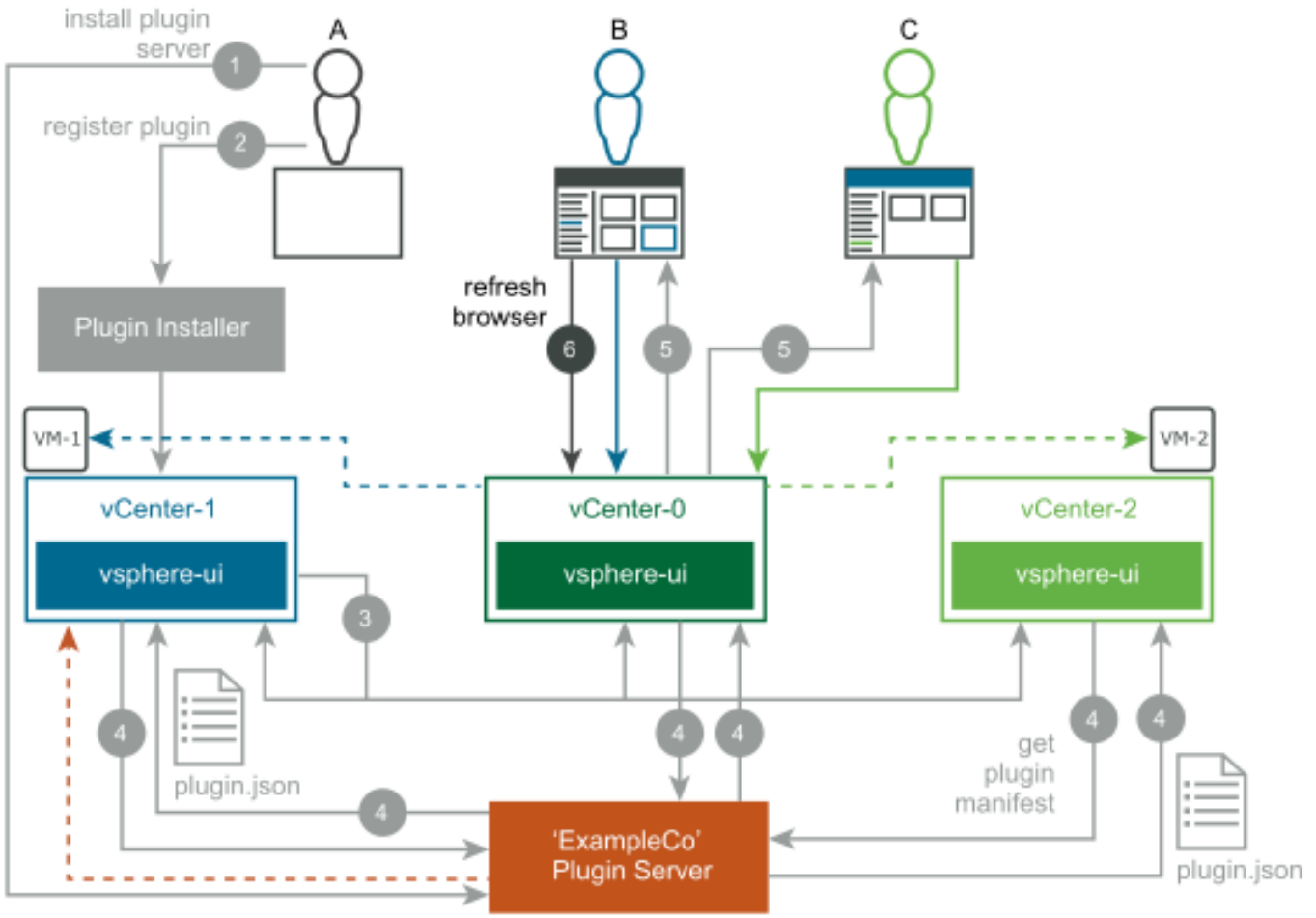
4. The `vsphere-ui` service in each vCenter Server instance downloads the plug-in manifest JSON of the `ExampleCo` plug-in from the plug-in manifest URL in the extension registration record:



5. The vsphere-ui service in `vCENTER-0` sends notifications to currently logged in vSphere Client users (Blue and Claire). Each user sees a blue notification banner at the top of the screen:

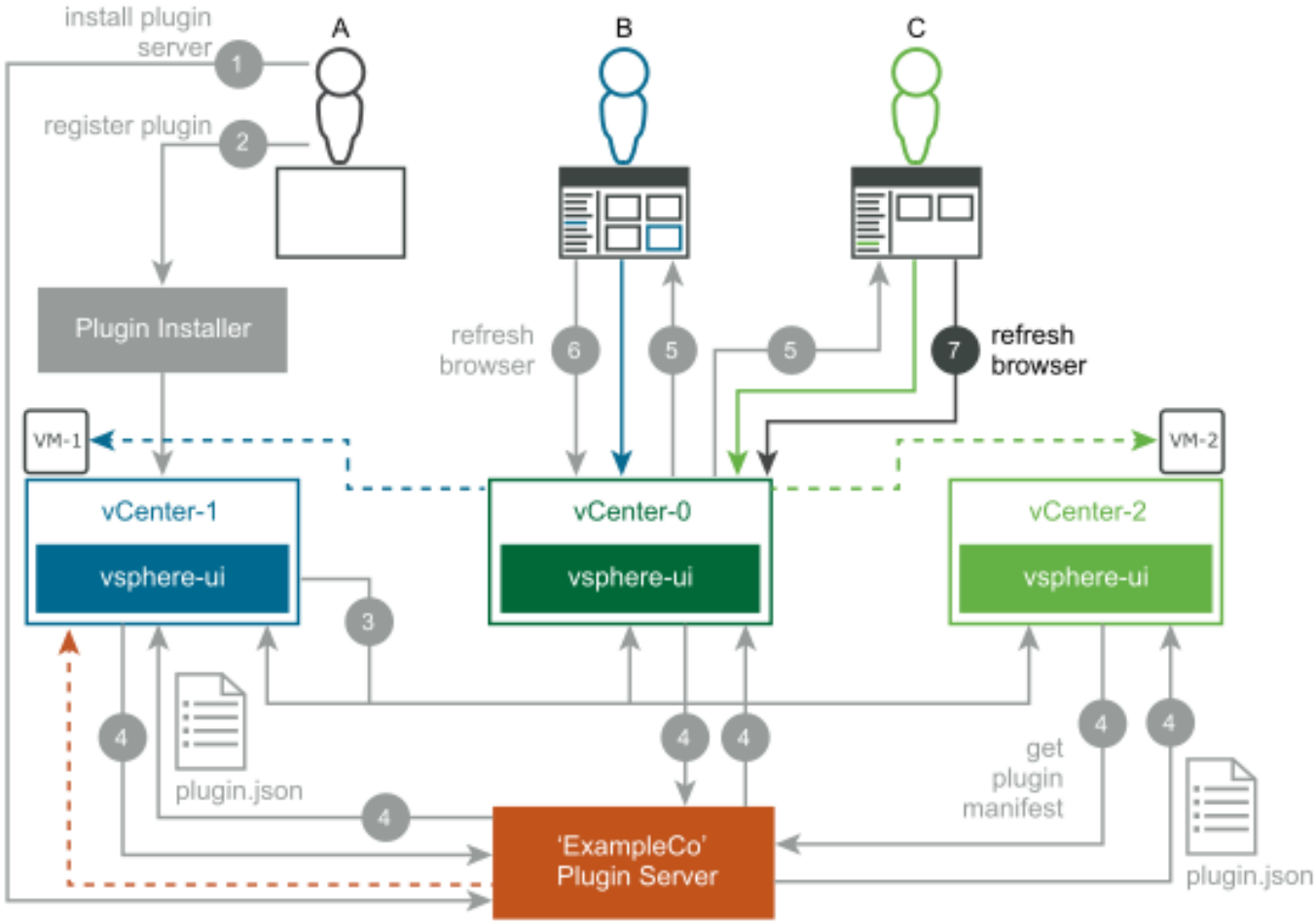


6. When Blue refreshes the vSphere Client UI in the browser, the `ExampleCo` plug-in is loaded for this user. The plug-in adds a card that extends the summary page of VMs. Because `VM-1` is managed by `vCENTER-1`, which has the `ExampleCo` plug-in registered, Blue sees the newly added card:

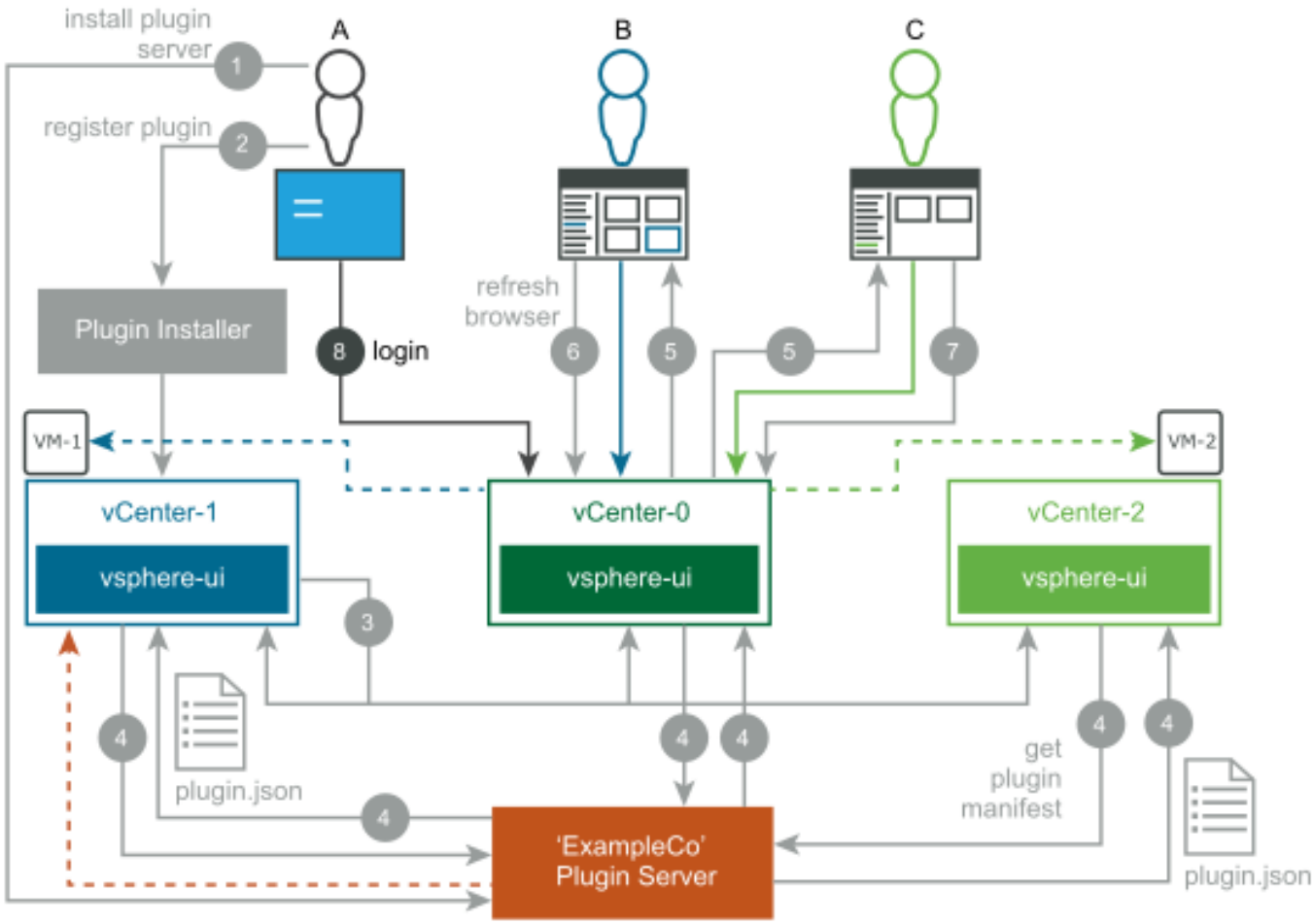


7. When Claire refreshes the vSphere Client UI in the browser, the `ExampleCo` plug-in can now be loaded for this user. However, Claire is looking at `VM-2`, which is managed by `vCENTER-2`. Because `vCENTER-2` does not have the `ExampleCo` plug-in registered, Claire does not see the newly added card.

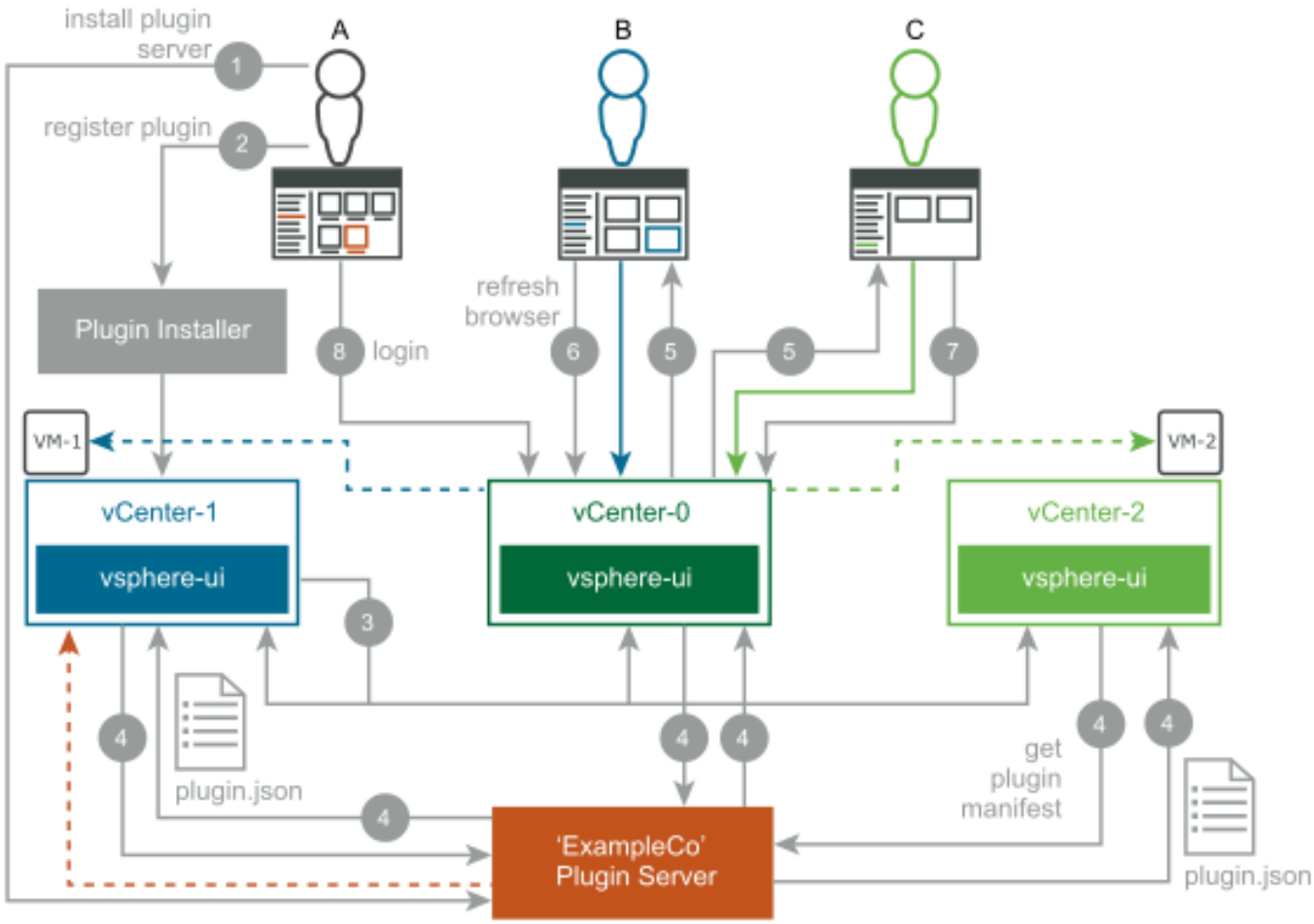
   If Claire later navigates to a VM on `vCENTER-1`, the vSphere Client will display the card added by the `ExampleCo` plug-in.



8. Alpha logs in to the vSphere UI connected to `VCENTER-0`:



9. Alpha has completed the login and has loaded the vSphere Client UI, which displays the home screen of the vSphere Client. `VCENTER-0` deployed the `ExampleCo` plug-in prior to the login, so Alpha sees the new home page menu item and the shortcut link immediately:



Remote Plug-in Multi-Version Support

Unlike local plug-ins, the remote plug-in architecture allows for co-existence of remote plug-ins with the same ID but different versions.

Consider an ELM environment with three vCenter Server instances and the following plug-in registrations:

```
 ./extension-registration.sh -action registerPlugin -url https://vcenter-ip-or-fqdn-of-vc1
/sdk -u "Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'ExampleCo' -s
'A test plugin demonstrating plugin instances' -k com.example.exampleco -pu "https://my-
remote-plugin-server-version-1/path-to/plugin.json" -v "1.0.0" -st
plugin_server_1_thumbprint -remote

 ./extension-registration.sh -action registerPlugin -url https://vcenter-ip-or-fqdn-of-vc2
/sdk -u "Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'ExampleCo' -s
'A test plugin demonstrating plugin instances' -k com.example.exampleco -pu "https://my-
remote-plugin-server-version-2/path-to/plugin.json" -v "2.0.1" -st
plugin_server_2_thumbprint -remote

 ./extension-registration.sh -action registerPlugin -url https://vcenter-ip-or-fqdn-of-vc3
/sdk -u "Administrator@vsphere.local" -p 'Admin!23' -c 'Example, Inc.' -n 'ExampleCo' -s
'A test plugin demonstrating plugin instances' -k com.example.exampleco -pu "https://my-
```

remote-plugin-server-version-3/*path-to*/plugin.json" -v **"3.2.0"** -st ***plugin_server_3_thumbprint*** -remote

These commands register three extensions (one in each of the three vCenter Servers) with the same ID (`com.example.exampleco`) but different versions - version `1.0.0` on VC1, version `2.0.1` on VC2 and version `3.2.0` on VC3. These are three different versions of the remote plug-in with ID `com.example.exampleco`. When you log in to the UI you will see the following:

- Object views declared by plug-in `com.example.exampleco` version `1.0.0` will be shown on applicable objects from VC1. Calls to the plug-in back-end server will be routed to the plug-in server dedicated to version `1.0.0` of the plugin.
- Object views declared by plug-in `com.example.exampleco` version `2.0.1` will be shown on applicable objects from VC2. Calls to the plug-in back-end server will be routed to the plug-in server dedicated to version `2.0.1` of the plug-in.
- Object views declared by plug-in `com.example.exampleco` version `3.2.0` will be shown on applicable objects from VC3. Calls to the plug-in back-end server will be routed to the plug-in server dedicated to version `3.2.0` of the plug-in.
- There will be a single entry point in the object navigator that will take the user to a plug-in instance/version selector view where the user will be able to switch between the global views of the different versions and instances of the remote plug-in.

Remote Plug-in Multi-Manifest Support

You can create a remote plug-in capable of supporting different vSphere Client feature sets. To deploy this plug-in, you specify different manifest files to match the feature sets. The vSphere Client will choose a manifest compatible with the schema it supports.

You should use the multi-manifest feature when you want a single plug-in version to support different feature sets in the vSphere Client or different capabilities in the JavaScript API. The plug-in user interface code can check the existence of API methods or invoke `app.getClientInfo()` to determine which feature set the vSphere Client makes available.

Before 8.0 U3, it was possible for a multi manifest with a `vsphere.client` and/or `vcenter.server` requirements specification to be parsed and chosen for deployment, despite being incompatible with vSphere Client and/or vCenter Server requirements. This caused the plug-in to appear as Incompatible.

As of vSphere 8.0 U3, plug-in environment requirements are processed as part of plug-in manifest parsing and if they are not satisfied, parsing of the current file fails, causing the platform to try the subsequent plug-in manifest file in the queue.

Start by creating two or more manifest files, each conforming to the schema of a supported feature set. For instance, `plugin-80.json` excludes a feature deprecated in vSphere Client 7.0, while `plugin-70.json` includes that feature. After you test your manifest files separately, create a list of their file names as a JSON object named `manifests` and store it in a file named `plugin-multi-manifest.json`:

```
{
  "manifestVersion": "1.0.0",
  "manifests": [
    "plugin-80.json",
    "plugin-70.json",
    "plugin-67.json"
  ]
```

```
}
```

The `manifests` object conforms to a separate multi-manifest schema, such that the manifest parser treats it as an ordered list. The parser tries to validate each manifest file in turn for compatibility with `vsphere-ui`. The first file that conforms to the parser's standard manifest schema is the manifest that the vSphere Client uses for the plug-in.

Then create a zip file containing the `plugin-multi-manifest.json` file and the supported manifest files for different feature sets that the plug-in supports. All the files must be at the root of the zip file, rather than in subdirectories. The zip file can have any name you choose.

Finally, register the plug-in with a vCenter Server instance. Use the name of the zip file as the `--pluginUrl` argument to the registration script or as the `client.url` property in the vCenter Server extension record that your registration tool creates.

> **NOTE**
> Multi-manifest support was added in vSphere 7.0 U1. To support versions before 7.0 U1 and a multi-manifest after 7.0 U1, you must add a manifest for all versions before 7.0 U1 in a `plugin.json` and place it inside of the Zip archive. This way, versions before 7.0 U1 use the `plugin.json` whereas versions after 7.0 U1 process the multi-manifest JSON. The first one that is parsed correctly will be used.

Choosing Extension Points for vSphere Client Plug-ins

The vSphere Client supports adding content at key extension points in the user interface. A plug-in developer can insert custom views that present objects and functions not provided by vSphere.

The available extension points closely follow the navigation experience in the vSphere Client, which facilitates a clear mapping of a plug-in to the views and workflows it owns. The extension points operate at a high level to allow the developer maximum creative space and flexibility.

Types of Extension Points in the vSphere Client

The vSphere Client provides a number of integration points that plug-ins can extend. These integration points are also known as extension points, where the developer can define extensions that integrate into the vSphere Client user interface. The extension points are designed to allow maximum flexibility for a plug-in while minimizing the potential impact on other plug-ins running concurrently.

Action and View Extensions

The vSphere Client offers two broad classes of extensions: views and actions. Views give direct access to part of the user interface, where the plug-in can define custom displays and controls. Actions generally trigger messaging and business logic.

Views and actions are specified in the plug-in manifest file because they are anchored to controls in the vSphere Client user interface. Actions are realized in the user interface by means of modal dialogs that the user activates by choosing from the plug-in submenu of the vSphere Client global actions menu. Alternatively, headless actions, without a dialog, can navigate directly to other plug-in views. The user activates a view by navigating to a page or a tab that has a plug-in view extension defined for that context.

When a view extension activates, the plug-in receives an iFrame context within which it is free to operate, given certain restrictions such as not accessing the iFrame parent or other DOM elements outside the iFrame. The plug-in has access to a JavaScript API that it can use to interact with processes outside the iFrame in a managed way. These interactions include retrieval of the reverse proxy endpoints and authenticating the back-end server.

The JavaScript API also provides methods for the plug-in to offer to the user actions that are anchored to controls within the plug-in iFrame. These actions do not appear in the global actions menu, so they are not described in the plug-in manifest file. For more information, see vSphere Client JavaScript API: Modal Interface.

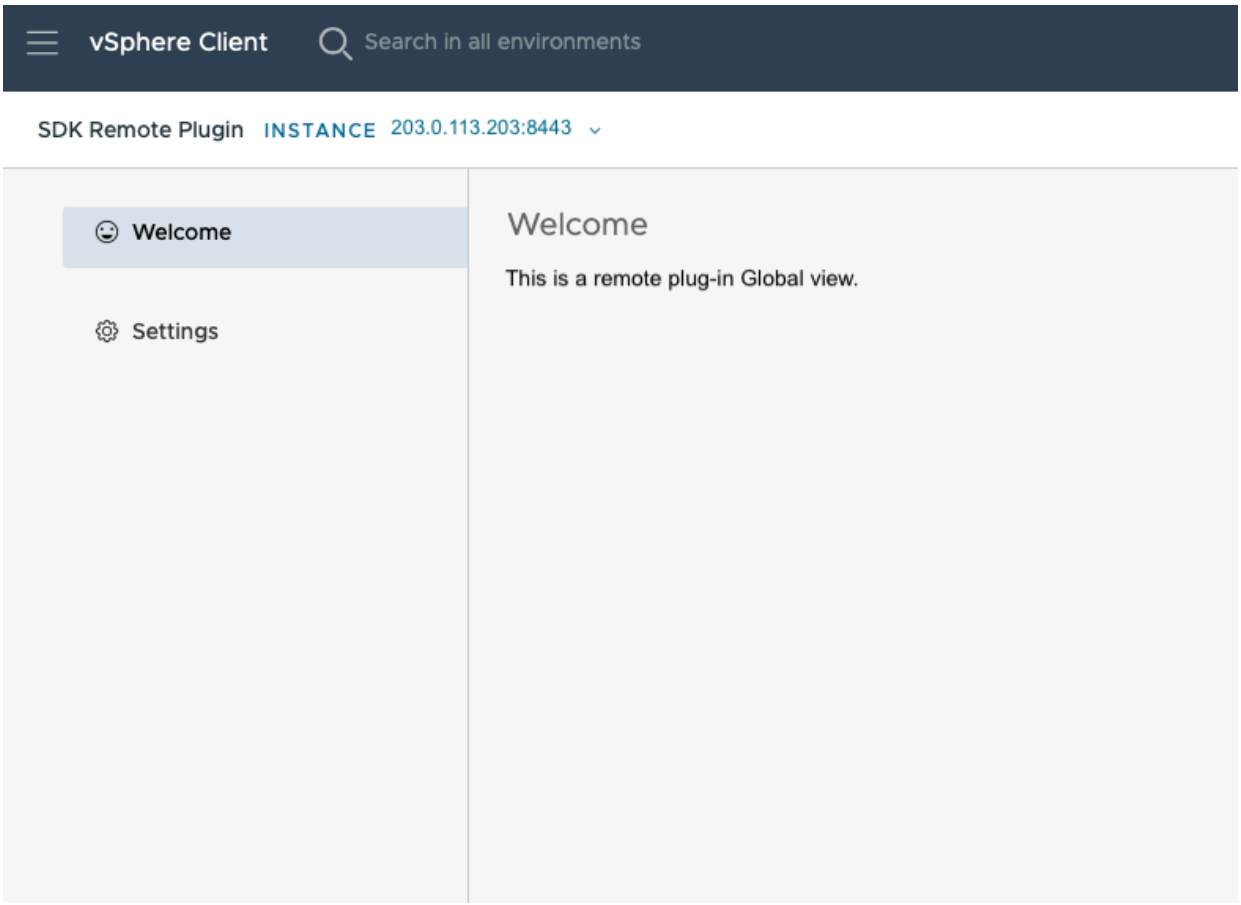
Extension Types

Extensions are separated into groups by the inventory context selection, and sub-groups by the purpose of the user interface element. The extension points are grouped as follows:

- **Global extension points**
  These are plug-in UI elements that have global scope. Their context is the entire plug-in, rather than a particular inventory object.
  - **View**
    This is a single global UI element that can consume a large section of the vSphere Client real estate. When multiple global pages are required, they should be implemented within this single global view. Navigation between the nested pages must be handled by the plug-in front end.

    For example, this includes configuration pages or status dashboards. The following illustration shows the screen context for a global view.

    Figure 142: Global View Extension

- **Object extension points**
  These are plug-in UI elements within the scope of the currently selected context object in the inventory. They are defined per object type and displayed only for the selected object. Their views can include other inventory objects or external objects, as long as there is a logical relevance to the current object.
  - **Summary Section View**

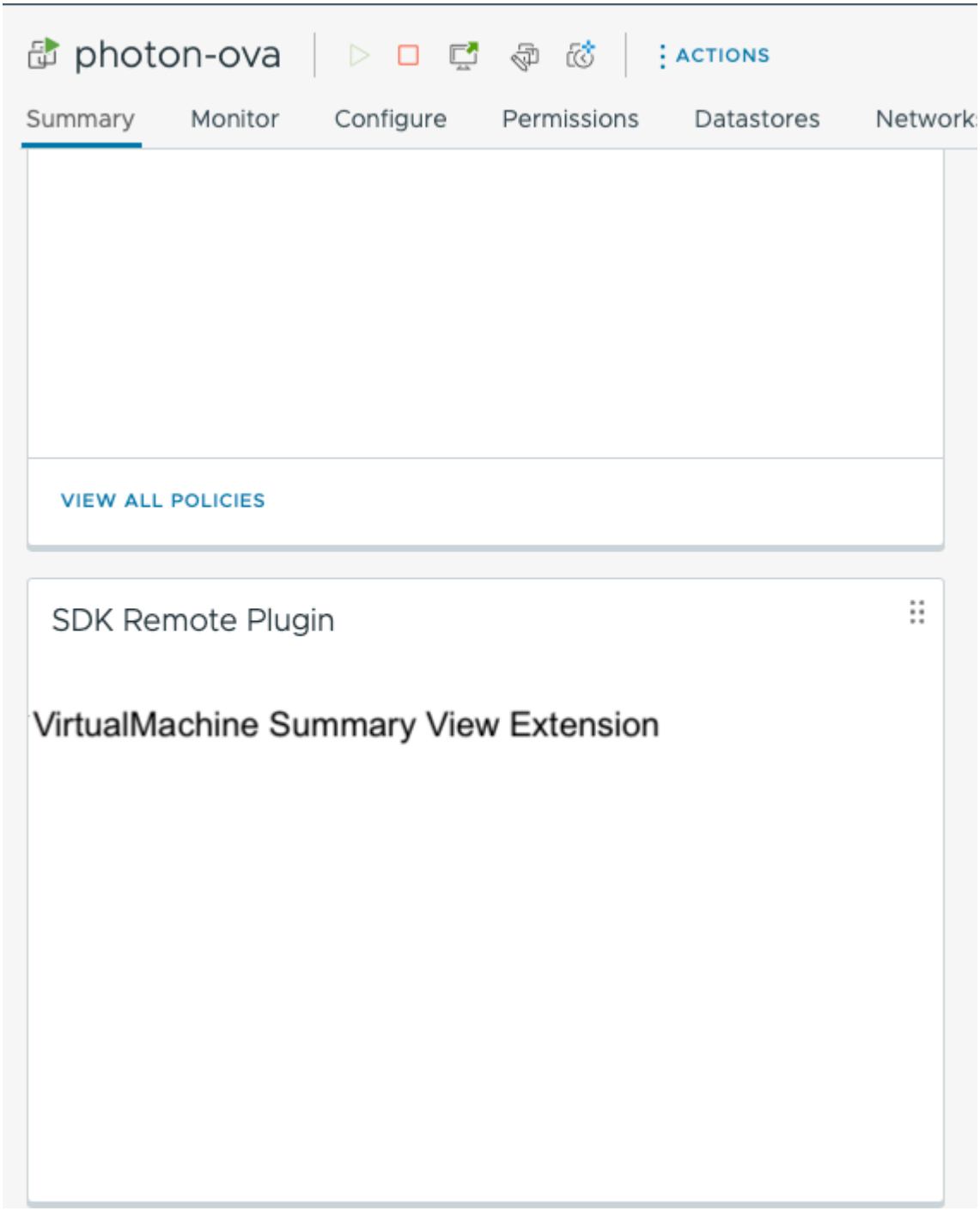
A Summary section view extension creates a small box in the object's Summary view tab. The box uses the Clarity card template. It should contain primarily simple name-value data. Optionally it can contain action buttons or links to more detailed Monitor or Configure pages.

For example, this could be a view for a selected host that shows the following:

- The number of virtual machines that need backup.
- A button that backs up all virtual machines that need backup.
- A link to a Monitor view that lists all virtual machines that need backup, and allows a user to back them up separately.

The following illustration shows the screen context for a VirtualMachine object Summary section view.

Figure 143: VirtualMachine Summary Section View Context



Summary section view extensions have fixed height but variable width. If you need more space, you can either select "large" width, create multiple cards in the object's Summary view tab, or link from a Summary view card to a Monitor view that contains more information.

You can create up to five cards in an object's Summary view tab. Each card can have its own title and its own icon. If the title is not specified, it defaults to the plug-in name. If the icon is not specified, it defaults to the plug-in icon.

All cards share a single `dynamicUri`, if used. For more information about `dynamicUri`, see Dynamic Extensions for Remote Plug-ins.

Specify a single card for the Summary view with the following syntax in the manifest:

```
...
  "summary": {
    "view": {
      "uri": "index.html?view-host-card"
    }
  }
...
```

Specify multiple cards for the Summary view with the following syntax in the manifest. As of vSphere 8.0 U3, card width may be specified as small, medium, or large.

```
...
  "summary": {
    "views": [
      {
        "id": "vmCard1",
        "titleKey": "Title 1",
        "icon": {
          "name": "icon_card1"
        },
        "size": {
          "width": "small"
        },
        "uri": "index.html?view-host-card-card1"
      },
      {
        "id": "vmCard2",
        "titleKey": "Title 2",
        "icon": {
          "name": "icon_card2"
        },
        "size": {
          "width": "large"
```

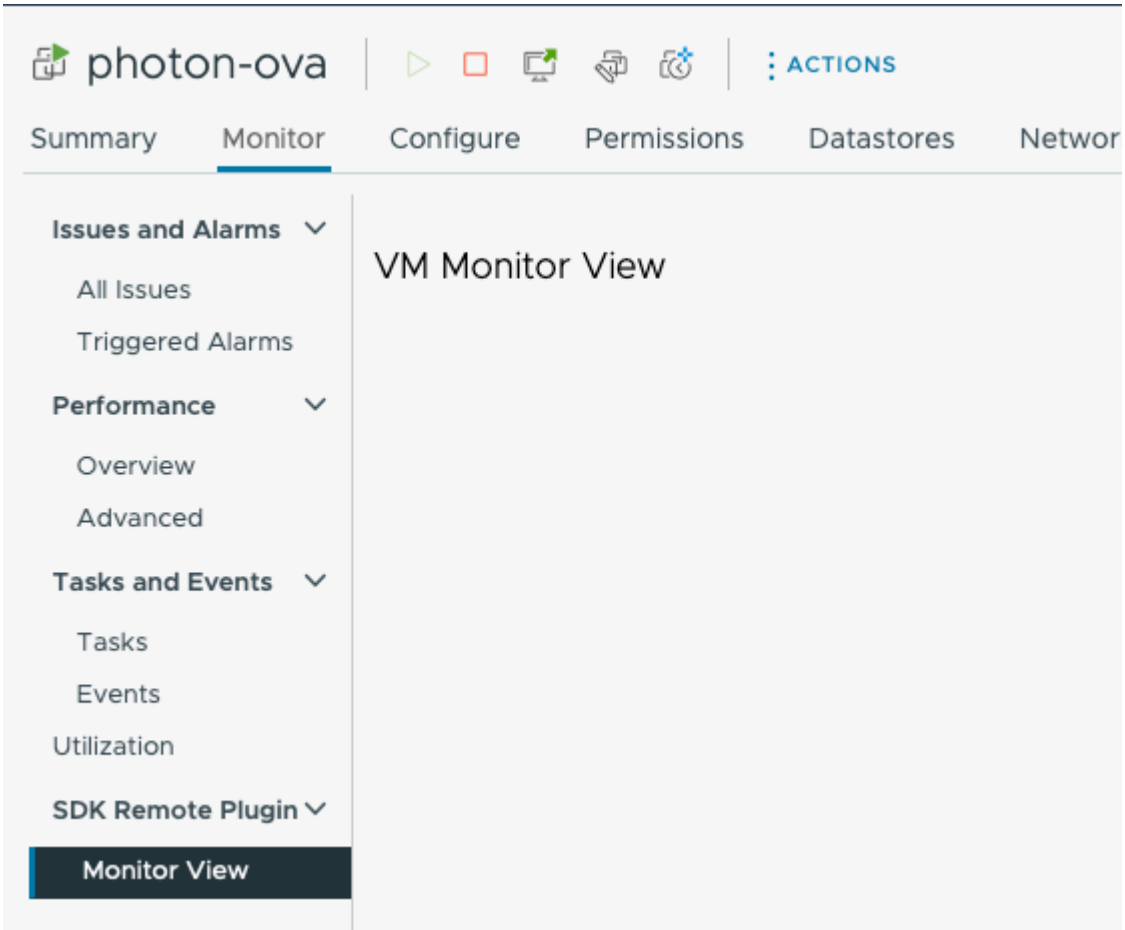
```
        },
        "uri": "index.html?view-host-card-card2"
      }
    ]
  }
...
```

- **Monitor Views**
  This comprises a single Monitor category with one or more views. It can contain detailed monitoring and maintenance data and workflows relevant to the current object.

  For example, you could use a Monitor view to show the backup status of all virtual machines on a selected host.

  The following illustration shows the screen context for an Object Monitor view

  Figure 144: Object Monitor View Screen Context

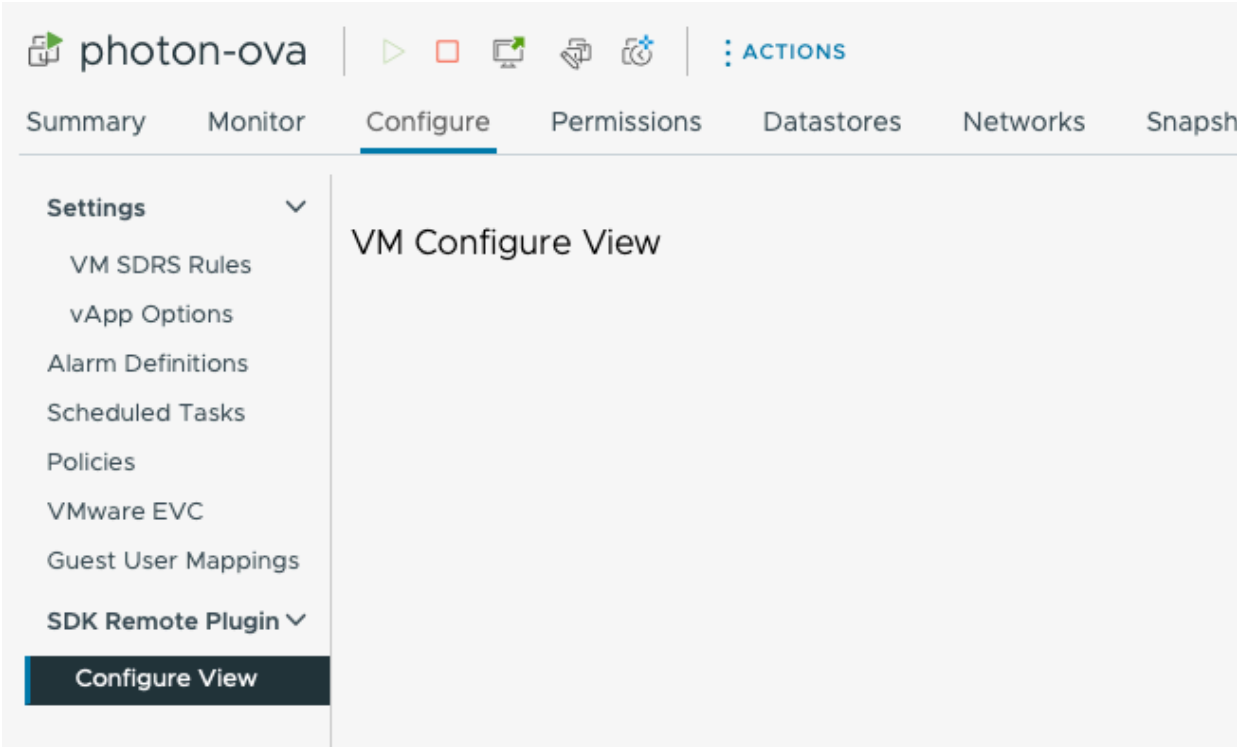


- **Configure Views**
  This comprises a single Configure category with one or more views. It can contain detailed configuration data and workflows relevant to the current object.

For example, it could show a list of virtual machines on a selected host that should be backed up.

The following illustration shows the screen context for an Object Configure view

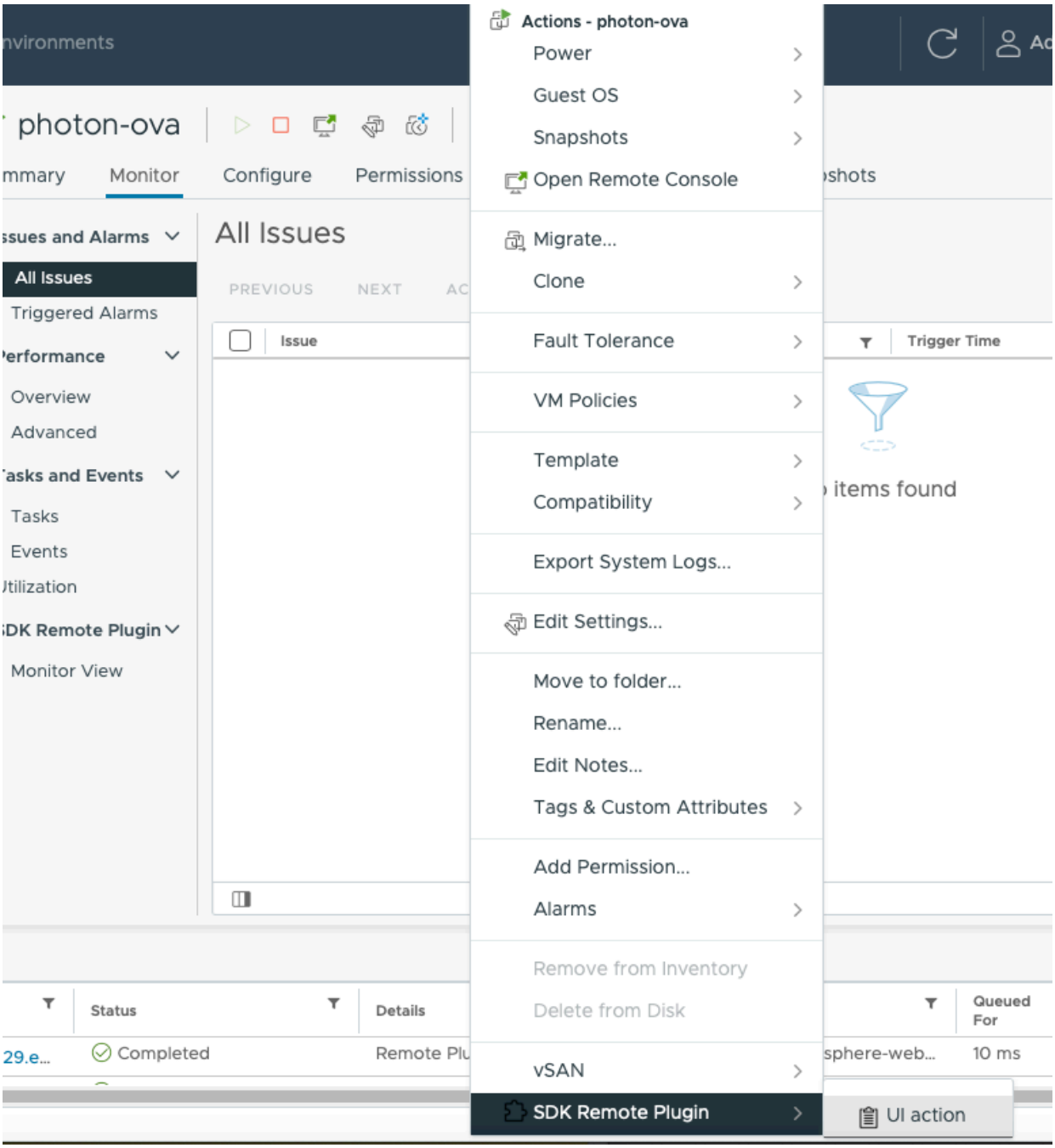
Figure 145: Object Configure View Screen Context



– **Menu**
This is a single plug-in solution menu with one or more actions. It contains actions that apply to the currently selected object, either navigating directly to a different view of the plug-in or else allowing user input in a modal dialog before the action runs. For example, you could use a Menu action to back up all virtual machines on a selected host.

The following illustration shows the screen context for a Menu Action.

Figure 146: Menu Action Screen Context



A plug-in solution submenu can contain horizontal separators, which allow you to group menu choices. To insert a separator into the submenu, insert the following element into the `menu.actions` array:

```
{ "type": "separator" },
```

NOTE
Separator elements at the beginning or end of the `menu.actions` array are ignored. A pair of adjacent separator elements produces only a single separator displayed on the screen.

Menu actions can also apply to a set of selected objects, under the following conditions:

- All selected objects are managed by the same vCenter Server instance.
- All selected objects have the same type, such as `HostSystem` or `VirtualMachine`.
- The plug-in manifest specifies `acceptsMultipleTargets` in the configuration for the selected object type.

If all of these conditions are true when you select multiple objects, you can apply a menu action to all of the selected objects. If any of the conditions is untrue, for instance, if you select both a `VirtualMachine` object and a `HostSystem` object, the menu shows no actions for the plug-in.

To enable selecting multiple targets, set `acceptsMultipleTargets` to `true` in the menu action configuration for the type or types that should allow it. For example, a configuration for `HostSystem` objects could contain the following lines:

```
...
  "objects": { ...
    "HostSystem": { ...
      "menu": {
        "actions": [
          {
            "id": "TakeAction",
            "labelkey": "Take action",
            "acceptsMultipleTargets": true,
            "trigger": {
              "type": "modal",
            ...
```

NOTE
Headless actions that navigate to a different view for the same object and served by the same remote plug-in are also supported. Specify these headless actions by setting the `trigger.type` to `"navigation"`.

Remote Plug-in Manifest Example

The following JSON code is an example of a plug-in manifest file that demonstrates how to specify some of the vSphere Client SDK extension points for your plug-in.

```
{
  "manifestVersion": "1.0.0",
  "requirements": {
    "plugin.api.version": "1.0.0"
  },
  "configuration": {
```

```
    "nameKey": "My Plugin"
  },
  "global": {
    "view": {
      "navigationId": "myGlobalViewId",
      "uri": "myplugin/globalView.html",
      "navigationVisible": false
    }
  },
  "objects": {
    "Datacenter": {
      "summary": {
        "view": {
          "uri": "myplugin/summary.html",
          "icon": {
            "name": "main"
          }
        }
      },
      "monitor": {
        "views": [
          {
            "navigationId": "myview1",
            "labelKey": "category.view1",
            "uri": "myplugin/view1.html"
          }
        ]
      },
      "configure": {
        "views": [
          {
            "navigationId": "myview1",
```

```
      "labelKey": "category.view1",
      "uri": "myplugin/view1.html"
    }
  ]
},
"menu": {
  "actions": [
    {
      "labelKey": "action1",
      "icon": {
        "name": "action-1"
      },
      "trigger": {
        "type": "modal",
        "uri": "myplugin/modal-action.html",
        "titleKey": "myplugin modal title",
        "accessibilityTitleKey": "myplugin modal accessibility title",
        "size": {
          "height": 250,
          "width": 600
        }
      }
    },
    {
      "type": "separator"
    },
    {
      "labelKey": "Action2",
      "icon": {
        "name": "action-2"
      },
      "trigger": {
```

```
            "type": "navigation",
            "targetViewId": "myGlobalViewId"
          }
        }
      ]
    }
  }
},
"definitions": {
  "iconSpriteSheet": {
    "uri": "myplugin/images/icon-sprite.png",
    "definitions": {
      "main": {
        "x": 0,
        "y": 0
      }
    }
  },
  "i18n": {
    "locales": [
      "en-US",
      "de-DE",
      "fr-FR"
    ],
    "definitions": {
      "category.view1": {
        "en-US": "Monitor View 2",
        "de-DE": "Monitoransicht 2",
        "fr-FR": "Vue Surveiller 2"
      }
    }
  }
```

```
   }
}
```

Dynamic Extensions for Remote Plug-ins

By default, plug-in views and menu items display in the vSphere Client user interface unconditionally. The vSphere Client supports a Service Provider Interface (SPI) that allows plug-in servers to filter views or menu items so they display conditionally. Conditional extensions are known as dynamic extensions.

Dynamic extensions are supported for Summary, Monitor, and Configure views, and for Menu actions. The vSphere Client SDK does not currently support dynamic extensions for Global views.

Dynamic Extension Use Cases

The vSphere Client supports dynamic extensions for remote plug-ins with an SPI that allows a plug-in server to determine at run time which views display in the vSphere Client user interface. This allows the plug-in to use any basis for the decision.

The dynamic extension SPI gives you flexibility to choose and implement your own filter processing in the plug-in server. For example, a plug-in could choose to display or hide an extension based on user authorization or object state.

How the vSphere Client Displays Dynamic Extensions

The vSphere Client determines whether to show a dynamic extension by asking a plug-in server at the time the context makes the extension relevant. The vSphere Client identifies the relevant extensions and sends a request to each plug-in server that makes visibility choices for the relevant extensions. Each plug-in server collects the information needed to decide whether to hide or show its extensions, and responds with its visibility choice for each of its relevant extensions.

As of vSphere 8.0 U3, dynamic plug-in actions can be hidden or disabled depending on response from the plug-in server, which can return items with two properties: `visible` or `relevant`.

- Setting the `visible` property to `false` has a different effect on actions than on Cards, Monitor or Configure views. For Cards, Monitor views, or Configure views, `visible=false` means that the view is not shown at all. For actions, `visible=false` makes the action visible but disabled.
- Setting the `relevant` property to `false` makes the action disappear from the menu. The `relevant` property is not required because it is `true` by default. All actions are deemed relevant and get enabled or disabled by the `visible` property. To remove an action, the plug-in should return a response with `relevant=false` and `visible=false`.

Dynamic Views

While the vSphere Client waits for answers, it might display views associated with static extensions, as well as indications that it is waiting for more data. If a plug-in has any dynamic extensions that are relevant in the current context, the vSphere Client will display the plug-in category name in the items list, with a spinner beside it indicating that the display is not yet complete. The vSphere Client will not display the view names in the plug-in category until the list has stabilized.

Dynamic Cards

If a card on the Summary tab is configured to be dynamic, the vSphere Client queries the associated plug-in server for visibility choices. The Summary tab displays only static cards while the vSphere Client waits for a response. After the response arrives, the vSphere Client adds to the display any dynamic cards that the response indicates should be visible.

Dynamic Actions

If a plug-in has dynamic action extensions in the Action menu, the vSphere Client creates place-holders in the menu until the plug-in response indicates whether or not the plug-in actions should be visible. While waiting for a response, the

vSphere Client displays the plug-in name in the Actions menu, and puts a brief message in the plug-in submenu that indicates the client code is still loading information. When the plug-in response indicates that an action should be visible, the vSphere Client enables the action in the plug-in submenu. As of 8.0 U3, the plug-in can disable an action with `visible=false` or remove it with `relevant=false` also.

Direct Links to Dynamic Views

If a user clicks a link to a dynamic view, the vSphere Client might need to query the plug-in server for the view's visibility. While waiting for the response, the vSphere Client might open an iFrame for the dynamic view, with only a spinner and a message that the view is not yet ready. If the response indicates that the dynamic view should be hidden, the vSphere Client displays a message saying that the view is not available.

Direct Links to Static Views

If a user clicks a link to a static view, the vSphere Client displays the static view immediately. If there are also dynamic views that might apply, the client code marks the plug-in name in the category list with a spinner while it queries the plug-in server for its visibility choices.

Time-outs

In case the plug-in server does not return a timely response, the vSphere Client will time out and cancel the request. When the vSphere Client times out a request for dynamic extension visibility, it assumes the extension should be hidden.

Caching Dynamic Extension Visibility

When the user navigates to a context (such as a **Monitor** tab) that has the potential to show dynamic extensions, the client code queries the `dynamicUri` for filter choices. The plug-in server returns its visibility choices for the dynamic extensions that apply to the user context. The client code caches the dynamic view choices to speed navigation between the views that pertain to the active vSphere object.

The client code caches visibility choices for only those dynamic views that apply to the current object. For example, if the user navigates to a virtual machine's **Monitor** tab and the plug-in server returns visibility choices for both a view on the **Monitor** tab and a view on the **Configure** tab that apply to the current virtual machine, the client code caches choices for both tabs. The client code does not cache visibility choices for dynamic actions.

The visibility choices for the current object generally remain in cache as long as the user navigates only to tabs for that object. The client code empties the cache when the user navigates to a different vSphere object. Then the client builds up the cache for dynamic views that pertain to the new object.

> **NOTE**
> The global refresh button clears the cache and queries the applicable `dynamicUri` for filter choices.

Configure Dynamic Extensions

You configure dynamic extensions for your plug-in by using properties in the `plugin.json` manifest file. These properties identify the dynamic extensions and provide SPI endpoints for queries from the client code.

Dynamic Monitor and Configure extensions are configured in the same way. Dynamic Summary extensions and dynamic Menu extensions are similar. Where the steps differ, the examples show the differences.

1. Edit the manifest file, `plugin.json`.

```
{ "manifestVersion": "1.0.0",

  "requirements": {"plugin.api.version": "1.0.0"}

  ...
```

2. Within the `objects` object, locate the type of vSphere managed object for which the plug-in can supply a dynamic extension.

```
{ "manifestVersion": "1.0.0",
  "requirements": {"plugin.api.version": "1.0.0"}
  ...
  "objects": { ...
    "VirtualMachine": {
      ...
```

3. Within the managed object type, locate the extension type for which the plug-in can supply a dynamic extension.

For a dynamic extension in the **Summary** tab.	`...`
	`"objects": { ...`
	`"VirtualMachine": { ...`
	`"summary": {`
	`...`

For a dynamic extension in the **Monitor** tab.	`...`
	`"objects": { ...`
	`"VirtualMachine": { ...`
	`"monitor": {`
	`...`

For a dynamic extension in the **Configure** tab.	`...`
	`"objects": { ...`
	`"VirtualMachine": { ...`
	`"configure": {`
	`...`

For a dynamic extension in the **Actions** menu.	`...`
	`"objects": { ...`
	`"VirtualMachine": { ...`
	`"menu": {`
	`...`

4. Within the object representing the extension type object, add a `dynamicUri` property to identify the endpoint where the client code will query the plug-in server for its filter choices.

   If the plug-in manifest server will supply filter choices, you can specify a string containing the URI path. If an auxiliary server will supply filter choices, specify a JSON object containing a `path` property (type `string`) and a `serverType` property (type `string`), where the `serverType` corresponds to the `ServerInfo.type` property in

the Extension Manager registration record. The `serverType` value maps to the auxiliary server base URI that the user interface code uses to construct the endpoint for the dynamic extension filter query.

NOTE
When you register an auxiliary server that will supply filter choices, its `ServerInfo.url` property in the `Extension` record must end with a slash (`/`). See Registering Auxiliary Plug-in Servers.

For a dynamic extension in the **Summary** tab.	`...`
	`"objects": { ...`
	`"VirtualMachine": { ...`
	`"summary": {`
	`"dynamicUri": "rest/dynamics/vm/summary",`
	`...`

(if an auxiliary server handles filter queries)	`...`
	`"objects": { ...`
	`"VirtualMachine": { ...`
	`"summary": {`
	`"dynamicUri": {`
	`"serverType": "DYNAMIC_AUX_SERVER",`
	`"path": "rest/dynamics/vm/summary"`
	`}, ...`

For a dynamic extension in the **Monitor** tab.	`...`
	`"objects": { ...`
	`"VirtualMachine": { ...`
	`"monitor": {`
	`"dynamicUri": "rest/dynamics/vm/monitor",`
	`...`

(if an auxiliary server handles filter queries)	`...`
	`"objects": { ...`
	`"VirtualMachine": { ...`
	`"monitor": {`
	`"dynamicUri": {`

Table continued on next page

Continued from previous page

	`"serverType": "DYNAMIC_AUX_SERVER",`
	`"path": "rest/dynamics/vm/monitor"`
	`},`
	`...`

For a dynamic extension in the **Configure** tab.	`...`
	`"objects": { ...`
	`"VirtualMachine": { ...`
	`"configure": {`
	`"dynamicUri": "rest/dynamics/vm/configure",`
	`...`

(if an auxiliary server handles filter queries)	`...`
	`"objects": { ...`
	`"VirtualMachine": { ...`
	`"configure": {`
	`"dynamicUri": {`
	`"serverType": "DYNAMIC_AUX_SERVER",`
	`"path": "rest/dynamics/vm/configure"`
	`},`
	`...`

For a dynamic extension in the **Actions** menu.	`...`
	`"objects": { ...`
	`"VirtualMachine": { ...`
	`"menu": {`
	`"dynamicUri": "rest/dynamics/vm/actions",`
	`...`

(if an auxiliary server handles filter queries)	`...`
	`"objects": { ...`
	`"VirtualMachine": { ...`

Table continued on next page

Continued from previous page

	<pre>"menu": {
	"dynamicUri": {
	"serverType": "DYNAMIC_AUX_SERVER",
	"path": "rest/dynamics/vm/actions"
	},
	...</pre>

For more information about the `ServerInfo.type` property, see Registering Auxiliary Plug-in Servers.

5. At the same level as the `dynamicUri` property, locate the `views` array or the `view` object (for Summary extensions), the `views` array (for Monitor or Configure extensions) or the `actions` array (for Menu extensions)

For a single dynamic extension in the **Summary** tab.	<pre>...
	"objects": { ...
	"VirtualMachine": { ...
	"summary": {
	"dynamicUri": "rest/dynamics/vm/summary",
	"view": {
	...</pre>

For multiple dynamic extensions in the Summary tab.	<pre>...
	"objects": { ...
	"VirtualMachine": { ...
	"summary": {
	"dynamicUri": "rest/dynamics/vm/summary",
	"views": [
	...
	]
	}
	...</pre>

For a dynamic extension in the **Monitor** tab.	<pre>...
	"objects": { ...
	"VirtualMachine": { ...</pre>

Table continued on next page

Continued from previous page

	`"monitor": {`
	`"dynamicUri": "rest/dynamics/vm/monitor",`
	`"views": [`
	`...`

For a dynamic extension in the **Configure** tab.	`...`
	`"objects": { ...`
	`"VirtualMachine": { ...`
	`"configure": {`
	`"dynamicUri": "rest/dynamics/vm/configure",`
	`"views": [`
	`...`

For a dynamic extension in the **Actions** menu.	`...`
	`"objects": { ...`
	`"VirtualMachine": { ...`
	`"menu": {`
	`"dynamicUri": "rest/dynamics/vm/actions",`
	`"actions": [`
	`...`

6. To each view or action that will be treated as dynamic, add the `dynamic` property.

For a single dynamic extension in the **Summary** tab.	`...`
	`"objects": { ...`
	`"VirtualMachine": { ...`
	`"summary": {`
	`"dynamicUri": "rest/dynamics/vm/summary",`
	`"view": {`
	`"dynamic": true`
	`...`

For multiple dynamic extensions in the Summary tab.	`...` `"objects": { ...`

Table continued on next page

Continued from previous page

	<pre>"VirtualMachine": { ...
	"summary": {
	"dynamicUri": "rest/dynamics/vm/summary",
	"views": [
	{
	"dynamic": true,
	...
	},
	{
	"dynamic": true,
	...
	}
	]
	}
	...</pre>

For a dynamic extension in the **Monitor** tab.	<pre>...
	"objects": { ...
	"VirtualMachine": { ...
	"monitor": {
	"dynamicUri": "rest/dynamics/vm/monitor",
	"views": [
	{
	"dynamic": true,
	...</pre>

For a dynamic extension in the **Configure** tab.	<pre>...
	"objects": { ...
	"VirtualMachine": { ...
	"configure": {
	"dynamicUri": "rest/dynamics/vm/configure",</pre>

Table continued on next page

Continued from previous page

	<pre>"views": [
	{
	"dynamic": true,
	...</pre>

For a dynamic extension in the **Actions** menu.	<pre>...
	"objects": { ...
	"VirtualMachine": { ...
	"menu": {
	"dynamicUri": "rest/dynamics/vm/actions",
	"actions": [
	{
	"dynamic": true,
	...</pre>

7. Add an `id` property to each Summary card or Menu action, and a `navigationId` to each Monitor or Configure view, if not already present.

For a single dynamic extension in the **Summary** tab.	<pre>...
	"objects": { ...
	"VirtualMachine": { ...
	"summary": {
	"dynamicUri": "rest/dynamics/vm/summary",
	"view": {
	"dynamic": true
	"id": "vmSummaryCard",
	...</pre>

For multiple dynamic extensions in the Summary tab.	<pre>...
	"objects": { ...
	"VirtualMachine": { ...
	"summary": {
	"dynamicUri": "rest/dynamics/vm/summary",</pre>

Table continued on next page

Continued from previous page

	<pre>"views": [
	{
	"dynamic": true,
	"id": "summary-card-1",
	...
	},
	{
	"dynamic": true,
	"id": "summary-card-2",
	...
	}
	]
	}
	...</pre>

For a dynamic extension in the **Monitor** tab.	<pre>...
	"objects": { ...
	"VirtualMachine": { ...
	"monitor": {
	"dynamicUri": "rest/dynamics/vm/monitor",
	"views": [
	{
	"dynamic": true,
	"navigationId": "MonitorDynView",
	...</pre>

For a dynamic extension in the **Configure** tab.	<pre>...
	"objects": { ...
	"VirtualMachine": { ...
	"configure": {
	"dynamicUri": "rest/dynamics/vm/configure",</pre>

Table continued on next page

Continued from previous page

	"views": [{ "dynamic": true, "navigationId": "ConfigureDynView", ...

For a dynamic extension in the **Actions** menu.	...
	"objects": { ...
	"VirtualMachine": { ...
	"menu": {
	"dynamicUri": "rest/dynamics/vm/actions",
	"actions": [
	{
	"dynamic": true,
	"id": "TakeAction",
	...

A `navigationId` or `id` string is required to identify the dynamic view or action in the plug-in server's response to a filter query.

The following example shows a `plugin.json` file with a dynamic Summary card, a dynamic Monitor view, a fixed Configure view, and a dynamic action. The dynamic Summary view is filtered by the manifest server, which is the default when a specific server is not named. The dynamic Monitor view is filtered by the the manifest server, configured with an explicit reference. The dynamic menu action is filtered by an auxiliary server.

plugin.json With Dynamic Extensions

```
{ "manifestVersion": "1.0.0",
  "requirements": {"plugin.api.version": "1.0.0"},
  "configuration": {
    "nameKey": "Dynamic Extension Manifest Example"
  }
  "objects": {
    "VirtualMachine": {
      "summary": {
```

```
    "dynamicUri": "rest/dynamics/vm/summary",
    "view": {
      "dynamic": true
      "id": "vmSummaryCard",
      "uri": "index.html#vm-portlet",
    }
  },
  "monitor": {
    "dynamicUri": {
      "serverType": "MANIFEST_SERVER",
      "path": "rest/dynamics/vm/monitor"
    },
    "views": [
      {
        "dynamic": true,
        "navigationId": "MonitorDynView",
        "uri": "rest/views/vm/monitor/index.html"
      }
    ],
  "configure": {
    "views": [
      {
        "navigationId": "ConfigureFixView",
        "uri": "rest/views/vm/configure/index.html"
      },
  "menu": {
    "dynamicUri": {
      "serverType": "DYNAMIC_AUX_SERVER",
      "path": "rest/dynamics/vm/actions"
    },
    "actions": [
      {
```

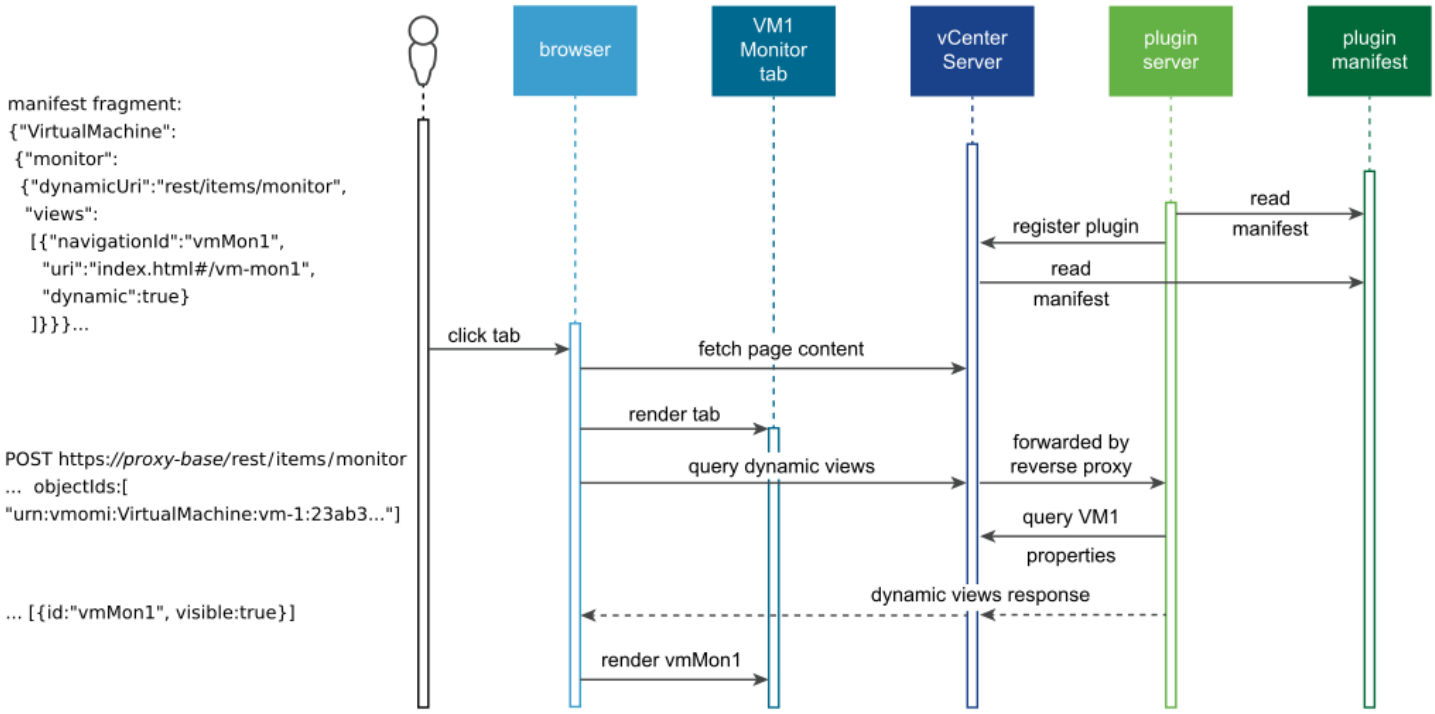
```
            "dynamic": true,
            "id": "TakeAction",
            "labelKey": "Take action",
            "trigger": {
              "type": "modal",
              "uri": "rest/actions/vm/action1.html"
            }
          }
        ]
      }
    }
  }
}
```

Implement server-side code to handle filter queries at the endpoints that you configured in the manifest file. For more information about filter queries, see Dynamic Extensions Filter Query.

Dynamic Extensions Filter Query

The `vsphere-ui` process parses the plug-in manifest when the plug-in is registered wih the Extension Manager of vCenter Server, and stores the information as long as the plug-in remains registered. When a user navigates to a Summary tab, Monitor tab, or Configure tab, or when a user clicks the Actions menu, the user interface code in the browser sends a filter query to a plug-in server to determine its choices for dynamic extensions. The plug-in server responds with a visibility choice for each dynamic extension that applies in the current context.

An example message sequence for a filter query is illustrated in the following diagram.



Notes about the sequence diagram:

- Reading the manifest file is not part of the filter query sequence. The diagram includes the manifest fragment for context.
- The variable *`proxy-base`* in the endpoint URL refers to the proxied plug-in server base URL that the client code prefixes to the `dynamicUri` in the manifest file. All client queries to the plug-in server pass through the reverse proxy for security reasons. For more information, see Client-Server Communications with Remote Plug-ins.

The client filter query consists of an HTTPS POST request to a `dynamicUri` endpoint specified in the plug-in manifest file. The client code chooses the URL configured for the vSphere object type and the extension type that are currently active in the user interface.

> **TIP**
> A best practice is to maintain a different endpoint for each extension type (Summary, Monitor, Configure, or Menu) and return visibility choices (`true` or `false`) for only the relevant dynamic extensions.

The request header contains:

- The body type specifications (`'application/json'` for both request and response).
- HTTP cache control settings.
- Identity and authentication headers:
  - Node ID (unused)
  - The vsphere-ui endpoint URL This tells the plug-in server which vCenter Server manages the context objects.
  - The session token for the client's authentication with the vsphere-ui endpoint. This allows the plug-in server to authenticate with vCenter Server to retrieve properties needed to make filtering choices.

For example:

```
'content-type': 'application/json',

'Accept': 'application/json',

'Cache-Control': 'no-cache, no-store, max-age=0',

vmware-api-session-id: c621c819-4f65-1b02-2214-c7ac159ad4d4
```

```
vmware-api-ui-endpoint-url: https://93.184.216.34/api/ui

vmware-api-ui-node-id: 6079314c-d525-43a1-8a54-735f5417f11e
```

The request body contains:

- The API version of the filter query protocol, used to negotiate message format with the plug-in server.
- The locale code.
- A list of object IDs for the current context objects. For dynamic actions, the list of object IDs may be any length. For dynamic views, the list may contain only the object ID that pertains to the **Summary** tab or **Monitor** tab or **Configure** tab that the client is rendering.

For example:

```
apiVersion: "1.0.0"

objectIds: ["urn:vmomi:VirtualMachine:vm-1005:27a09c68-d1d6-4fe2-a28f-616949f30930"]

locale: "en-US"
```

The response from the plug-in server must be a single anonymous JSON object that contains:

- The latest version of the filter query protocol that the server is prepared to handle, used to negotiate message format with the client.
- A list of dynamic items, where each item is an object that contains:
  - The `id` or `navigationId` of a dynamic view or the `id` of a dynamic menu action. A `navigationId` value is assigned to the `id` property in the response object.
  - Boolean values `visible` to control whether the dynamic extension displays in the user interface, and (as of 8.0 U3) `relevant` to control whether to remove it from the interface.

For example:

```
{
  apiVersion: "1.0.0",
  dynamicItems:
  [
    {
      id: "MonitorDynView",
      visible: true,
      relevant: true
    },
    {
      id: "DeleteAction",
      visible: false,
      relevant: false
    }
  ]
}
```

Example Code for Filtering Dynamic Extensions

Plug-ins that support dynamic extensions must implement the SPI that the client code uses to determine which dynamic extensions to display in the browser. The plug-in server can use any criterion chosen by the plug-in developer. The following example illustrates server code that uses two kinds of criteria to decide whether dynamic extensions should be visible..

Two useful criteria for filtering dynamic extensions are properties of vSphere objects and the authorization level of the vSphere Client user. This example tests the run state of a virtual machine and the user's authorization with respect to the virtual machine.

This example depends on a manifest file that configures two dynamic extensions. One extension provides a dynamic Monitor view of a virtual machine. The other extension provides a dynamic Menu action to run a virtual machine.

The following example code illustrates a controller that serves dynamic extension endpoints. The controller accesses the JSON in the request body by using the DynamicItemsRequestModel object passed in for the specific REST endpoint.

```
package com.example.remote.controllers;

import java.util.ArrayList;

import java.util.List;

import org.springframework.http.MediaType;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RestController;


import com.vmware.sample.remote.model.DynamicItem;

import com.vmware.sample.remote.model.DynamicItemsRequestModel;

import com.vmware.sample.remote.model.PluginServerDynamicItemsResponse;

import com.vmware.sample.remote.vim25.services.AuthorizationService;


/**

* Provide public endpoints for vSphere Client to query about

* UI visibility for dynamic views/actions.

*/

@RestController

@RequestMapping(value = "/dynamicItems",

                method = RequestMethod.POST,

                consumes = MediaType.APPLICATION_JSON_VALUE,

                produces = MediaType.APPLICATION_JSON_VALUE)
```

```
public class DynamicItemsController {

   private static final String MANAGE_VM_PRIVILEGE =
"com.vmware.sample.remote.1.0.0.ManageVm";

   private final AuthorizationService authorizationService;

   private final VmRunstateService vmRunstateService;


   public DynamicItemsController(final AuthorizationService authorizationService) {

      this.authorizationService = authorizationService;

      this.vmRunstateService = vmRunstateService;

   }


   /* This action should be visible if user has authorization. */

   @RequestMapping(value = "/vm/actions")

   public PluginServerDynamicItemsResponse retrieveVmActions(

      @RequestBody DynamicItemsRequestModel payload) {

      final boolean hasPrivilege = authorizationService

            .hasPrivilege(payload.objectIds, MANAGE_VM_PRIVILEGE);

      final List<DynamicItem> dynamicItems = new ArrayList<>();

      dynamicItems.add(new DynamicItem("TakeAction", hasPrivilege));

      return new PluginServerDynamicItemsResponse("1.0.0", dynamicItems);

   }


   /* This VM view should be visible only if it is NOT currently running. */

   @RequestMapping(value = "/vm/monitor")

   public PluginServerDynamicItemsResponse retrieveVmMonitorViews(

      @RequestBody DynamicItemsRequestModel payload) {

      final String objectId = payload.objectIds.get(0);

      final boolean makeVisible = !isVmStateRunning(objectId));

      final List<DynamicItem> dynamicItems = new ArrayList<>();

      dynamicItems.add(new DynamicItem("MonitorDynView", makeVisible));

      return new PluginServerDynamicItemsResponse("1.0.0", dynamicItems);

   }

}
```

vSphere Client Plug-in User Interface Modules

The vSphere Client provides several JavaScript interfaces that your plug-in can use to communicate with the vSphere Client platform. These JavaScript methods are documented here as if they have TypeScript signatures, but they run as pure JavaScript, and all complex types are plain old JavaScript objects.

The plug-in web application runs in a separate iFrame which is part of the vSphere Client. The iFrame content is rendered from the web application server of the plug-in back end.

> **NOTE**
> Do not access the `window.parent`, which belongs to the vSphere Client. Do not access the internal JavaScript or CSS resources of the vSphere Client. Such access is unsupported and could cause your plug-in to fail in a future release of the vSphere Client.

Bootstrapping the JavaScript API

The vSphere Client loads plug-in resources in a tenant iFrame. The plug-in must load a thin JavaScript library to support communication with the parent window in the vSphere Client. The library implements a JavaScript API that the plug-in front-end code uses to manage resources outside its iFrame.

To bootstrap the Client Library the following script should be added to all HTML pages in the plug-in:
`<script type="text/javascript"src="/api/ui/htmlClientSdk.js"></script>`

After you load the script, you initialize the `htmlClientSdk` object, by invoking the `htmlClientSdk.initialize()` method. Before you initialize, you can only invoke the methods of the `htmlClientSdk` interface. After you initialize the `htmlClientSdk` object, you can invoke any of the methods in the JavaScript API.

If you use frameworks such as jQuery, or zone.js with Angular, you only need to initialize the `htmlClientSdk` object once. You should initialize as early as possible, so that the `htmlClientSdk` functions will be available to all plug-in user interface components.

> **NOTE**
> Do not use any communication method not provided by the APIs. Do not access any internal JavaScript or CSS resources of the vSphere Client. Doing so is unsupported because the implementation of the `htmlClientSdk` functions can change in future releases of the vSphere Client.

vSphere Client JavaScript API: htmlClientSdk Interface

The htmlClientSdk object provides access to all the JavaScript API interfaces. You load and initialize the htmlClientSdk first in your plug-in views.

`htmlClientSdk.getProxiedPluginServerOrigin()`

Signature	`htmlClientSdk.getProxiedPluginServerOrigin():string`
Description	Returns a proxy URL for the plug-in manifest server root. The URL contains the protocol, domain, and port of the vCenter Server, with a proxy path that routes to the plug-in manifest server root. This URL is useful to set a root-relative `base` tag for relative URLs in a front-end

Table continued on next page

Continued from previous page

	framework such as Angular. This method can be safely called before the SDK has been initialized.

`htmlClientSdk.initialize()`

Signature	`htmlClientSdk.initialize(callback:function):void`
Description	Initializes the htmlClientSdk object and invokes the callback function when initialization is complete and the other JavaScript interfaces are available. After the first call, subsequent calls act to register additional callbacks.
Parameter: `callback`	The optional callback function must have the following signature: `function callback():void` The callback function has access to all the functions in the JavaScript API. You use the callback to code logic for initializing plug-in state.

`htmlClientSdk.isInitialized()`

Signature	`htmlClientSdk.isInitialized():boolean`
Description	Tests whether the htmlClientSdk object has been initialized.

vSphere Client JavaScript API: Modal Interface

The `modal` interface enables your plug-in to manage modal dialog windows.

> **NOTE**
> This SDK uses the Structured Clone Algorithm, which has limitations as described in the Mozilla Developer Network (MDN) web doc. Limitations affect Modal, Confirmation Modal, and Shared Modal APIs.

`modal.AlertLevel`

Description	Enum: • `SUCCESS` • `INFO` • `WARNING` • `DANGER` Clarity adds a standard icon, depending on the alert level. See https://v2.clarity.design/alerts.

Used by:
modal.ConfirmationModalConfig

modal.ButtonStyle

Description	Enum: • SUCCESS • INFO • WARNING • DANGER Clarity adds default CSS tags, depending on the action importance: • btn-success • btn-primary • btn-warning • btn-danger For examples of Clarity button styling, see the Solid Buttons illustrations at https://v2.clarity.design/buttons/#examples.

Used by:
modal.ModalButton

modal.ButtonType

Description	Enum: • PRIMARY • SECONDARY Clarity styles PRIMARY buttons with greater emphasis. See https://v2.clarity.design/buttons.

Used by:
modal.ModalButton

modal.close()

Signature	Closes the modal dialog box in the parent iframe.
Description	Optional data that will be passed to callback function specified by onClosed property at dialog open.
Parameter: data	Optional data that will be passed to callback function specified by onClosed property at dialog open.

NOTE
No data is passed to the callback function if the user clicks the dialog close box.

modal.ConfirmationModalConfig

Description	Specifies the properties of a confirmation modal dialog box.

Property	Type	Required?	Notes
`content`	`string`	yes	Confirmation message displayed in the dialog.
`buttons`	modal.ModalButton `[]`	yes	Buttons displayed in the dialog footer. (min 1, max 4)
`title`	`string`	no	Dialog main title. May not contain an icon. (default=`''`)
`AccessibilityTitle`	`string`	no	Used when dialog title is not displayed, such as in a wizard dialog.
`size`	modal.ModalSize	no	Width of the dialog box. (Default width is chosen by Clarity. Height is fixed.)
`closable`	`boolean`	no	Whether the dialog displays a close button. (default=`true`)
`onClosed`	`function(result:any):void`	no	Function runs when user closes the dialog. If modal.close() is called, its parameter is in turn passed to the `onClosed()` function.
`secondaryTitle`	`string`	no	Optional subtitle for the dialog.
`alertLevel`	modal.AlertLevel	no	Causes Clarity to add an icon to the dialog.

Used by:
modal.openConfirmationModal()

`modal.DynamicModalConfig`

Description	Specifies values for some properties of a modal dialog box.

Property	Type	Required?	Notes
`title`	`string`	no	Dialog title. May not contain an icon. (If not present, no change to dialog title.)
`accessibilityTitle`	`string`	no	Used when dialog title is not displayed, such as in a wizard dialog.
`height`	`number`	no	Dialog height, specified in pixels. (If not present, no change to dialog height.)

Used by:
modal.setOptions()

`modal.getCustomData()`

Signature	`modal.getCustomData():any`

Table continued on next page

Continued from previous page

Description	Returns the `customData` object provided when a modal dialog box was opened, or `null` if no `customData` object was provided.

`modal.getSharedModalsMap()`

Signature	`modal.getSharedModalsMap(targetPluginId:string, callback:function):void`
Description	Creates listings of modal dialog boxes shared by a specific plug-in and accessible to the calling plug-in. The callback function must have the following signature: `function callback(sharedModalsMap:`vsphere-client-javascript-api-modal-interface.dita#O2_wont_generate_ID_202208201257-en`):void`

`modal.ModalButton`

Description	Describes button properties.

Property	Type	Required?	Notes
`label`	`string`	yes	
`type`	modal.ButtonType	no	
`style`	modal.ButtonStyle	no	
`callback`	`function(): void`	no	

Used by:
modal.ConfirmationModalConfig

`modal.ModalConfig`

Description	Specifies the properties of a modal dialog box.

Property	Type	Required?	Notes
`url`	`string`	yes	Location of HTML content for the dialog.
`title`	`string`	no	Dialog title. May not contain an icon. (default=`''`)
`accessibilityTitle`	`string`	no	Used when dialog title is not displayed, such as in a wizard dialog.
`size`	`(width:number, height:number)`	no	Specify in pixels. Default is chosen by Clarity. See https://v2.clarity.design/modals.
`closable`	`boolean`	no	Whether the dialog displays a close button. (default=`true`)

Table continued on next page

Continued from previous page

Property	Type	Required?	Notes
`onClosed`	`function(result:any):void`	no	Function runs when the dialog closes. If `modal.close()` is called, its parameter is in turn passed to the `onClosed()` function. Function runs when the dialog closes. If modal.close() is called, its parameter is in turn passed to the `onClosed()` function.
`customData`	`any`	no	Data the calling module passes to the dialog.
`contextObjects`	`any[]`	no	IDs of relevant objects the calling module passes to the dialog.

Used by:
modal.open()

`modal.ModalSize`

Description	Width of a modal dialog box.

Property	Type	Required?	Notes
`width`	`number`	yes	Specified in pixels.

Used by:
modal.ConfirmationModalConfig

`modal.open()`

Signature	`modal.open(configObj:`
	modal.ModalConfig
	`):void`

Description	Opens a modal dialog box specified by the `configObj` parameter.
Parameter: `configObj`	Specifies the properties of this modal dialog box.

`modal.openConfirmationModal()`

Signature	`modal.openConfirmationModal(configObj:`
	modal.ConfirmationModalConfig
	`):void`

Description	Opens a lightweight modal dialog box designed to present information and confirmation buttons.
Parameter: `configObj`	Specifies the properties of this modal dialog box.

modal.openSharedModal()

Signature	`modal.openSharedModal(configObj:` vsphere-client-javascript-api-modal-interface.dita#GUID-7A73F8A7-B0DD-4D6D-B39E-ACC27CD521F5-en `):void`
Description	Opens a modal dialog box that is defined and shared by another plug-in.

modal.SharedModalConfig

Description	Specifies the properties for opening a shared modal dialog.

Property	Type	Required?	Notes
`pluginId`	`string`	yes	ID of the plug-in that defines the dialog.
`sharedModalId`	`string`	yes	ID of the shared modal dialog.
`vcGuid`	`string`	yes	The GUID of the vCenter Server instance where the plug-in that defines the dialog is registered.
`objectId`	`string`	no	
`onClosed`	`function(result:any):void`	no	Function runs when user closes the dialog. If modal.close() is called, its parameter is in turn passed to the `onClosed()` function.
`contextObjects`	`any[]`	no	IDs of relevant objects the calling module passes to the dialog.
`customData`	`any`	no	Additional data the calling module passes to the dialog.

Used by: vsphere-client-javascript-api-modal-interface.dita#SECTION_07BB474C-C87E-4295-B0DB-BEC83BB99D0B-en

modal.setOptions()

Signature	`modal.setOptions(configObj:` vsphere-client-javascript-api-modal-interface.dita#section_100902d7-9c2f-453e-9418-5bcbef1de3da-en `):void`
Description	Called by the parent view to modify some properties for a modal dialog box in the parent iframe.
Parameter: `configObj`	Specifies values for some dialog box properties.

modal.SharedModalsMap

Description	Information about modal dialog boxes shared by a specific plug-in instance registered with one or more vCenter Server instances.

Property	Type	Required?	Notes
`sharedModalsInfoByVcGuid`	`[VcGuid:string]:` vsphere-client-javascript-api-modal-interface.dita#SECTION_B7F375B5-C49A-4117-A881-AC80B268F1A0-en	yes	Map of modal dialog boxes shared by a plug-in instance, listed by each vCester Server instance where the plug-in is registered.

Table continued on next page

Continued from previous page

Property	Type	Required?	Notes
sharedModalsInfoByVcObjects		no	Reserved for future use.

Used by: vsphere-client-javascript-api-modal-interface.dita#SECTION_4BDA1279-509B-427C-BDED-74272C8B0EBC-en

modal.VcSharedModalsInfo

Description	Information about modal dialog boxes shared by a plug-in instance registered with a specific vCenter Server instance.

Property	Type	Required?	Notes
pluginVersion	string	yes	The plug-in version registered with the vCenter Server instance.
sharedModalsIds	string[]	yes	List of modal dialog boxes shared by the plug-in.

Used by: vsphere-client-javascript-api-modal-interface.dita#O2_wont_generate_ID_202208201257-en

vSphere Client JavaScript API: Application Interface

The app interface provides context object information and helps your plug-in navigate and control the vSphere Client user interface.

app.ApiEndpoints

Description	Holds a nested object that contains the parsed elements of the plug-in URL.

Property	Type	Required?	Notes
uiApiEndpoint	app.UiApiEndpoint	info only	

Used by: app.getApiEndpoints()

app.ClientInfo

Description	Documents type and version of vSphere Client.

Property	Type	Required?	Notes
type	string	info only	The vSphere Client type (must be HTML).
version	string	info only	The vSphere Client version string.

Used by: app.getClientInfo()

app.ClientViewNavigationOptions

Description	Specifies a destination view that is owned by the vSphere Client.

Property	Type	Required?	Notes
targetviewid	string	no	Navigation ID of the destination view. • administration.ceip • cluster.configure.settings.drs • cluster.configure.settings.ha • cluster.configure.settings.quickstart • cluster.monitor.allIsues • folder.configure.keyProviders • host.configure.networking.physicalAdapters • host.summary • namespace.storage.persistentVolumeClaims • network.summary • profilesAndPolicies.vmStoragePolicies • vm.monitor.performance.overview • vm.summary • folder.configure.settings.general • workload.platform.supervisor.summary
objectid	string	no	ID of any object associated with the view. (For a global view, this field is not required.).

Used by app.navigateToClientView()

app.formatDateTime()

Signature	app.formatDateTime(instant:number, options: app.PluginDateTimeFormatOptions):string
Description	Returns a formatted string that contains a human-readable representation of the instant parameter. The instant parameter holds the number of milliseconds since the start of 1 January 1970 UTC.

app.getApiEndpoints()

Signature	app.getApiEndpoints(): app.ApiEndpoints
Description	Returns the URLs of the vsphere-ui service API endpoints available to plug-ins. For an example, see the sample remote plug-ins available in the vSphere Client SDK.

app.getClientInfo()

Signature	app.getClientInfo(): app.ClientInfo
Description	Returns type and version info for the vSphere Client.

app.getClientLocale()

Signature	app.getClientLocale():string
Description	Returns the current locale of the vSphere Client.

app.getContextObjects()

Signature	app.getContextObjects():any[]
Description	Returns the IDs of the current context objects, depending on the view or dialog from which the method is called.
Return value:	**for global view** Returns empty array. Global views have no associated vSphere objects. **for vSphere object** Returns a context item for the associated vSphere object. **for dialog opened by modal.open()** If dialog opened by htmlClientSdk. modal.open() , returns value of configObj.contextObjects (or empty array, if contextObjects undefined) **for dialog opened by plugin.json actions** If dialog opened by action defined in plugin.json, returns an array of action targets. A context item is a JavaScript object containing a single property, id:string. This is the ID of the associated vSphere object.

app.getNavigationData()

Signature	app.getNavigationData():any
Description	Returns the custom data passed to the view by the app.navigateTo() . (If no custom data passed, returns null.)

app.getPluginBackendInfo()

NOTE
Applies to remote plug-ins only.

Signature	app.getPluginBackendInfo(callback:function):void
Description	Creates listings of service endpoints registered for the running plug-in instance, enabling plug-in front ends to use the vCenter Server ExtensionManager as a service registry.

Table continued on next page

Continued from previous page

	The callback function must have the following signature:
	
	`Function callback(backendInfo:`
	`app.PluginBackendInfo`
	`):void`
	
	If the method throws an exception, the `callback` parameter will be `null`.

`app.getRemotePluginNavigationMap()`

Signature	`app.getRemotePluginNavigationMap(targetPluginId:string, callback:function):void`

Description	Returns view IDs that belong to a plug-in specified by the `targetPluginId` parameter. If the targetPluginId is not the ID of the caller, the method returns IDs of only the public views for the target plug-in. Returns information for all vCenter Server instances where the target plug-in is registered and enabled.
	
	The callback function must have the following signature:
	
	`function callback(navigationInfo:`
	`app.RemotePluginNavigationMap`
	`):void`
	
	If the method throws an exception, the `callback` parameter will be `null`.

`app.getSessionInfo()`

Signature	`app.getSessionInfo(callback:function):void`

Description	Retrieves and processes information about the client's authentication session.
	
	The callback function must have the following signature:
	
	`function callback(info:`
	`app.SessionInfo`
	`):void`
	
	If the method throws an exception, the `callback` parameter will be `null`.

`app.getTheme()`

Signature	`app.getTheme():` `app.PluginTheme`
Description	Retrieves information about the UI theme that is currently selected.

`app.navigateTo()`

Signature	`app.navigateTo(configObj:`
	`app.NavigationOptions`
	`):void`

Description	Navigates to a specified view, and optionally passes custom data to the view.
Parameter: `configObj`	Specifies the destination view and custom data.

app.navigateToClientView()

Signature	app.navigateTo(configObj: app.ClientViewNavigationOptions):void
Description	Navigates to a specified vSphere Client view.
Parameter: configObj	Specifies the destination view.

app.navigateToRemotePluginView()

Signature	app.navigateTo(configObj: app.RemotePluginViewNavigationOptions):void
Description	Navigates to a view implemented by another remote plug-in, or by another instance of the same plug-in deployed by a different vCenter Server instance. Optionally passes custom data to the view.
Parameter: configObj	Specifies the destination view and custom data.

NOTE
A best practice is to call app.getRemotePluginViewNavigationMap() before this procedure, to ensure that the navigation target is registered and enabled.

app.NavigationOptions

Description	Specifies a destination view and custom data for the view.

Property	Type	Required?	Notes
targetViewId	string	no	Navigation ID of the destination view. (For a remote plug-in, this property must identify a view created by the same plug-in.) Omit this to navigate to the last-used view of the object specified by objectId.
objectId	string	no	ID of any object associated with the view. Omit this to navigate to a global view.
customData	any	no	A custom data structure passed to the view.

Used by: app.navigateTo()

app.PluginBackendInfo

NOTE
Applies to remote plug-ins only.

Description	Contains two objects that list endpoint descriptors available to a given plug-in instance.

Property	Type	Required?	Notes
allPluginBackendServers	Array< app.PluginBackendServerI	info only	A list of plug-in server endpoint descriptors registered for a given plug-in instance. The list includes all registrations

Table continued on next page

Continued from previous page

Property	Type	Required?	Notes
	nfo >		within the same link group. The list is unordered and contains no duplicates.
backendServersPerVc	{[vcGuid:string]:Array< app.PluginBackendServerInfo >}	info only	A one-to-many mapping: vCenter Server GUID to a list of plug-in server endpoint descriptors registered with the vCenter Server instance on behalf of the running plug-in instance.

Used by: app.getPluginBackendInfo()

app.PluginBackendServerInfo

NOTE
Applies to remote plug-ins only.

Description	A descriptor for a plug-in server endpoint registered for a plug-in instance. Choose the server by the `type` value, then form a resource URL from the endpoint descriptor. To access a resource belonging to the server, prefix the `proxiedBaseUrl` value to the root-relative path of the resource on the plug-in server: /<proxiedBaseUrl>/<path to resource from server root>

Property	Type	Required?	Notes
proxiedBaseUrl	string	info only	The path component of the server root URL, as seen on the reverse proxy service port.
type	string	info only	The `type` of the server, as specified in its registration record (`Extension` data object) with a vCenter Server instance. Plug-ins can use this property to identify auxiliary servers that are part of the same plug-in instance. For more information, see Using Auxiliary Plug-in Servers.

Used by: app.PluginBackendInfo

app.PluginDateTimeFormatOptions

Description	Specifies which part of a timestamp to format.

Property	Type	Required?	Notes
format	string	no (default: `DATE_AND_TIME`)	Possible values: • `DATE` • `TIME` • `DATE_AND_TIME`

Used by: app.formatDateTime()

app.PluginTheme

Description	Indicates a choice of UI theme.

Property	Type	Required?	Notes
name	string	info only	Possible values: light or dark.

Used by: app.getTheme()

app.QueryParam

Description	Holds a single query parameter of a URL.

Property	Type	Required?	Notes
name	string	info only	Name of query parameter, as in ?name=value.
value	string	info only	Value of query parameter, as in ?name=value.

Used by: app.UiApiEndpoint

app.refreshPluginItemsState()

Signature	app.refreshPluginItemsState():void
Description	Triggers the refresh of all dynamic UI content hosted by the current plug-in. For example, the plug-in introduces a view that is dynamically filtered based on a property value of some object. If the property value is changed by the plug-in using the public vSphere APIs, then the plug-in must call this API to signal the UI to re-evaluate the state of the initiator's plug-in related items.

app.RemotePluginNavigationMap

Description	Lists remote plug-in destination views, belonging to a single vCenter Server instance, that are exposed by plug-ins using the isPublic property in the plug-in manifest. Use the map key as the value of the vcGuid property in the parameter to app.navigateToRemotePluginView() .

Property	Type	Required?	Notes
navigationInfoByVcGuid	Map<string, app.RemotePluginVcNavigationInfo>	info only	A one-to-many mapping of vCenter Server instances to view IDs that are served by a specified plug-in instance registered with each vCenter Server instance. Each map entry contains the GUID of a vCenter Server instance within the same link group, and a corresponding list of view descriptors.

Used by: app.getRemotePluginNavigationMap()

app.RemotePluginVcNavigationInfo

Description	Descriptor for views served by a given plug-in. Use a view ID as the value of the `targetViewId` property in the parameter to app.navigateToRemotePluginView()

Property	Type	Required?	Notes
`pluginVersion`	`string`	info only	The version of a plug-in instance registered with a given vCenter Server instance.
`viewIds`	`string[]`	info only	All the view IDs, served by a given plug-in instance, that are accessible to the current plug-in instance.

Used by: app.RemotePluginNavigationMap

app.RemotePluginViewNavigationOptions

Description	Specifies a destination view served by a different plug-in instance, and custom data for the view.

Property	Type	Required?	Notes
`pluginId`	`string`	yes	ID of the plug-in that owns the destination view.
`targetViewId`	`string`	yes	Navigation ID of the destination view. Use only values returned by `htmlClientSdk.app.getRemotePluginNavigationMap()` in the `viewsIds` array.
`vcGuid`	`string`	yes	The GUID of the vCenter Server instance that determines the context for the target view. If `objectId` is specified, the object must be managed by the specified vCenter Server instance.
`objectId`	`string`	no	ID of any object associated with the view. (For a global view, this field is not required.)
`customData`	`any`	no	A custom data structure passed to the view.

Used by: app.navigateToRemotePluginView()

app.SessionInfo

Description	Holds information about the current session of the vSphere Client.

Property	Type	Required?	Notes
`sessionToken`	`string`	info only	Identifier of the plug-in authentication session with vCenter Server.
`nodeId`	`string`	info only	Reserved for internal use only.

Used by: app.getSessionInfo()

app.UiApiEndpoint

Description	Holds the parsed elements of a plug-in URL.

Property	Type	Required?	Notes
origin	string	info only	<protocol>://<hostname><port>
pathname	string	info only	
queryParams	Array< app.QueryParam >	info only	<name>=<value>
fullUrl	string	info only	<origin>/<pathname>?<queryParams>

Used by: app.ApiEndpoints

vSphere Client JavaScript API: Event Interface

The event interface helps your plug-in with event management.

event.onDateTimeFormatChanged()

Signature	event.onDateTimeFormatChanged(callback:function):void
Description	Registers an event handler for changes to the datetime format preferences in the vSphere Client.
Parameter: callback	A reference to a function that responds to changes in datetime preferences. The callback function must have the following signature: function callback():void

event.onGlobalRefresh()

Signature	event.onGlobalRefresh(callback:function):void
Description	Registers a global refresh handler that the vSphere Client will call when the **Global Refresh** button is clicked.
Parameter: callback	A reference to a global refresh handler. The callback function must have the following signature: function callback():void

event.onThemeChanged()

Signature	event.onThemeChanged(callback:function):void
Description	Registers an event handler that the vSphere Client will call when the vSphere Client changes the current theme.
Parameter: callback	A reference to a theme change handler. The callback function must have the following signature:

Table continued on next page

Continued from previous page

	`function callback(theme:app.PluginTheme):void` The `theme` parameter identifies the new vSphere Client theme.

Example Using the `modal` API

This example shows some basic features of the `modal` interface of the Client API.

modal.html

```
<html>
  <head>
    <script src="http://code.jquery.com/jquery-latest.min.js"
            type="text/javascript"></script>
    <script src="/api/ui/htmlClientSdk.js"
            type="text/javascript"></script>
    <script type='text/javascript'>
      function handler(event)
      {
        var choice = $('input[name=heads_or_tails]:checked').val();
        htmlClientSdk.modal.setOptions({title: choice});
        settimeout(function(){htmlClientSdk.modal.close(choice);}, 3000);
      }
    </script>
  </head>
  <body>
    <form name='flip' onSubmit='return handler()'>
      <p><input type='radio' name='heads_or_tails' value='HEADS' />HEADS</p>
      <p><input type='radio' name='heads_or_tails' value='TAILS' />TAILS</p>
      <input type='submit' name='submit' value='Submit' />
    </form>
  </body>
</html>
```

modal.js

```
flipper = function(){
  # Select correct answer.
```

```
  correct = ['heads', 'tails'][2*Math.random()-1];

  # Create callback function.
  checker = function(choice){
    var correct = htmlClientSdk.modal.getCustomData();
    if (choice === correct) {
      alert('You chose wisely.');
    } else {
      alert('Sorry, you lose.');
    }}

  # Configure modal dialog.
  var config ={
    url: "example/dialog.html",
    title: 'Choose!',
    size: { width: 490, height: 240 },
    onClosed: checker,
    customData: correct}

  # Open modal dialog.
  htmlClientSdk.modal.open(config);
}

# Initialize Javascript API.
$(document).ready(htmlClientSdk.initialize(flipper));
```

Using Themes with vSphere Client Plug-ins

The vSphere Client SDK provides the means for a plug-in to integrate with the themes supported by the vSphere Client. Modifying a plug-in to support themes requires changes to the plug-in style sheets and front-end code to switch style sheets whenever the user changes the theme in the vSphere Client.

To integrate with the vSphere Client themes, a plug-in uses these methods of the JavaScript API:

- `app.getTheme()`
- `event.onThemeChanged(callback)`

To prepare your code for a theme change, you must identify and isolate theme-dependent styles and icons, then create variables with which to manage the style changes and overrides.You can merge your styles with the standard Clarity styles to improve performance. Finally, you can use examples in this book to load new styles in response to user theme changes.

The following procedures assume that the plug-in's front-end code is built using Angular and Clarity Design System. For other frameworks and build tools, the approach is similar but you will need to adapt the approach to suit the chosen tools. The examples in this guide are based on the HTML Plug-in Sample provided as part of the vSphere Client SDK.

Using Style Variables in Plug-In CSS

If a plug-in uses custom styles that depend on the theme colors, the plug-in style sheets (CSS or SASS or LESS) need to be parameterized. This enables the plug-in to adapt when the user switches themes in the vSphere Client user interface.

Ensure that the plug-in's Clarity version supports the `dark` theme. The first Clarity version to support the `dark` theme is 0.10.16.

In this procedure you copy any custom colors that depend on the current theme into variables in separate style sheets that are specific to the `light` or `dark` theme. You replace the colors in the original style sheets with instances of CSS variables. This is done to avoid style sheet duplication and to easily integrate theming with any custom Angular components the plug-in has defined. For more information about CSS variables, see https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_variables.

1. Identify any theme-dependent colors or styles in your plug-in.
2. Factor out theme-dependent colors or styles into two new style sheets as CSS variables.

   The SDK includes the following sample file at `html-client-sdk/samples/remote-plugin-sample/src/main/ui/src/styles-light.css`.

   ```
   :root {

           --border-color: rgb(204, 204, 204);

           --overlay-color: rgba(255, 255, 255, 0.2);

           --info-icon-color: darkblue;

         }
   ```

   The SDK includes the following sample file at `html-client-sdk/samples/remote-plugin-sample/src/main/ui/src/styles-dark.css`.

   ```
   :root {

           --border-color: rgb(72, 87, 100);

           --overlay-color: rgba(0, 0, 0, 0.2);

           --info-icon-color: darkblue;

         }
   ```

3. Replace the theme-dependent colors or styles in the original style sheets with variable references.

   The SDK includes the following code in the sample file at `html-client-sdk/samples/remote-plugin-sample/src/main/ui/src/app/views/list/list.component.scss`.

   ```
   .splitter {

               flex: 0 0 auto;
   ```

```
    width: 1px;
    margin: 0 20px;
    background-color: var(--border-color);
  }
```

4. For Internet Explorer 11, which does not include support for CSS variables, include a polyfill library to provide support for CSS variables.

   The vSphere Client SDK includes a remote plug-in sample that uses `css-vars-ponyfill`. The following example is borrowed from `html-client-sdk/samples/remote-plugin-sample/src/main/ui/src/index.html`.

   ```
   <script type="text/javascript" src="scripts/css-vars-ponyfill.js"></script>
   ```

- Configure theme-dependent icons in the plug-in manifest file.
- Use the modified input style sheets to build the output style sheets for your plug-in.

Building Output Style Sheets for vSphere Client Plug-Ins

After you isolate theme-dependent colors or styles as CSS variables, you can merge the resulting style sheets with the standard Clarity styles to produce a set of output style sheets for optimized performance.

Refactor the input style sheets for the plug-in so that they isolate theme-dependent colors and styles in separate style sheets as CSS variables.

Angular applications which use `webpack` and `angular-cli` place the style sheet declarations inline by default, when in development mode. Inline style declarations interfere with dynamic CSS loading. When you build the output style sheets, always configure the build to output and use external CSS:
To build external style sheets, add the `--extract-css` parameter to the `ng build` command. The vSphere Client SDK has examples of this usage in `html-client-sdk/samples/remote-plugin-sample/src/main/ui/package.json`.
You must deactivate any output file name hashing in the development and production builds. Otherwise the names of the style sheet files will change whenever the code changes, and the plug-in will not be able to load them.
To deactivate file name hashing when you build style sheets, use this syntax: `ng build --prod --output-hashing none`.

1. Create a base output style sheet that is independent of the themes.

   The base style sheet contains the Clarity icons style sheet and the base input style sheet for the plug-in, which uses CSS variables. The vSphere Client SDK builds this output style sheet by using Angular to compile the SCSS.

   The following example comes from the vSphere Client SDK file `html-client-sdk/samples/remote-plugin-sample/src/main/ui/angular-cli.json`.

   ```
   "styles": [
     {
       "input": "../node_modules/clarity-icons/clarity-icons.min.css",
       "output": "styles",
       "lazy": true
     },
   ```

```
    {
      "input": "styles.css",
      "output": "styles",
      "lazy": true
    }
    ...
]
```

This step combines the contents of `html-client-sdk/samples/remote-plugin-sample/src/main/ui/node_modules/remote-plugin-sample/src/main/ui/styles.css` and `html-client-sdk/samples/remote-plugin-sample/src/main/ui/styles.css` into `html-client-sdk/samples/remote-plugin-sample/target/classes/ui/styles.bundle.css`.

2. Create an output style sheet file for the `light` theme.

   This style sheet includes the Clarity style sheet for the `light` theme and the plug-in style sheet for the `light` theme, which contains the CSS variable definitions for the `light` theme.

   The following example comes from the vSphere Client SDK file `html-client-sdk/samples/remote-plugin-sample/src/main/ui/angular-cli.json`.

```
"styles": [
    ...
    {
      "input": "../node_modules/clarity-ui/clarity-ui.min.css",
      "output": "theme-light",
      "lazy": true
    },
    {
      "input": "styles-light.css",
      "output": "theme-light",
      "lazy": true
    }
    ...
]
```

   This step combines the contents of `html-client-sdk/samples/remote-plugin-sample/src/main/ui/node_modules/clarity-ui/clarity-ui.min.css` and `html-client-sdk/samples/remote-plugin-sample/src/main/ui/styles-light.css` into `html-client-sdk/samples/remote-plugin-sample/target/classes/ui/theme-light.bundle.css`.

3. Create an output style sheet file for the `dark` theme.

This style sheet includes the Clarity style sheet for the `dark` theme and the plug-in style sheet for the `dark` theme, which contains the CSS variable definitions for the `dark` theme.

The following example comes from the vSphere Client SDK file `html-client-sdk/samples/remote-plugin-sample/src/main/ui/.angular-cli.json`.

```
"styles": [
  ...
  {
    "input": "../node_modules/clarity-ui/clarity-ui-dark.min.css",
    "output": "theme-dark",
    "lazy": true
  },
  {
    "input": "styles-dark.css",
    "output": "theme-dark",
    "lazy": true
  }
  ...
]
```

This step combines the contents of `html-client-sdk/samples/remote-plugin-sample/src/main/ui/node_modules/clarity-ui/clarity-ui-dark.min.css` and `html-client-sdk/samples/remote-plugin-sample/src/main/ui/src/styles-dark.css` into `html-client-sdk/samples/remote-plugin-sample/target/classes/ui/theme-dark.bundle.css`.

Write front-end code to load style sheets that match the theme selected by the user.

Configuring and Loading Theme Style Sheets in vSphere Client Remote Plug-Ins

After you compile the output style sheets for your plug-in user interface, you write front-end code to load the style sheets that cause your plug-in to conform to the style selected in the vSphere Client.

- Refactor the input style sheets for the plug-in so that they isolate theme-dependent colors and styles in separate style sheets as CSS variables.
- Configure theme-dependent icons in the plug-in manifest file.
- Build output style sheets into a base style sheet and a style sheet for each theme.

1. Load and configure polyfill libraries to provide CSS variable support in Internet Explorer 11.

   If you use `css-vars-ponyfill`, consider whether to configure options to create a MutationObserver and whether to remove CSS rulesets and declarations that do not reference a CSS custom property value. For more information

about configuring `css-vars-ponyfill`, see https://github.com/jhildenbiddle/css-vars-ponyfill/tree/v1.17.1#optionswatch and https://github.com/jhildenbiddle/css-vars-ponyfill/tree/v1.17.1#optionsonlyvars.

The vSphere Client SDK includes a remote plug-in sample that uses `css-vars-ponyfill`. The following example is borrowed from the file `html-client-sdk/samples/remote-plugin-sample/src/main/ui/src/index.html`.

```
<script type="text/javascript" src="scripts/css-vars-ponyfill.js"></script>


// Initialize CSS vars to configure polyfill.

cssVars({

          watch: true,

          onlyVars: true

});
```

The following example is borrowed from the file `html-client-sdk/samples/remote-plugin-sample/src/main/ui/.angular-cli.json`.

```
  "assets": [

    "assets",

    {

      "glob":

      "css-vars-ponyfill.js",

      "input": "../node_modules/css-vars-ponyfill/dist/",

      "output": "scripts/

    },

   …

  ]
```

2. Load the base style sheet initially.

   The following example is borrowed from `html-client-sdk/samples/remote-plugin-sample//src/main/ui/src/index.html`.

   ```
   <link rel="stylesheet" type="text/css" href="styles.bundle.css">
   ```

3. Load and initialize the vSphere Client JavaScript API.

   ```
   <script type="text/javascript" src="/api/ui/htmlClientSdk.js"></script>

   <script type="text/javascript">

     htmlClientSdk.initialize(init_plugin_view());

   </script>
   ```

For examples in the SDK, see `html-client-sdk/samples/remote-plugin-sample//src/main/ui/src/index.html` and `html-client-sdk/samples/remote-plugin-sample//src/main/ui/src/app/app.component.ts`.

4. Load the style sheet for the current theme initially and whenever the style changes.

   The following example is adapted from `html-client-sdk/samples/remote-plugin-sample/src/main/ui/src/app/app.component.ts`.

```
if (this.globalService.htmlClientSdk.app.getTheme &&
  this.globalService.htmlClientSdk.event.onThemeChanged) {
    this.loadTheme(true, this.globalService.htmlClientSdk.app.getTheme());
    this.globalService.htmlClientSdk.event.onThemeChanged(
                         this.loadTheme.bind(this, false));
  } else {
    this.loadTheme(true, { name: 'light' });
  }


private loadTheme(firstLoad: boolean, theme: any): void {
  let themeName: string = theme.name;
  let supportedThemeNames: string[] = ['light', 'dark'];
  if (supportedThemeNames.indexOf(themeName) === -1) {
    themeName = supportedThemeNames[0];
  }
  let styleSheetLinkElement =
    (<HTMLLinkElement> document.getElementById('theme-stylesheet-link'));
  let themeCssUrl = `theme-${themeName}.bundle.css`;


  if (firstLoad) {
    let initialThemeLoadCompleteListener = (event: Event) => {
      this.initialThemeLoadComplete = true;
      styleSheetLinkElement.removeEventListener('load',
                      initialThemeLoadCompleteListener);
      styleSheetLinkElement.removeEventListener('error',
                       initialThemeLoadCompleteListener);
    };
```

```
      styleSheetLinkElement.addEventListener('load',
                  initialThemeLoadCompleteListener);
      styleSheetLinkElement.addEventListener('error',
                   initialThemeLoadCompleteListener);
    }

    styleSheetLinkElement.setAttribute("href", themeCssUrl);
    document.documentElement.setAttribute("data-theme", themeName);
  }
```

Configuring Theme-Dependent Icons for vSphere Client Remote Plug-ins

When you refactor style sheets for your plug-in to accommodate theme changes, you can also specify icons suited for alternative themes. You specify theme-dependent icons in the plug-in manifest file.

Configuring theme-dependent icons is an optional step that can improve the user experience after a theme change.

1. Edit the plug-in manifest file, `plugin.json`, and locate the `iconSpriteSheet` property at the second level of the JSON, within the `definitions` property.

```
...
"definitions": {
  "iconSpriteSheet": {
    "uri" : "assets/images/sprites.png",
    "definitions": {
      "main" : {
        "x": 0,
        "y": 96
      }
    }
  },
  "i18n": {
    "locales": ["en-US"],
    "definitions": {
      "plugin.name": {
        "en-US": "Theme Example"
      }
```

```
      }
    }
  },
  ...
```

2. Within the `iconSpriteSheet` property, add a **`themeOverrides`** property that maps each theme name to a URI and the coordinates of its theme-dependent icons.

   The structure of each theme property inside the `themeOverrides` property is identical to the structure of the `iconSpriteSheet` property, which contains a `uri` and a `definitions` property.

```
  ...
  "definitions": {
    "iconSpriteSheet": {
      "uri" : "assets/images/sprites.png",
      "definitions": {
        "main" : {
          "x": 0,
          "y": 96
        }
      },
    "themeOverrides": {
      "dark": {
        "uri": "assets/images/sprites_dark.png",
        "definitions": {
          "main": {
            "x": 0,
            "y": 96
          }
        }
      }
    },
    "i18n": {
      "locales": ["en-US"],
      "definitions": {
        "plugin.name": {
```

```
          "en-US": "Theme Example"
        }
      }
    }
  },
  ...
```

A best practice is to maintain a separate style sheet for each theme, with corresponding icons in the same positions. When you do this, you do not need to override the default coordinates, and you can omit the `definitions` from the `themeOverrides` element.

3. Save your changes and close the manifest file.

Write front-end code to load style sheets that match the theme selected by the user.

Integrated Solution Installer for the vSphere Client

When you create a plug-in server encapsulated in an Open Virtual Format package (OVF), you can configure the OVF to automate the registration of the plug-in. This avoids the need to create a custom registration tool or to use the manual registration script in the SDK.

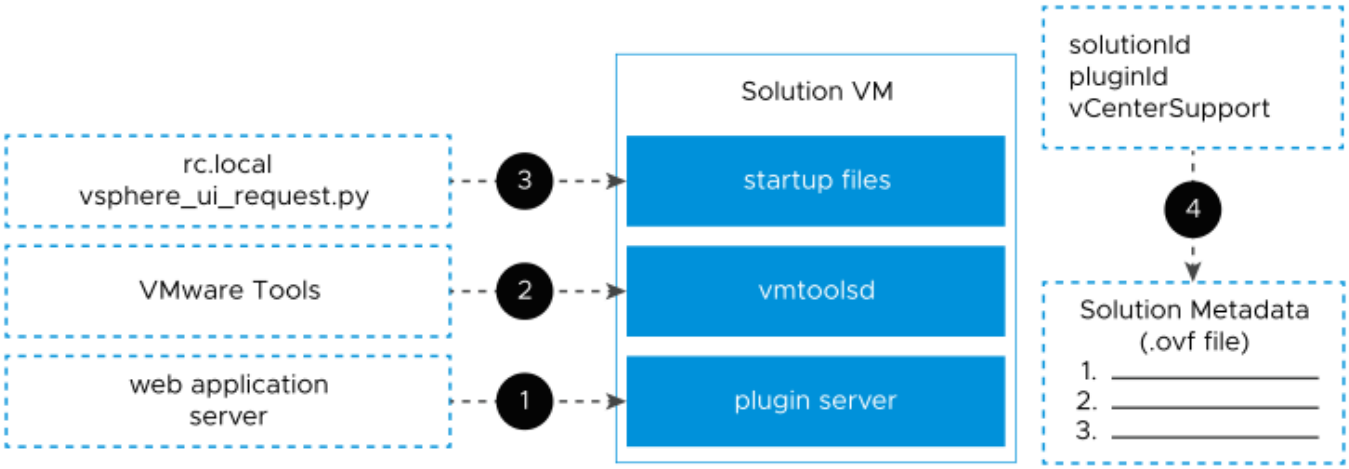
Conceptual Overview of the Preparation Process

A self-registering plug-in solution requires the coordination of several software functions. The following procedure shows how to prepare an OVF package for self-registration by installing needed software and configuring metadata that guides the registration process when the virtual machine first starts up after deployment to a vCenter Server instance.

In brief, these are the needed software and metadata components:

- The virtual machine needs a web application server. The following instructions show how to install the Remote Sample Starter from the vSphere Client SDK, which you can customize to support your application. Alternatively, you could install `nginx` or another server that is capable of serving the plug-in manifest during the registration process.
- The virtual machine needs VMware Tools installed. VMware Tools provides the mechanism to inject environmental parameters into the virtual machine. The following instructions begin with downloading a Photon OS virtual machine that has VMware Tools already installed. If you choose a different virtual machine as a starting point, you might need to install VMware Tools.
- The virtual machine also needs files that run during startup to assemble the environmental parameters and send a registration request to vCenter Server. Sample registration files intended for a Linux guest OS are provided in the vSphere Client SDK. A bash script queries VMware Tools for the parameters needed to build the registration request and launch a Python program to do the registration.
- The registration request includes solution identifying details. During deployment of the OVF files, the vSphere Client passes these details to the guest OS by means of VMware Tools..

Figure 147: Preparing the Solution OVF for the Integrated Installer



Preparing a Solution OVF to Use the Integrated Solution Installer

You can prepare a virtual machine with software and configuration metadata that work in conjunction with the vSphere Client to register a plug-in solution when the virtual machine starts up in a production environment.

Prerequisites

- vCenter Server 6.7 or later.
- An OVA package that contains a virtual machine with virtual hardware version 11 or later. PhotonOS, as used in this procedure, is recommended.
- `root` or `administrator` access to the guest operating system.
- (If you choose to use Docker to download `nginx`) access to the Docker Hub.

Outline of Steps to Prepare a Plug-in Server for Self-Registration and Installation

1. Acquire the OVA File for Photon OS, virtual hardware version 11
2. Use the vSphere Client to install the OVA in a vSphere environment
3. Configure vAPP options for deployment in a production environment
4. Configure Solution Properties in the OVF
5. Run the virtual machine and set the `root` password in the guest OS
6. Install your web application server in the guest OS
7. Installing VMware Tools if needed
8. Copy startup files to the `/etc/rc.d/` directory
9. Export the virtual machine as a new OVF template
10. Deploy the OVF template in a production environment

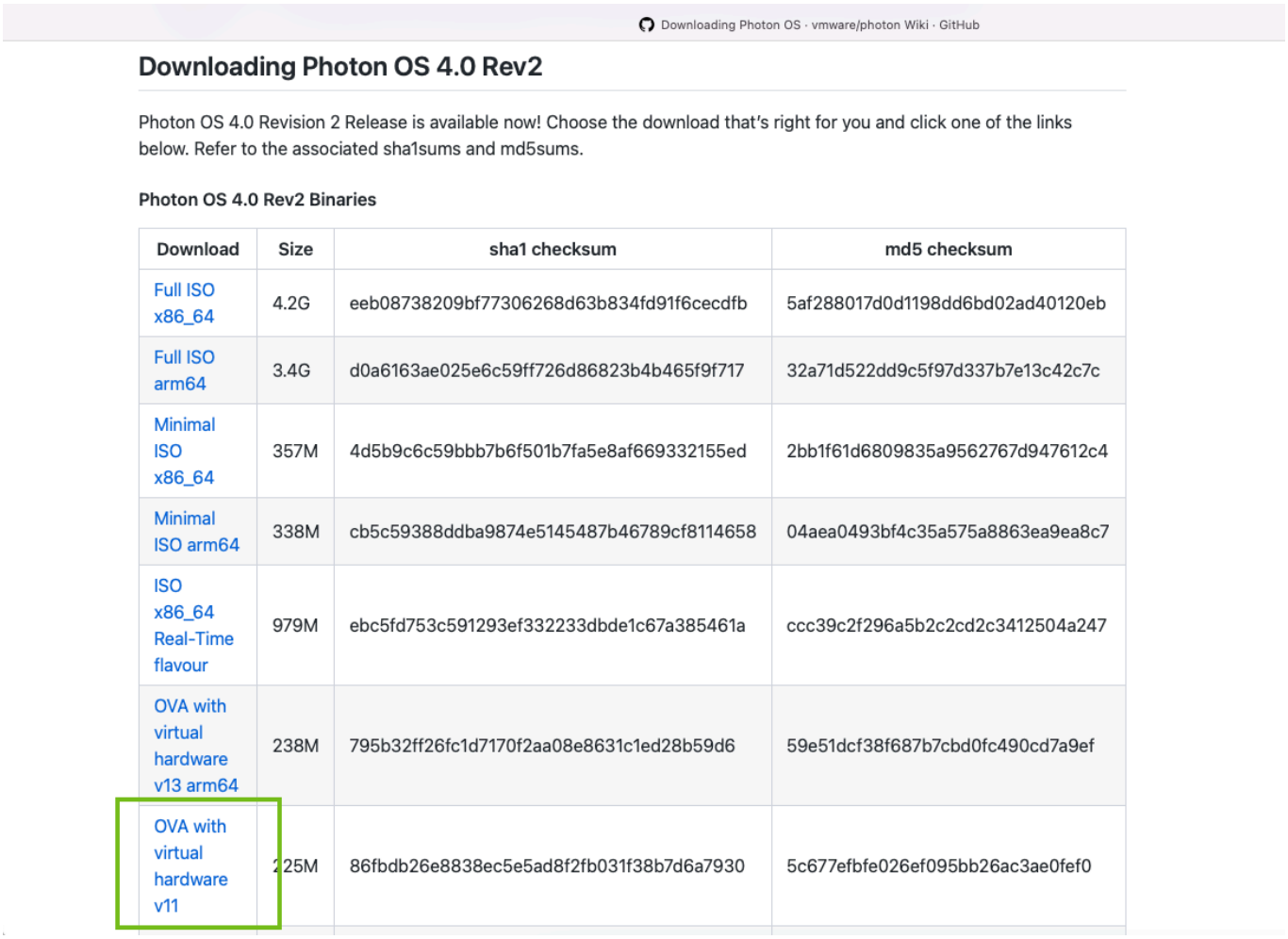
When a user deploys a virtual machine from the finished OVF template, the virtual machine starts the plug-in web server in the guest OS and registers it with vCenter Server.

Acquire the OVA File for Photon OS

Photon OS is an open-source, security-hardened, enterprise-grade Linux distribution designed for Cloud and Edge applications.

You can download a Photon OS OVA with virtual hardware v11 from https://github.com/vmware/photon/wiki/Downloading-Photon-OS

Figure 148: Download Photon OS from GitHub

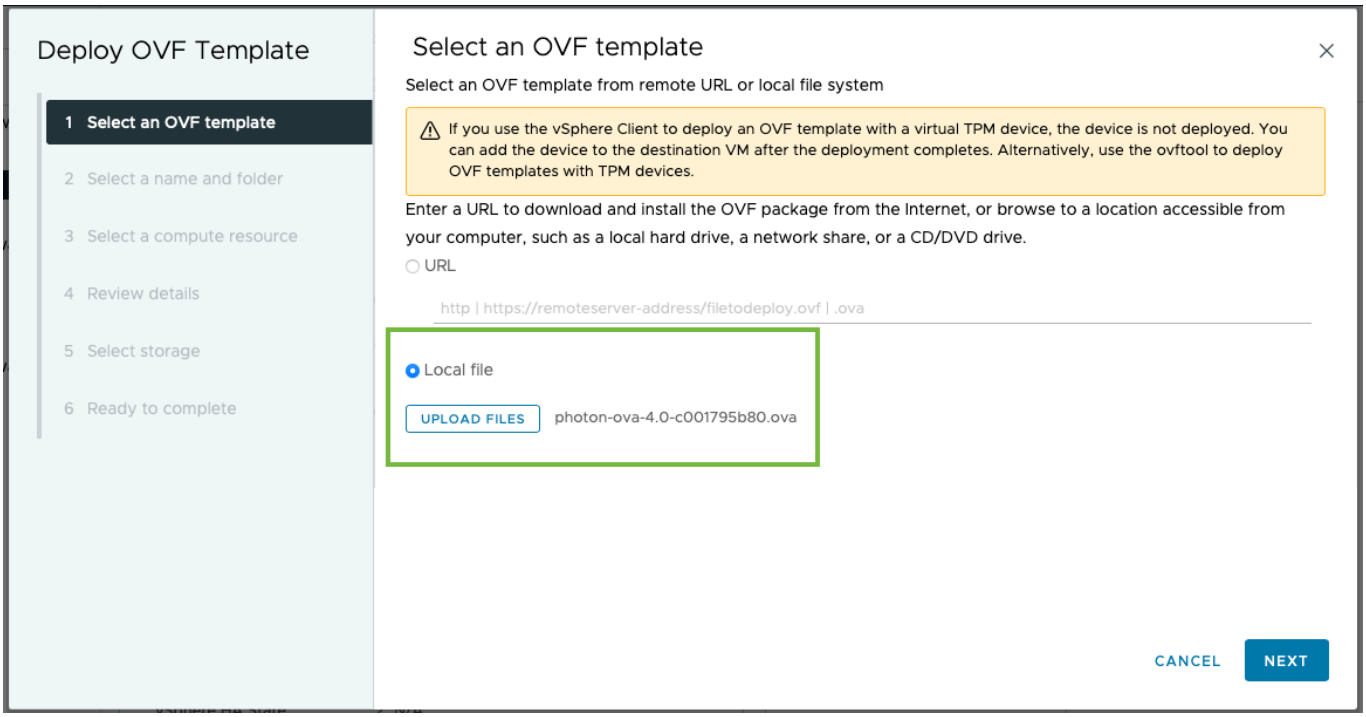


Install the OVA in a vSphere Environment

You need to install the virtual machine in an environment where you can modify its configuration and make some changes in the guest OS. To install the OVA, use the Deploy OVF Template wizard in the vSphere Client.

1. Right-click any inventory object that is a valid parent object of a virtual machine, such as a data center, folder, cluster, resource pool, or host.
2. Select **Deploy OVF Template**.
3. In the first step of the wizard, **Select an OVF template**, choose **Local file**.
4. Click **UPLOAD FILES** and choose the OVA file you downloaded.
5. Click **NEXT** and complete the Deploy OVF Template wizard.

Figure 149: The Deploy OVF Template wizard

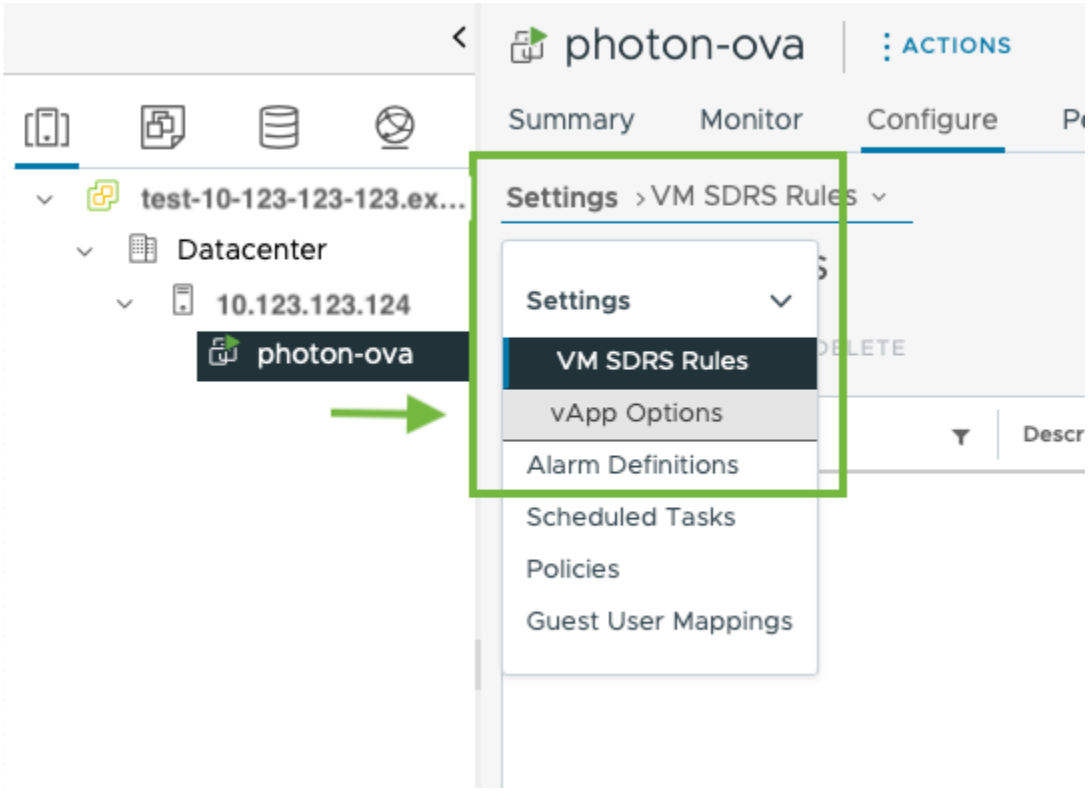


Configure vApp Options

After you install the virtual machine in your development environment, you configure certain vApp properties that will affect the virtual machine when it is deployed to a production environment.

1. Navigate to the deployed virtual machine name in the navigation pane on the left.
2. Click the **Configure** tab.
3. Select **Settings** > **vApp Options**.

Figure 150: Settings > VApp Options



4. Click the **EDIT** button near the top of the VApp Options panel.

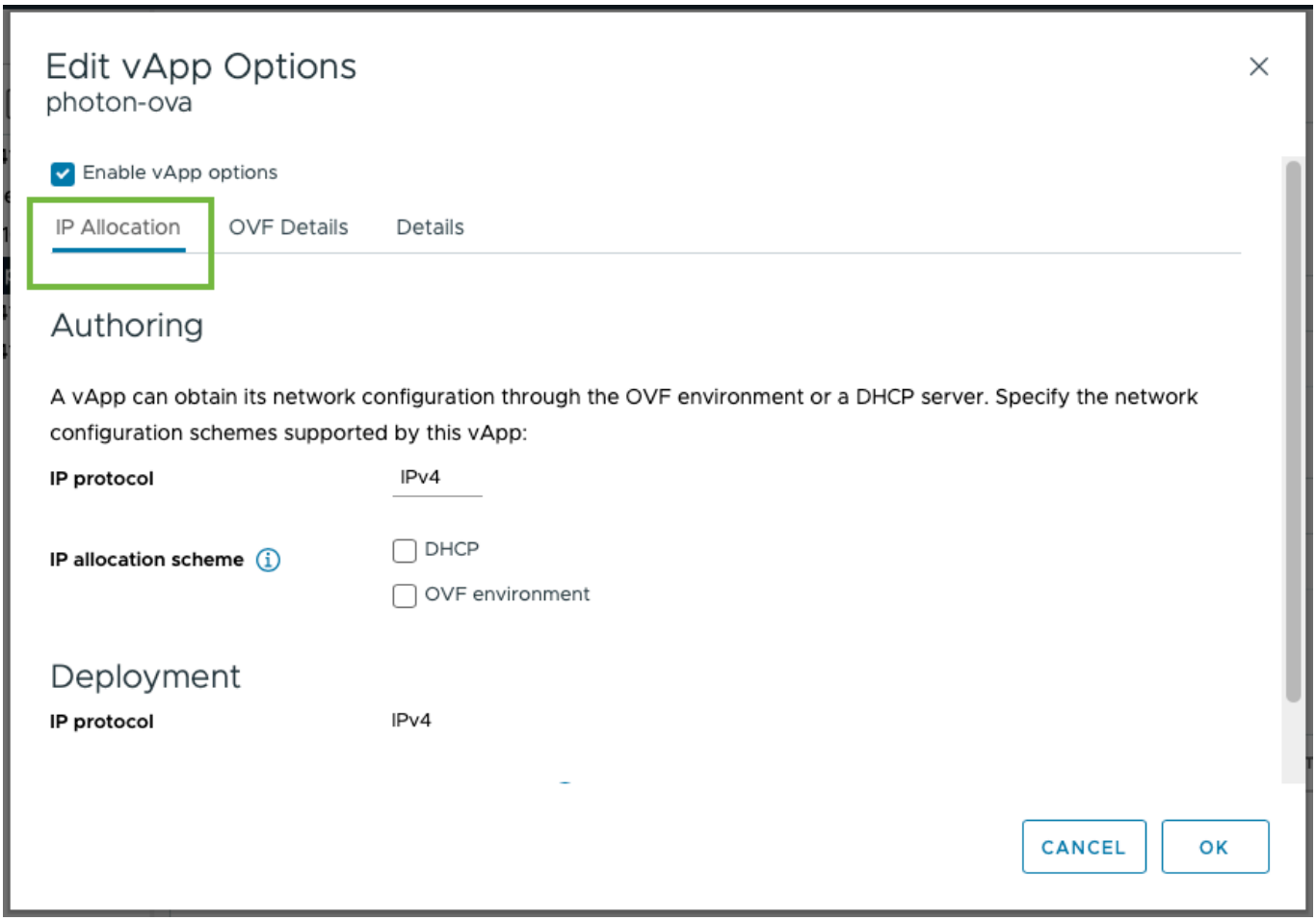
The Edit vApp Options panel contains three tabs: IP Allocation, vApp Details, and Details.

On the IP Allocation tab, select network constraints for the virtual machine.

- In the Authoring section, choose one or both of the network protocols (IPv4, IPv6, or both) for the vApp production environment. The choices you make in the Authoring section constrain the choices available in the Deployment section.

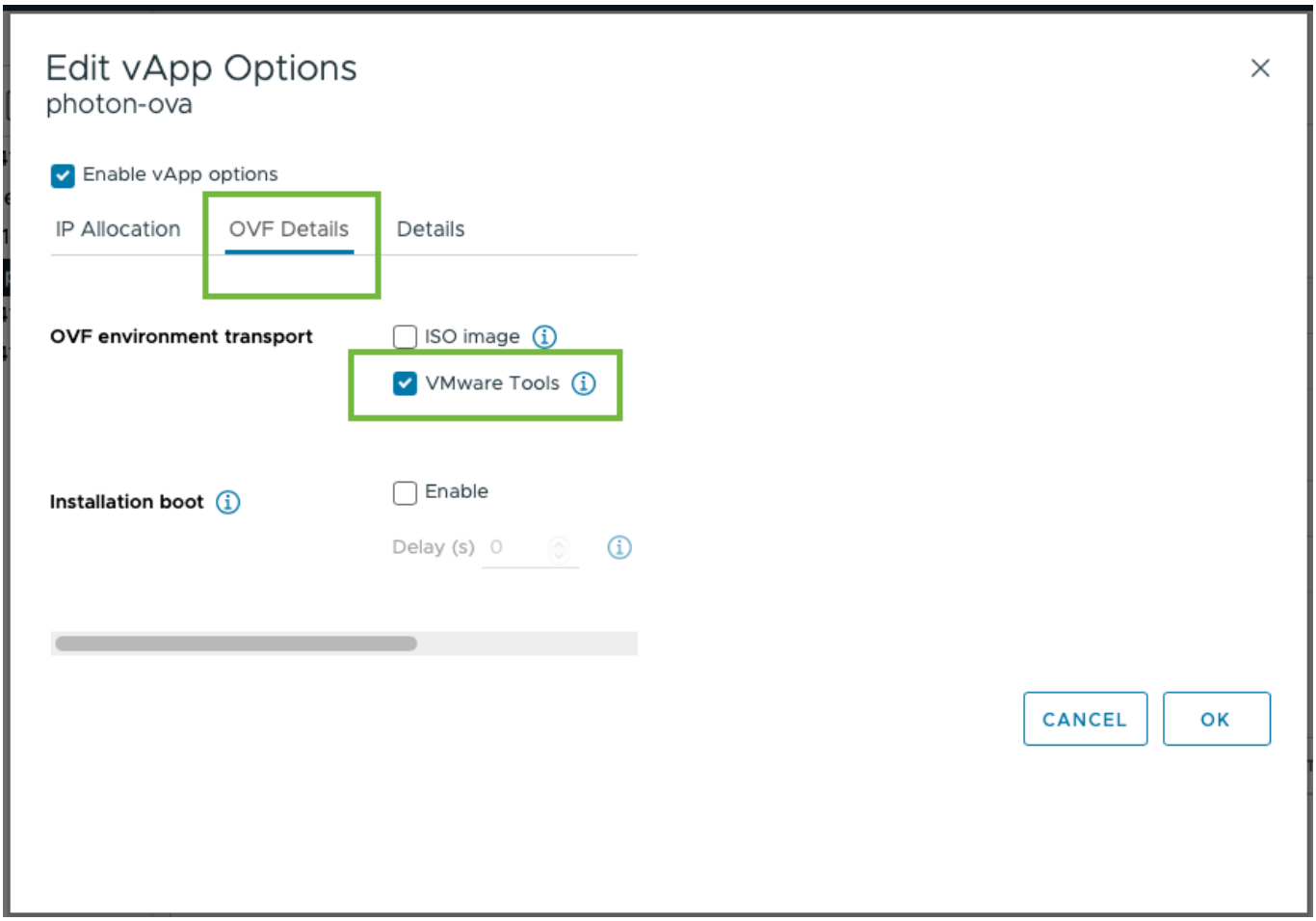
If you select OVF Environment in the Authoring section...	Choose IP pool settings in the Deployment section.
If you choose DHCP in the Authoring section...	DHCP service must be available on the vSphere networks where the vApp will be deployed.
If you make no selections in the Authoring section...	IP addresses must be assigned manually when the vApp is deployed.

Figure 151: Selecting IP Allocation Choices



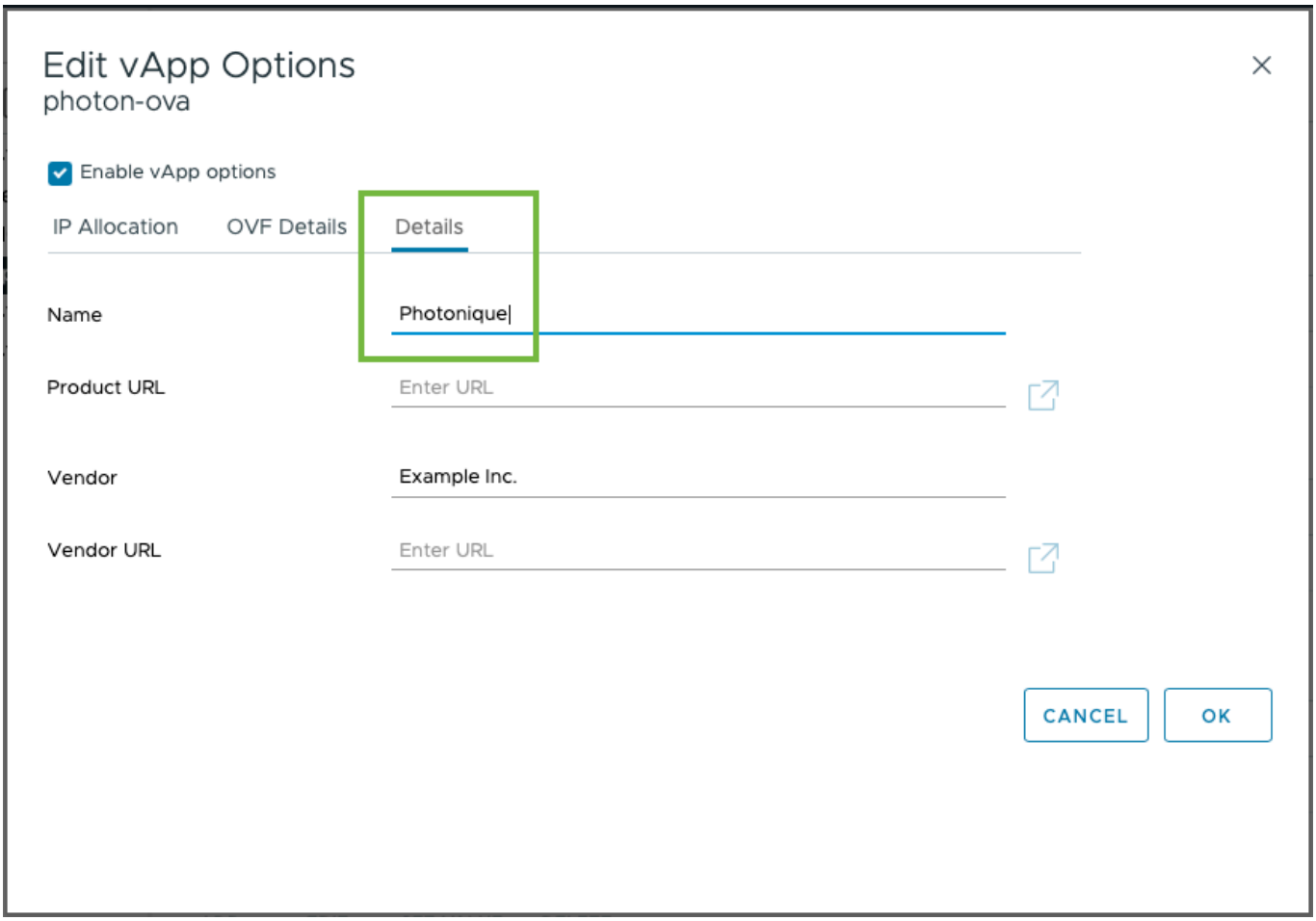
On the **OVF Details** tab of the Edit vApp Options window, in the **OVF environment transport** section, enable VMware Tools. When the virtual machine is deployed in a production environment, it will use VMware Tools to read solution properties that were configured in the OVF template. These properties will be injected into the plug-in registration request.

Figure 152: Selecting OVF Details



On the **Details** tab of the Edit vApp Options window, specify a vendor-supplied product name for the plug-in solution.

Figure 153: Entering Vendor Information

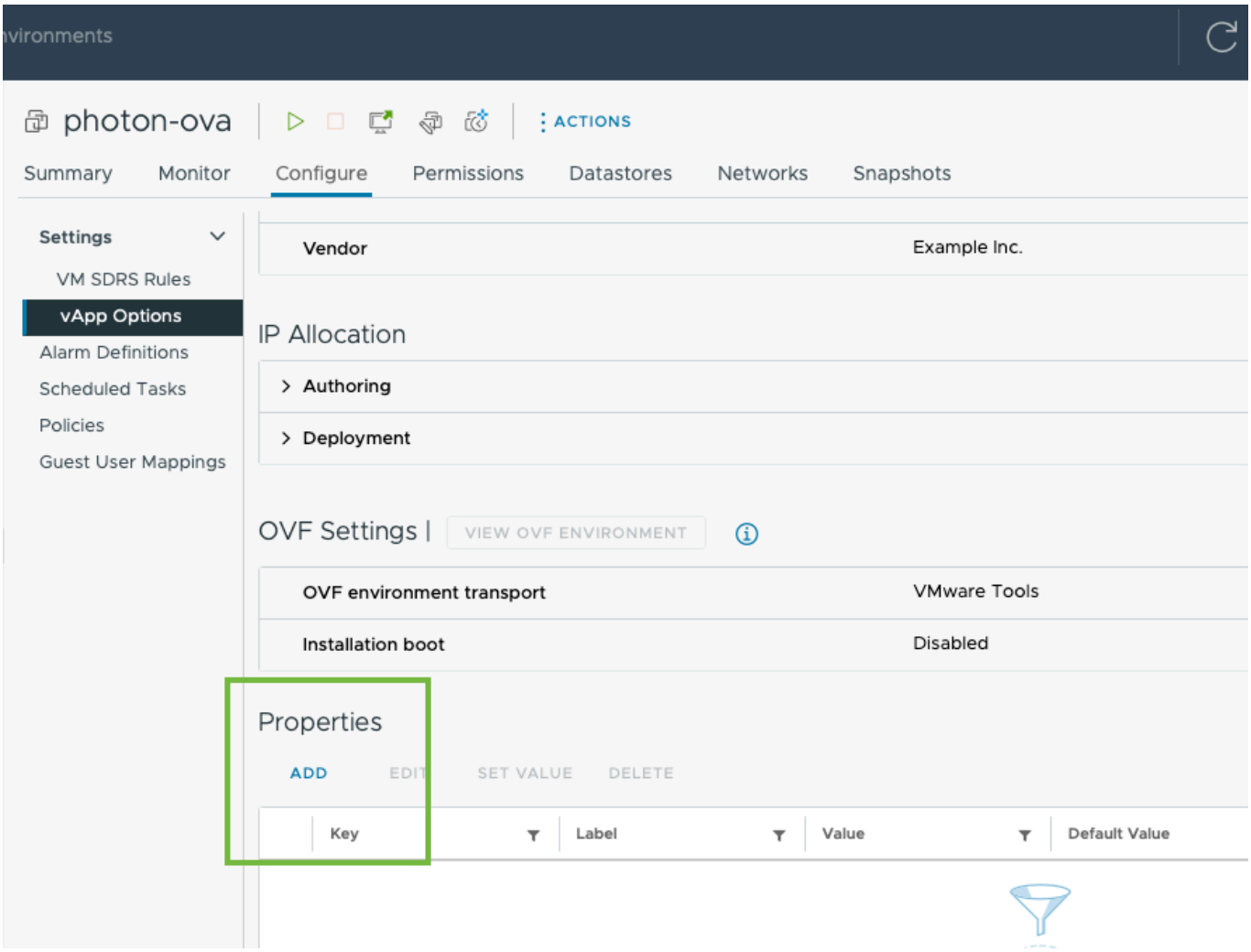


Configure Solution Properties in the OVF

When the virtual machine starts as a plug-in in a production environment, it needs to register itself with vCenter Server. Before you deploy the plug-in, you must configure the OVF template with several solution properties that will be injected into the registration record during startup. These include the solution ID, the plug-in ID, and other properties.

1. Navigate to the deployed virtual machine name in the navigation pane on the left
2. Power off the virtual machine if it is powered on.
3. Select the **Configure** tab in the selection pane on the left.
4. Select **Settings › vApp Options**.
5. In the vApp Options pane, scroll down to the Properties section.

Figure 154: Adding VApp Properties



Click **ADD** to add each new property. Configure the OVF environment with required properties, as shown in the following table. Default values are required, and must match the choice list value.

NOTE
Optional values and best practices are described in Integrated Installer Solution Metadata.

Property Key	Type	Example Choice List	Example Default Value
vmw.vsphereui.solutioninstall.solutionId	string choice	"com.example.solution.id"	com.example.solution.id
Choice list contains a single vendor-defined element. For example: "com.example.solution.id"			
vmw.vsphereui.solutioninstall.pluginId	string choice	"com.example.plugin.id"	com.example.plugin.id
Choice list contains a single vendor-defined element. For example: "com.example.plugin.id". Value may be same as solution ID.			
vmw.vsphereui.solutioninstall.vCenterSupport	string choice	"Single"	Single

Table continued on next page

Continued from previous page

Property Key	Type	Example Choice List	Example Default Value
Choice list contains either "Single" or "Multiple" to indicate whether the solution supports one or many vCenter Server instances.			

NOTE
Property keys are case sensitive.

NOTE
The OVF properties will be assigned values the next time the virtual machine is powered on. The values must be assigned before you export the OVF template.

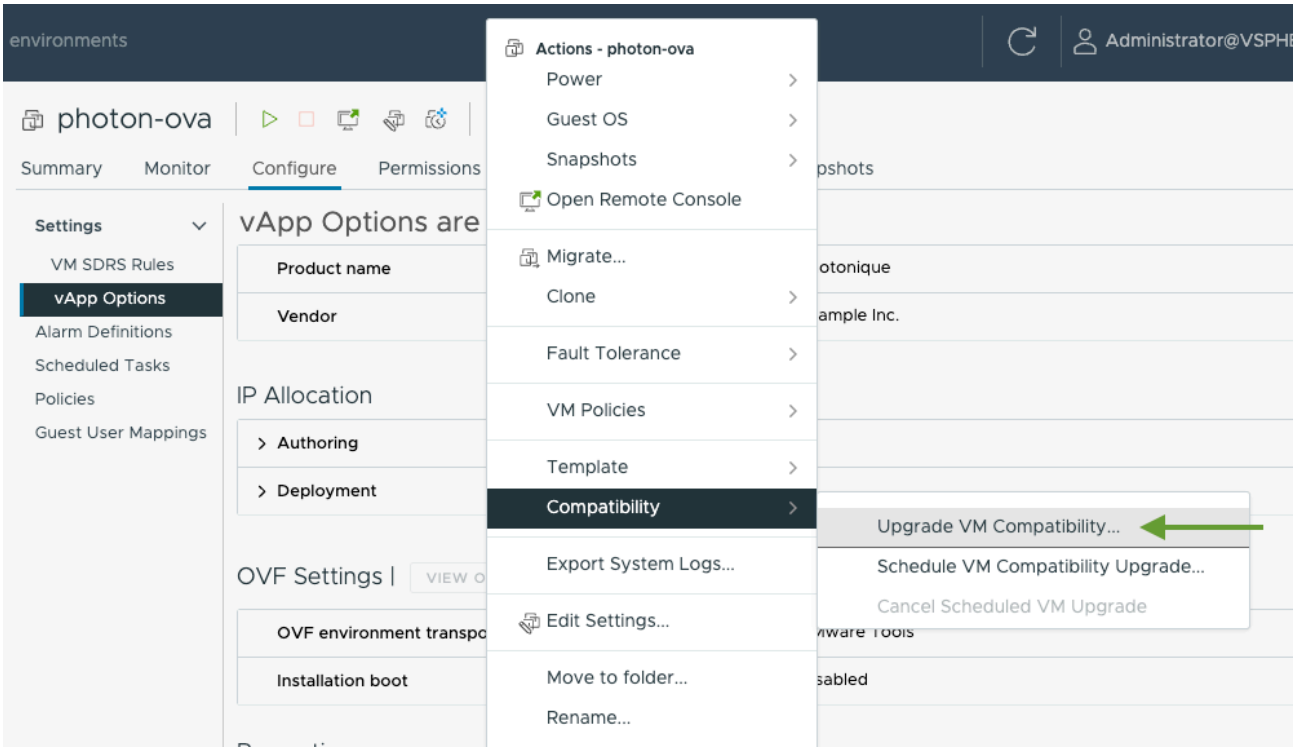
For more information about these properties and other optional properties, see Integrated Installer Solution Metadata.

Upgrade VM Compatibility

To be sure that the guest OS will be supported by the host when it is deployed, upgrade compatibility to **ESXi 6.7 Update 2 and later**.

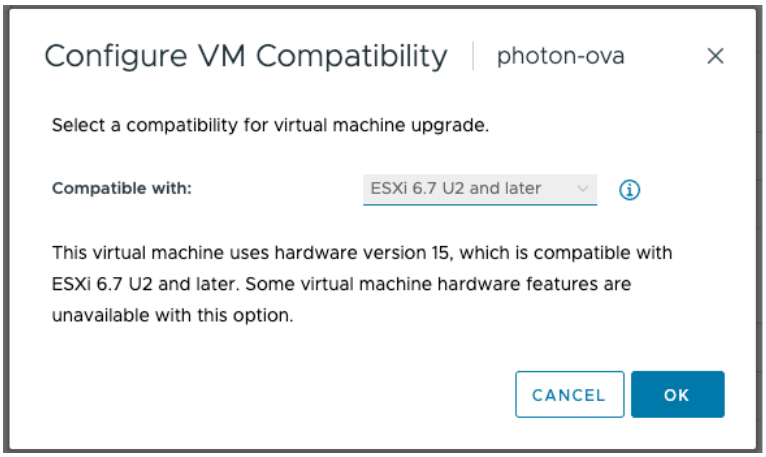
1. Power off the virtual machine, if running.
2. From the ACTIONS menu, choose **Compatibility › Upgrade VM Compatibility**.

Figure 155: Upgrading Compatibility for the Virtual Machine



3. In the Configure VM Compatibility dialog, choose **ESXi 6.7 U2 and later** from the **Compatible with** pop-up menu.

Figure 156: Upgrade VM Compatibility Dialog



Set the `root` Password in the Guest OS

When you download the OVA, the root password for the guest OS is insecure. You must change the password before you deploy the virtual machine in a production environment.

1. Power on the virtual machine.
2. Open a Web Console or Remote Console window to the guest OS.
   – The default root password for the Photon OS guest OS is `changeme`.
3. Set a new, secure, root password.
   – Make a note of the new password, in case you need to update the software or alter the configuration in the future.

Install Your Web Application Server in the Guest OS

The virtual machine needs a plug-in server to handle REST queries from the plug-in user interface. You can install the SDK sample starter or other web server, according to your preference.

To install the remote plug-in sample starter provided in the vSphere Client SDK:

1. Customize the sample starter as needed for your solution.
2. Build the remote-plugin-sample-starter from the SDK.
   See Build the Remote Plug-in Sample Starter.
3. Copy the remote-plugin-sample-starter-*VERSION*.jar into the guest OS. (for this example, we use the /root/ folder of the VM).

```
# cd /root

# scp myuserid@mydevbox.example.com:/html-client-sdk/samples/remote-plugin-sample-starter/target/*.jar .
```

Install VMware Tools

If you are using the Photon OS OVA downloaded from github as described in the first step of this procedure, VMware Tools is already installed in the guest. Otherwise, you might need to install VMware Tools manually. In the vSphere Client Hosts and Clusters view, right-click your virtual machine and select **Guest OS › Install VMware Tools**. For more detailed instructions, see the knowledge base article https://kb.vmware.com/s/article/2004754.

Copy Startup Files into the Guest OS

When the virtual machine starts up in a production environment, it needs to run startup scripts to register itself with vCenter Server. You must install the scripts in the Linux guest OS before you deploy the virtual machine in a production environment.

1. Copy the files `rc.local` and `vsphere_ui_request.py` from the SDK directory `/samples/solutioninstall/` to the directory `/etc/rc.d/` in the guest OS.
   ```
   # cd /etc/rc.d

   # scp myuserid@mydevbox.example.com:/html-client-sdk/samples/solutioninstall/
   rc.local .

   # scp myuserid@mydevbox.example.com:/html-client-sdk/samples/solutioninstall/
   vsphere_ui_request.py .
   ```
2. Set execute permissions on the files in the guest OS.
   ```
   # chmod +x /etc/rc.d/rc.local

   # chmod +x /etc/rc.d/vsphere_ui_request.py
   ```

When you deploy the virtual machine as a plug-in in vCenter Server, the guest OS runs `rc.local`, which in turn runs `vsphere_ui_request.py`. The request script uses VMware Tools to read the solution properties, then builds a registration request and sends it to vCenter Server.

If you install a web server other than the remote plug-in sample starter, you can customize `rc.local`, which is a bash script, or write your own script along similar lines. The script does the following actions:

- Reads solution metadata
- Installs a JDK
- Opens a firewall port for incoming HTTPS requests
- Starts the web server
- Launches `vsphere_ui_request.py` to send the plug-in registration request

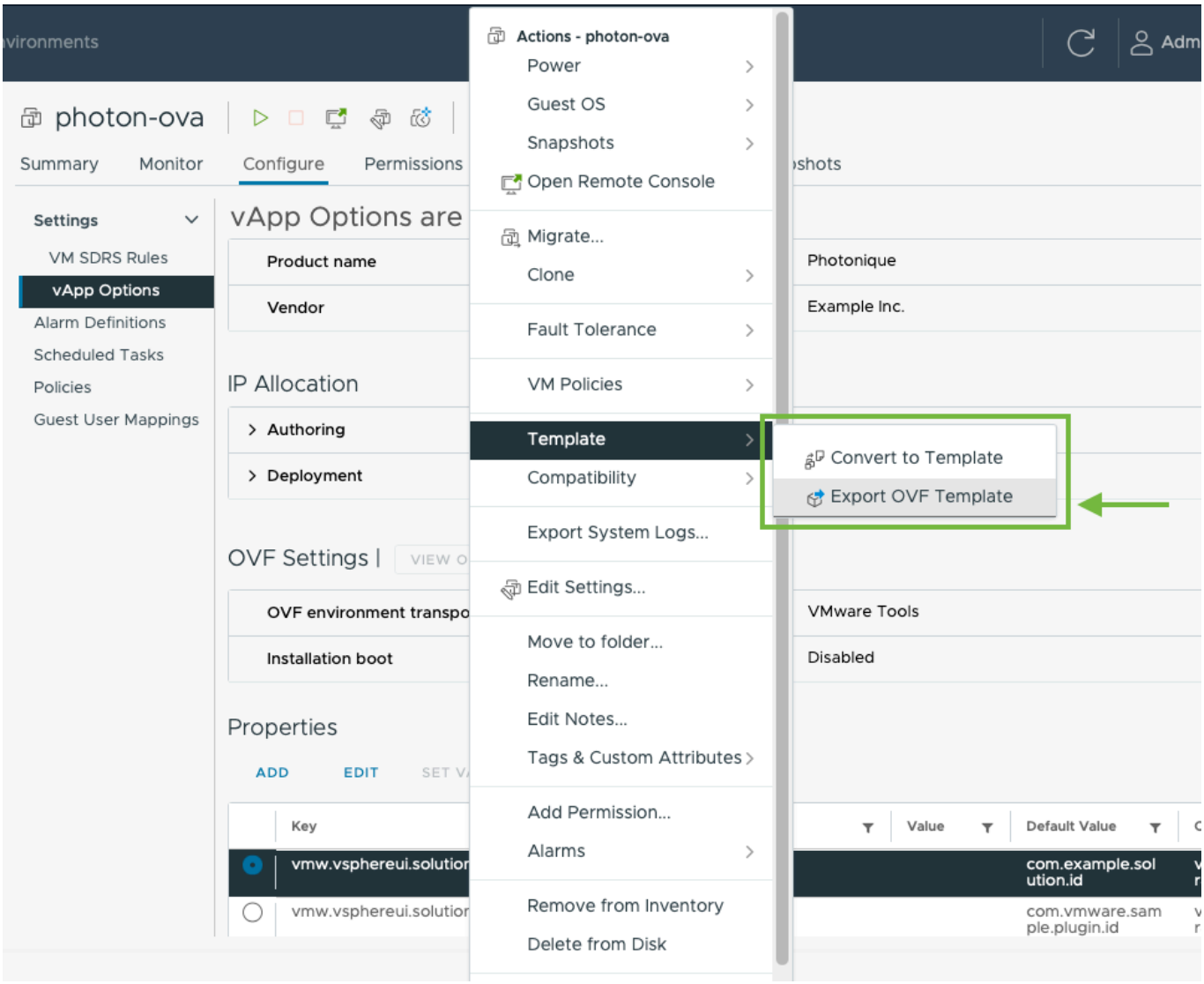
Note: If you customize the script, it is important that the vCenter Server thumbprint is verified as part of making the request to the vSphere UI.

Export a New OVF Template

After you configure VMware Tools, install the startup script, configure injectable solution properties, and install your plug-in web server, you can export the virtual machine as an OVF template that is available to deploy in your production environment.

1. Power off the virtual machine.
2. Export the virtual machine as an OVF, using the **Template › Export OVF Template** action.
   In the Export OVF Template dialog, supply a name and description for the template.

Figure 157: Exporting an OVF Template



Related Resources

- PhotonOS documentation: https://vmware.github.io/photon/docs/
- nginx downloads: https://hub.docker.com/_/nginx/

Deployment with the Integrated Solution Installer

After you complete the configuration steps and export the virtual machine as an OVF template, it is ready to install as a plug-in server in your production environment.

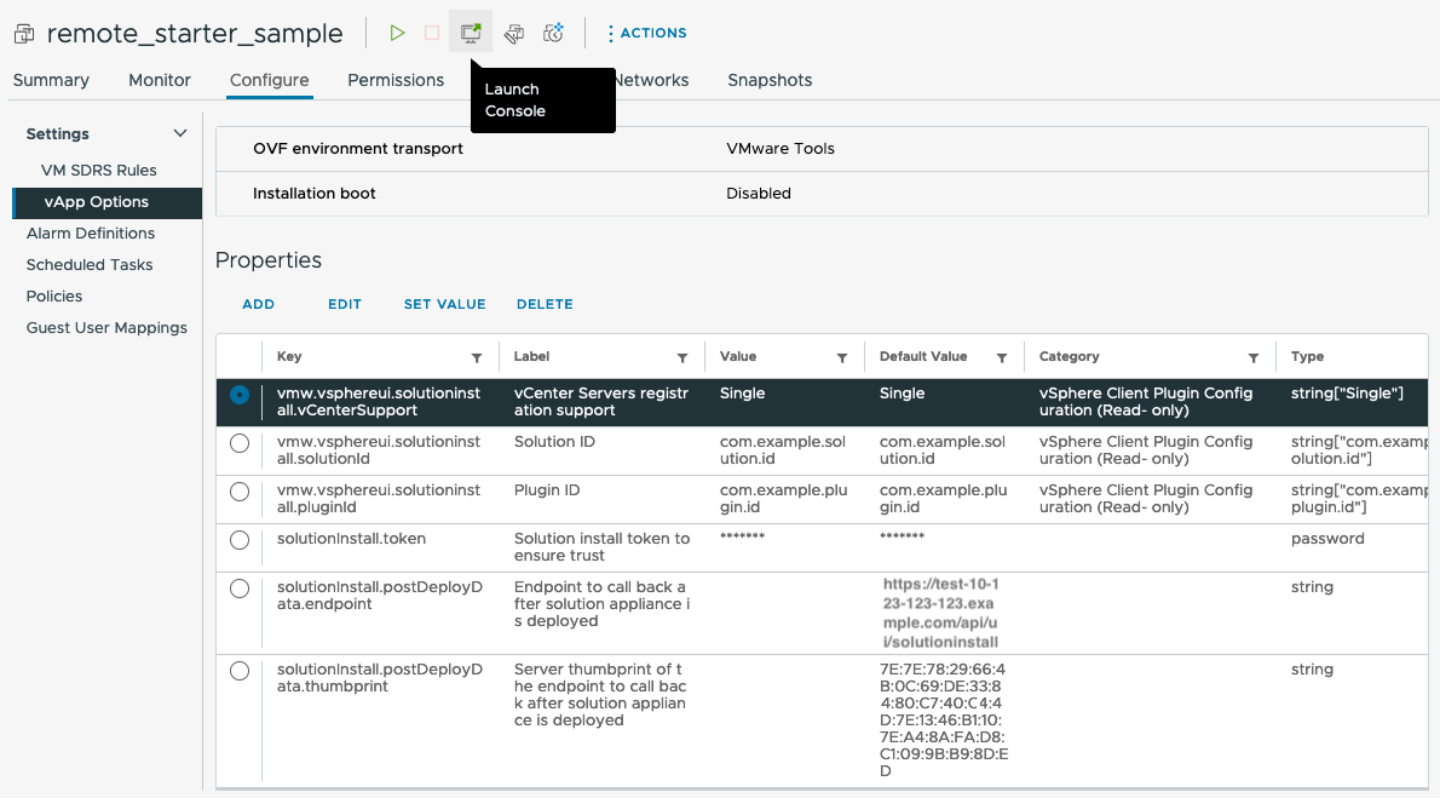
Deploy the OVF template in a production environment

1. If you prepared the virtual machine in a development environment, copy the OVF file to a location accessible to your production system.
2. Use the vSphere Client, connected to your production system, to add the OVF file to your plug-in deployment.

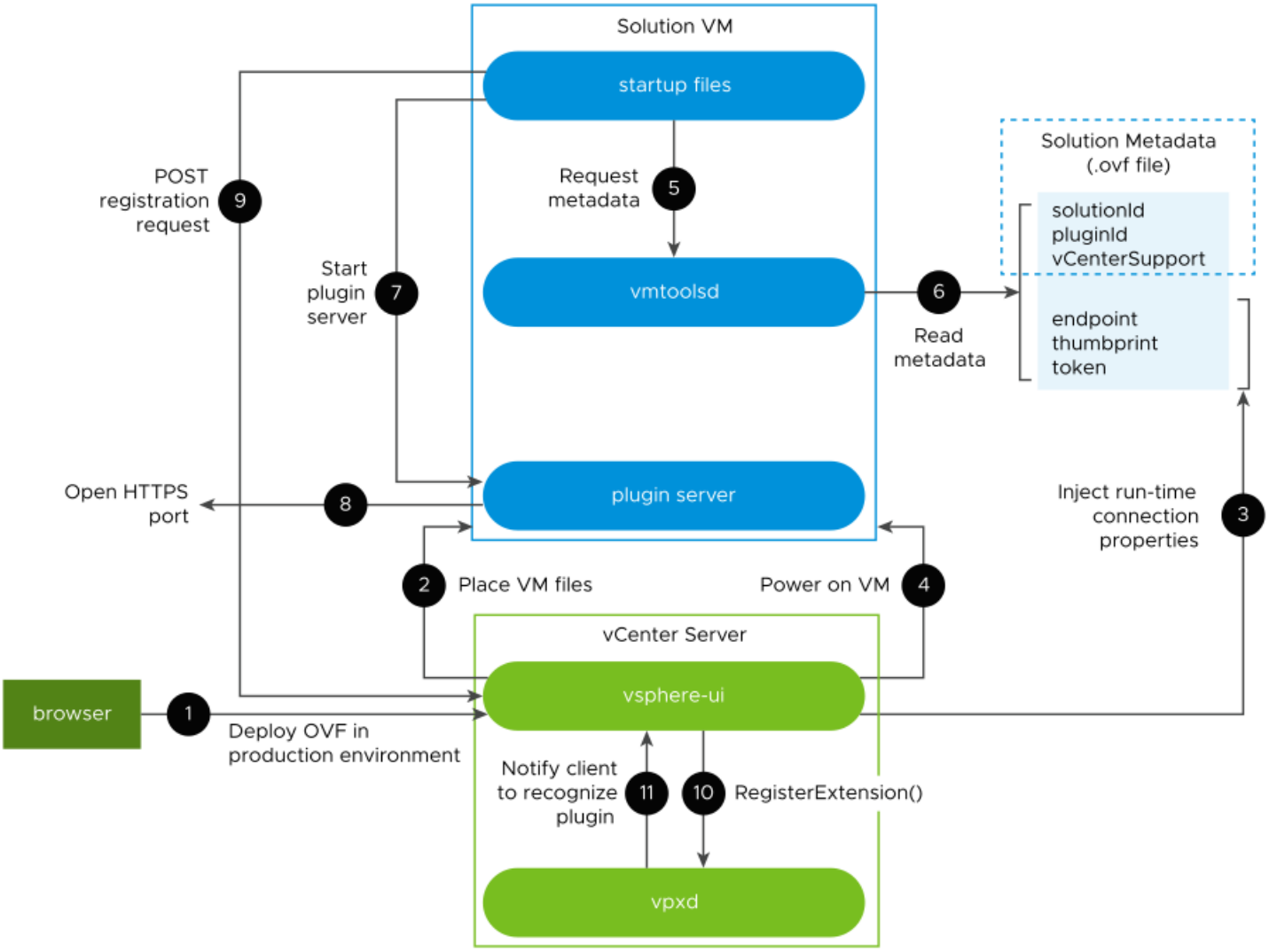
- Navigate to **Administration › Client Plugins** and click the ADD button to install and launch the plug-in in the vCenter Server environment.

3. There is a specific solution installation task in the Tasks console. As of vSphere 8.0 U2, the specific solution install task shows its progress and can be cancelled.

Figure 158: Remote Sample Starter Plug-in After Deployment



When the plug-in server deploys to the destination compute resource, the vSphere Client inserts additional connection properties into the OVF. When the virtual machine starts up, these properties enable the plug-in server to connect and authenticate with vCenter Server for the self-registration operation.



Injected OVF Properties

Here are properties injected into the OVF:

- `solutionInstall.token` – Solution install token to ensure trust. A simple base64 token that is issued by vSphere UI, cached and then validated upon receiving a request from the newly deployed Solution Manager. It has 30 minutes timeout after the VM has been deployed and is one-time use.
- `solutionInstall.postDeployData.endpoint` – Endpoint to call back after solution appliance is deployed.
- `solutionInstall.postDeployData.thumbprint` – Server thumbprint of the endpoint to call back after solution appliance is deployed. Must be verified when making a call to the endpoint.

Since vSphere 8.0 U2 (`<index>` = zero indicates first vCenter and increases to N-1):

- `solutionInstall.targetVcenterServers.count` – number of associated vCenter Servers
- `solutionInstall.targetVcenterServers.<index>.hostname` – vCenter FQDN
- `solutionInstall.targetVcenterServers.<index>.guid` – vCenter GUID
- `solutionInstall.targetVcenterServers.<index>.thumbprint` – vCenter thumbprint

As of vSphere 8.0 U3:

- `solutionInstall.targetVcenterServers.<index>.version` – vCenter version number
- `solutionInstall.postDeployData.version` – Version of the endpoint to call back after solution appliance is deployed (vSphere Client version).

Solution Install API

The solution's appliance VM is responsible for providing information to the vSphere Client about its plug-in extension. Once the special parameters are injected and the VM is powered on, the vSphere Client waits to receive a call from the solution appliance on the `solutionInstall.postDeployData.endpoint` (that is, *`vCenterIP`*`/api/ui/solutioninstall`). The `solutionInstall.token` should also be present in the headers (`solution-install-tokenheader`) of the request. Endpoint definition:

```
POST /api/ui/solutioninstall
```

Header	Value	Summary
solution-install-token	String	One-off base64-encoded token issued by vSphere UI, used to establish trust

Body payload JSON follows. For more details, see data object `Extension` in the *`Web Services API Reference`*. All fields down to `serverThumbprint` are required, including ones not so listed in the API reference, such as `company`, `client`, and `server`. Description `summary` is displayed in the Client Plug-ins view; `label` is currently not used. Server `type` and `adminEmail` are optional. If the solution should be available in the Solution Manager UI, set `shownInSolutionManager` true. Format of `privilegeList` and `resourceListmodule` are as prescribed in the `Extension` data object. If you want to add use cases and payload types later, set `additionalProperties` true.

```
{
  "vcenterExtensions": [
    {
      "extension": {
        "key": "<plugin id>",
        "version" : "<plugin version in format a.b.c.d>",
        "company" : "<company name>",
        "description" : {
          "label": "<plugin extension name>",
          "summary": "<plugin extension summary>"
        },
        "client": [
          { "type" : "vsphere-client-remote",
            "url": "<plugin manifest uri>" },
          ...
        ],
```

```
"server": [
  { "url": "<server1 uri>",
    "serverThumbprint": "<server1 thumbprint>",
    "type": "<server purpose type>"
    "adminEmail": "<email>" },
  ...
],
"shownInSolutionManager": <true|false>,
"faultList": [
  { "faultID": "<fault-id>" },
  { "faultID": "<fault-id>" }
],
"taskList": [
  { "taskID": "<task-id>" },
  { "taskID": "<task-id>" }
],
"eventList": [
  { "eventID": "<event-id>" },
  { "eventID": "<event-id>" }
],
"privilegeList": [
  { "privID": "<priv-group-name>.<priv-id>",
    "privGroupName": "<priv-group-name>" },
  { "privID": "<priv-group-name>.<priv-id>",
    "privGroupName": "<priv-group-name>" }
],
"resourceList": [
  { "data": [
      { "key": "key-1", "value": "value-1" },
      { "key": "key-2", "value": "value-2" } ],
    "locale": "EN_US",
    "module": "<module>"
```

```
          }
        ]
      },
      ...
      additionalProperties: true
    }
  ]
}
```

As of vSphere 8.0 U3, body payload `vcenterExtensionsPerGuid` indicates which extensions to register for a given vCenter GUID. You can specify a list of extensions for each vCenter and have different extensions for every vCenter if desired. Be sure to provide extensions for every vCenter that a user can select. Body payload `vcenterExtensionsPerGuid` extends `vcenterExtensions` – do not use both. Similar format is encapsulated in GUID designation.

```
{
  "vcenterExtensionsPerGuid": {
    "vc-guid01": [
      "extension" section similar to vcenterExtensions
    ]
    "vc-guid02": [
      "extension" section like vcenterExtensions
    ]
  }
}
```

Integrated Installer Solution Metadata

When you prepare an OVF for a plug-in solution, the metadata available to the virtual machine includes solution properties that control how the virtual machine registers itself with a vCenter Server instance. These properties, with their subproperties and recommended practices, are as follows.

vmw.vsphereui.solutioninstall.solutionId

Subproperty	Required?	Value	Best Practice
Category	no	vendor-defined	vSphere Client Plugin Configuration (read-only)
	Purpose: Distinguish groups of related properties		
Label	no	vendor-defined	`Solution ID`

Table continued on next page

Continued from previous page

Subproperty	Required?	Value	Best Practice
	Purpose: User-friendly property label		
Key class ID	no		
Key ID	Fixed value	`vmw.vsphereui.solutioninstall.solutionId`	
	Purpose: Unique property identifier		
Key instance ID	no		
Description	no	vendor-defined	Include vendor identification
	Purpose: User-friendly brief description of solution purpose		
Static property	yes	true (checked)	
	Purpose: Distinguish a fixed property of the solution		
Type	Fixed value	`String choice`	
	Purpose: Data format		
User configurable	Fixed value	true (checked)	
Choice list	yes	vendor-defined	Reverse domain name notation, in quotes, single value
	Purpose: Uniquely identify solution (for single plug-in solution can be same as plug-in ID)		
Default value	yes	vendor-defined	Value from choice list
	Purpose: Assign ID value to deployed solution		

vmw.vsphereui.solutioninstall.pluginId

Subproperty	Required?	Value	Best Practice
Category	no	vendor-defined	vSphere Client Plugin Configuration (Read-only)
	Purpose: Distinguesh groups of related properties		
Label	no	vendor-defined	`Plugin ID`
	Purpose: User-friendly property label		
Key class ID	no		
Key ID	Fixed value	`vmw.vsphereui.solutioninstall.pluginId`	
	Purpose: Unique property identifier		
Key instance ID	no		
Description	no	vendor-defined	Include vendor identification
	Purpose: User-friendly brief description of plug-in or solution purpose		
Static property	yes	true (checked)	
	Purpose: Distinguish a fixed property of the solution		
Type	Fixed value	`String choice`	
	Purpose: Data format		
User configurable	Fixed value	true (checked)	

Table continued on next page

Continued from previous page

Subproperty	Required?	Value	Best Practice
Choice list	yes	vendor-defined	Reverse domain name notation, in quotes, single value
	Purpose: Uniquely identify plug-in		
Default value	yes	vendor-defined	Value from choice list
	Purpose: Assign ID value to deployed plug-in		

vmw.vsphereui.solutioninstall.vCenterSupport

Subproperty	Required?	Value	Best Practice
Category	no	vendor-defined	vSphere Client Plugin Configuration (Read-only)
	Purpose: Distinguesh groups of related properties		
Label	no	vendor-defined	`vCenter Servers registration support`
	Purpose: User-friendly property label		
Key class ID	no		
Key ID	Fixed value	`vmw.vsphereui.solutioninstall.vCenterSupport`	
	Purpose: Unique property identifier		
Key instance ID	no		
Description	no	vendor-defined	Include link to solution documentation
	Purpose: User-friendly brief description of plug-in topology		
Static property	yes	true (checked)	
	Purpose: Distinguish a fixed property of the solution		
Type	Fixed value	`String choice`	
	Purpose: Data format		
User configurable	Fixed value	true (checked)	
Choice list	yes	`'Single'` or `'Multiple'`	Specify `'Multiple'` if shared server-side context or low request volume
	Purpose: `'Single'` restricts plug-in to registering with a single vCenter Server instance		
Default value	yes	vendor-defined	Value from choice list
	Purpose: Assign chosen value to deployed plug-in		

vmw.vsphereui.solutioninstall.requiredVcVersion

NOTE
As of vSphere 8.0 U1. Neither vSphere 8.0 nor vSphere 7.0 U3 support this property.

(optional property)

Subproperty	Required?	Value	Best Practice
Category	no	vendor-defined	vSphere Client Plugin Configuration (Read-only)
	Purpose: Distinguish groups of related properties		
Label	no	vendor-defined	`vCenter Servers Minimum Version`
	Purpose: User-friendly property label		
Key class ID	no		
Key ID	Fixed value	`vmw.vsphereui.solutioninstall.requiredVcVersion`	
	Purpose: Unique property identifier		
Key instance ID	no		
Description	no	vendor-defined	Include link to solution documentation
	Purpose: User-friendly brief explanation of reason for minimum version		
Static property	yes	true (checked)	
	Purpose: Distinguish a fixed property of the solution		
Type	Fixed value	`String choice`	
	Purpose: Data format		
User configurable	Fixed value	true (checked)	
Choice list	yes	vendor-defined	
	Purpose: Specify the minimum vCenter Server version that can support this plug-in		
Default value	yes	vendor-defined	Value from choice list
	Purpose: Assign chosen value to deployed plug-in		

vmw.vsphereui.solutioninstall.legacyExtensionId

(optional property)

Subproperty	Required?	Value	Best Practice
Category	no	vendor-defined	vSphere Client Plugin Configuration (Read-only)
	Purpose: Distinguesh groups of related properties		
Label	no	vendor-defined	Legacy Extension ID
	Purpose: User-friendly property label		
Key class ID	no		
Key ID	Fixed value	`vmw.vsphereui.solutioninstall.legacyExtensionId`	
	Purpose: Unique property identifier		
Key instance ID	no		

Table continued on next page

Continued from previous page

Subproperty	Required?	Value	Best Practice
Description	no	vendor-defined	Include vendor identification, link to documentation
	Purpose: Brief description of legacy plug-in replacement strategy		
Static property	yes	true (checked)	
	Purpose: Distinguish a fixed property of the solution		
Type	Fixed value	`String choice`	
	Purpose: Data format		
User configurable	Fixed value	true (checked)	
Choice list	yes	vendor-defined	Reverse domain name notation, in quotes, single value
	Purpose: Identify the plug-in that this plug-in replaces; facilitate check for legacy plug-in already installed		
Default value	yes	vendor-defined	Value from choice list
	Purpose: Assign chosen value to deployed plug-in		

vmw.vsphereui.solutioninstall.preventInstallationWhenLegacyExtension

(optional property)

Subproperty	Required?	Value	Best Practice
Category	no	vendor-defined	vSphere Client Plugin Configuration (Read-only)
	Purpose: Distinguish groups of related properties		
Label	no	vendor-defined	Prevent installation if legacy extension
	Purpose: User-friendly property label		
Key class ID	no		
Key ID	Fixed value	`vmw.vsphereui.solutioninstall.preventInstallationWhenLegacyExtension`	
	Purpose: Unique property identifier		
Key instance ID	no		
Description	no	vendor-defined	Include vendor identification, link to documentation
	Purpose: Brief description of legacy plug-in replacement strategy		
Static property	yes	true (checked)	
	Purpose: Distinguish a fixed property of the solution		
Type	Fixed value	`String choice`	
	Purpose: Data format		
User configurable	Fixed value	true (checked)	

Table continued on next page

Continued from previous page

Subproperty	Required?	Value	Best Practice
Choice list	Fixed value	`'true'` or `'false'`	
	Purpose: Indicate condition for replacing legacy plug already installed		
Default value	yes	vendor-defined	Value from choice list
	Purpose: Assign chosen value to deployed plug-in		

Plug-in Properties XML Example

An example of the `<ProductSection>` of an OVF file, containing the three required plug-in properties with example values, follows:

```
    <ProductSection>

      <Info>Information about the installed software</Info>

      <Product>Photonique</Product>

      <Vendor>Example Inc.</Vendor>

      <Version>4.0</Version>

      <FullVersion>4.0</FullVersion>

      <Category>vSphere Client Plugin Configuration (Read-only)</Category>

      <Property ovf:qualifiers="ValueMap{&quot;Single&quot;}" ovf:userConfigurable="true"
ovf:value="Single" ovf:type="string"
ovf:key="vmw.vsphereui.solutioninstall.vCenterSupport">

        <Label>vCenter Servers registration support</Label>

        <Description/>

      </Property>

      <Property ovf:qualifiers="ValueMap{&quot;com.example.solution.id&quot;}"
ovf:userConfigurable="true" ovf:value="com.example.solution.id" ovf:type="string"
ovf:key="vmw.vsphereui.solutioninstall.solutionId">

        <Label>Solution ID</Label>

        <Description/>

      </Property>

      <Property ovf:qualifiers="ValueMap{&quot;com.example.plugin.id&quot;}"
ovf:userConfigurable="true" ovf:value="com.example.plugin.id" ovf:type="string"
ovf:key="vmw.vsphereui.solutioninstall.pluginId">

        <Label>Plugin ID</Label>

        <Description/>

      </Property>

    </ProductSection>
```

Advanced Considerations for Remote Plug-in Servers

A remote plug-in for the vSphere Client has both a server portion and a user interface portion. You can use any coding language or framework you choose for the server portion. Your plug-in server must generally provide the following functionality:

- A web application server for both plug-in manifest file and plug-in user interface files.
- A fixed Service Provider Interface that responds to vSphere Client requests for dynamic view content such as menus.
- Data access and computation services on behalf of the plug-in user interface, if needed.
- Session cloning service to authenticate with the vCenter Server Web Services API, if needed.

Registering Auxiliary Plug-in Servers

When you register a plug-in with a vCenter Server instance, information about the plug-in server is stored in an `Extension` record with the ExtensionManager. Registration information includes endpoint addresses of the plug-in's primary server and of any auxiliary servers that the plug-in comprises.

The `Extension` record stores descriptors for the plug-in servers in the `server` property, which is an array of `ServerInfo` objects. The first element of the array represents the manifest server. All other array elements represent auxiliary servers, in any order. Several properties of the `ServerInfo` object must be set in specific ways:

type

Contains a service identifier, which is a string value chosen by the developer. The string enables client code to distinguish between auxiliary servers.

A best practice is to assign the value `MANIFEST_SERVER` to the first element of the array, to identify the manifest server for the plug-in.

url

Contains the server endpoint address. The vCenter Server reverse proxy translates the server endpoints into proxy endpoints, which the JavaScript API makes available to front-end code. In the descriptor for the primary server, which also serves the manifest file, the `url` property must contain the full path to the plug-in manifest file. Relative URLs for HTML content resolve relative to the directory that contains the manifest.

NOTE
In the descriptor for an auxiliary server that supplies filter choices for dynamic extensions, the `uri` property must end with a slash ('`/`'). See Configure Dynamic Extensions.

serverThumbprint

Each server descriptor must contain either the thumbprint or the server certificate (below) or both.

serverCertificate

The SSL certificate for the server endpoint in strict PEM format (string). In the plug-in server certificate, the server host name must be specified according to subjectAltName extension (RFC 2818, Section 3.1 Server Identity).

NOTE
When registering a plug-in, it's highly recommended to provide SSL certificates of the plug-in servers in addition to the thumbprints of the SSL certificates of the plug-in servers. Performing a full SSL certificate check during SSL handshake is more secure than performing an SSL certificate thumbprint check. Furthermore, in future vSphere releases, support for SSL certificate thumbprints will be dropped and full SSL certificates will be required.

In addition to the `server` property, the `Extension` record contains a `client` property, which is an array of `ClientInfo` objects. When you register the plug-in, you use the first element of this array as a descriptor for the plug-in as a whole. Three properties of the descriptor must be set in specific ways:

type

Must contain the value `'vsphere-client-remote'`

url

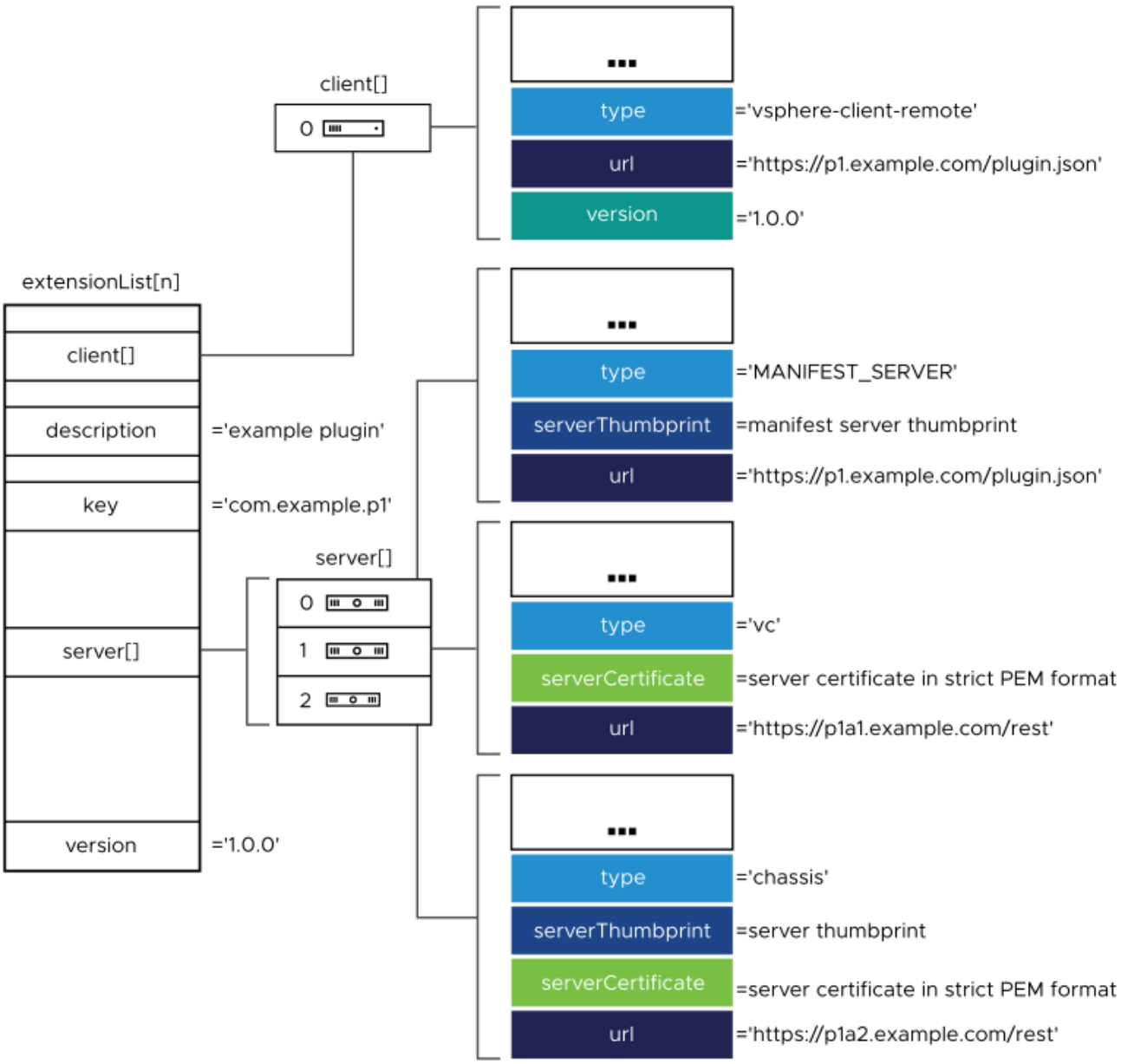
Must contain URL of the plug-in manifest file, which must match the `url` property in the first `ServerInfo` record.

version

Must contain a developer-assigned version string, which distinguishes a new plug-in version from a new plug-in instance of the same version. The version string must be an integer or a series of dot-separated integers: *n*`[.`*n*`[.`*n*`]]]`

The following illustration shows some of the basic structure of the Extension record used to register a plug-in.

Figure 159: Plug-in Registration Data



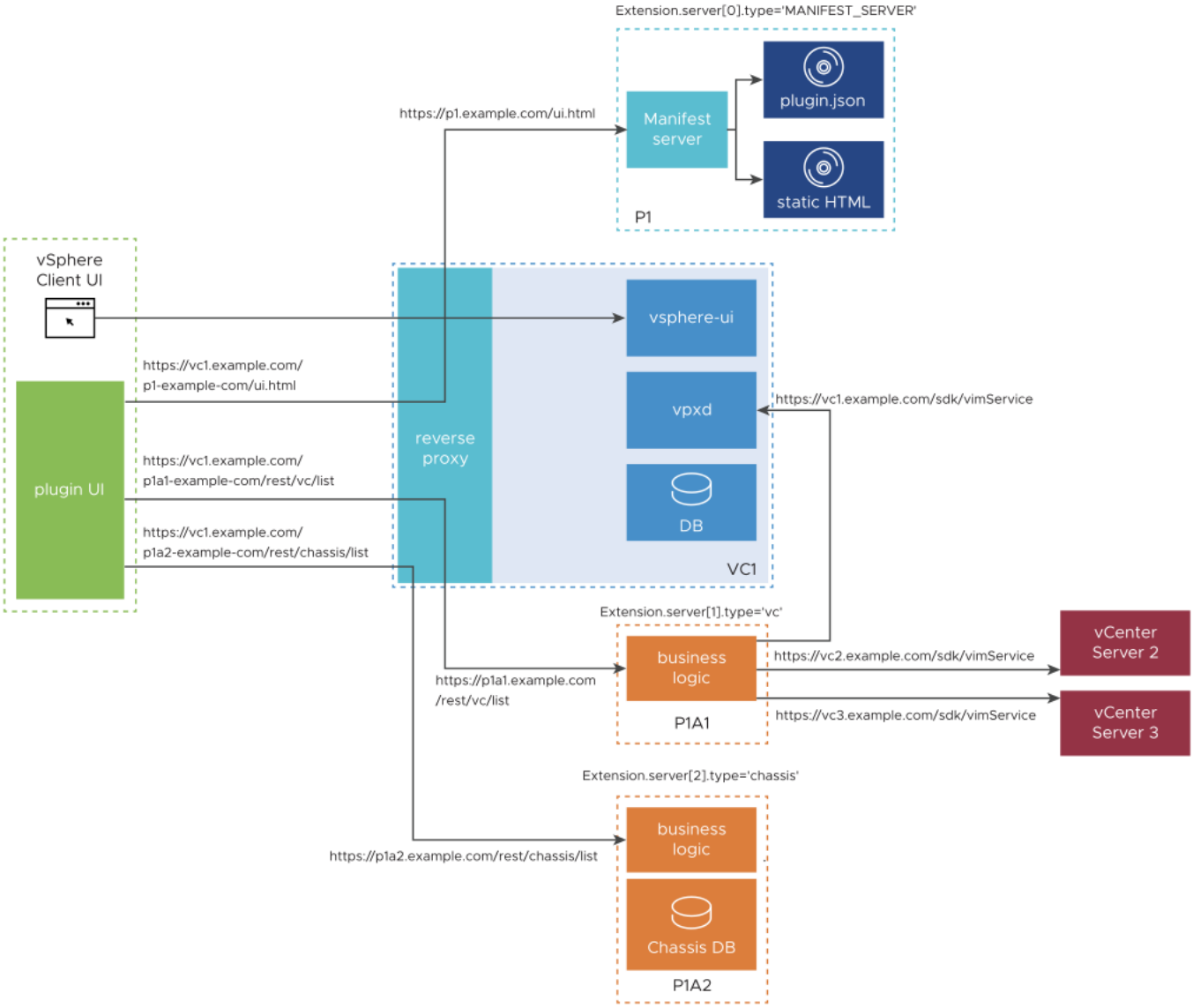
For information about how to register auxiliary plug-in servers by using the registration tool in the SDK, see vSphere Client Plug-in Registration Tool.

Auxiliary Server Data Paths

When a plug-in includes auxiliary servers, the auxiliary servers must be registered with vCenter Server, in the same Extension record as the primary server. When the plug-in user interface sends a request to an auxiliary server, the request passes through the vCenter Server reverse proxy, so that the browser treats it as having the same origin as the primary server.

The following illustration shows a hypothetical use of auxiliary plug-in servers that provide specific services supplemental to the functions of the primary server. The primary server provides the overall plug-in configuration in the manifest file, as well as the HTML components for the plug-in user interface. Separate concerns are handled by the auxiliary servers.

Figure 160: Auxiliary Server Data Paths



This hypothetical plug-in implements a Chassis abstraction, which represents a physical enclosure that contains a number of blade servers. The blades run instances of vCenter Server. The plug-in user interface presents a Chassis and its vCenter Server instances as related objects in the inventory.

Auxiliary server 1 (P1A1, in orange) interfaces with the vCenter Server instances and data centers encompassed by the plug-in. The auxiliary server uses the Web Services API to collect data from the vCenter Server instances, which it returns to the plug-in user interface. The plug-in user interface uses instance-specific data to supplement graphic and tabular displays in the UI.

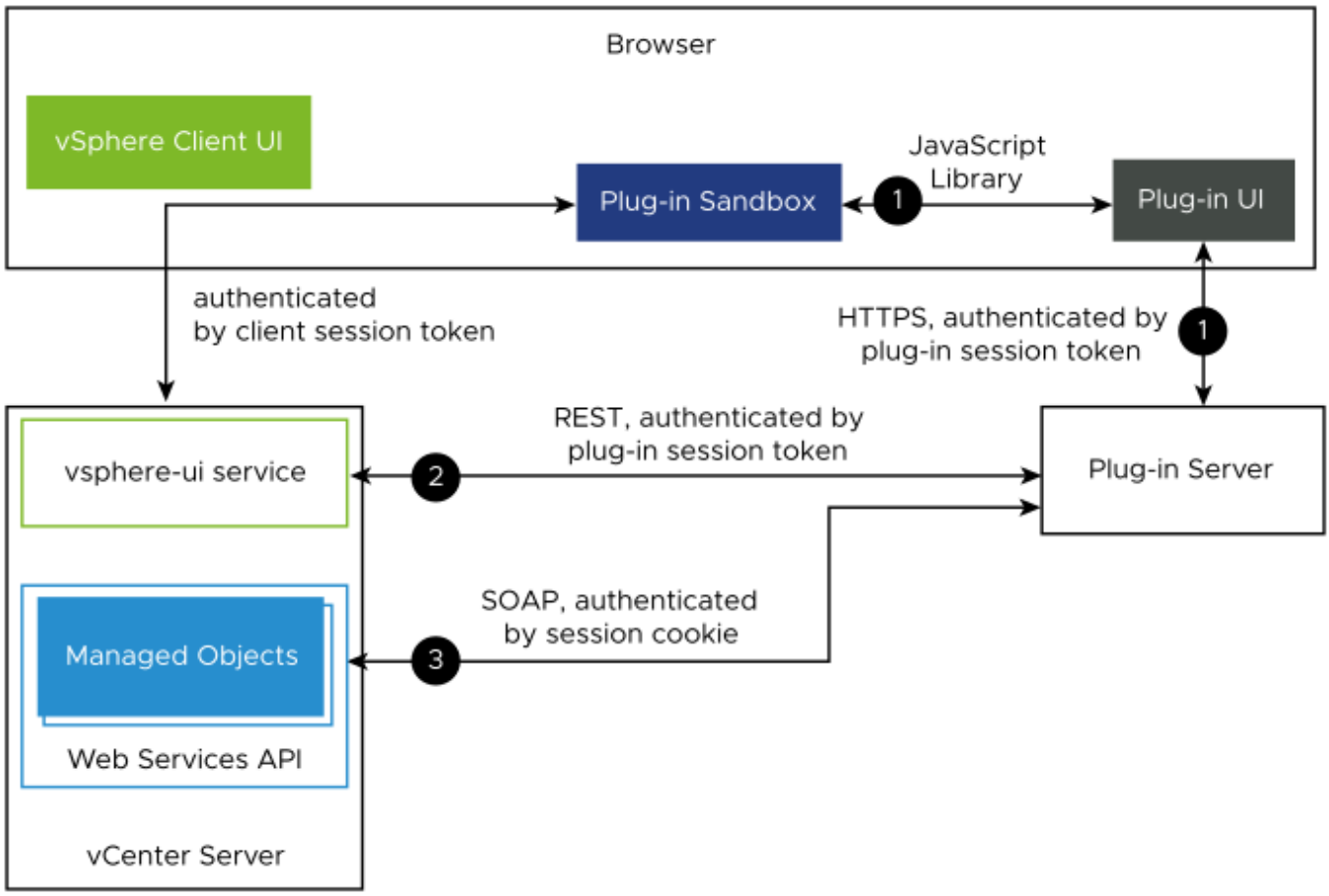
Auxiliary server 2 (P1A2, in orange) maintains a high-performance interface to a database of Chassis objects and links to related objects that represent the physical blades and VMware products that run on them. The plug-in uses this data to fill out a display of Chassis objects with related blades, vCenter Server instances, and other managed entities that populate VMware data centers.

Communication Paths for Authentication in the Remote Plug-in Server

The remote plug-in server operates outside the vCenter Server instance, and must authenticate with the Web Services API to identify and authorize its access to vSphere resources. The authentication procedure requires several steps, summarized below.

The plug-in user interface communicates with the `vsphere-ui` service through a plug-in sandbox in the browser. The plug-in sandbox uses the vSphere Client session token to authenticate with the `vsphere-ui` service in vCenter Server. The plug-in server needs to get a SOAP client session token to authenticate its operations with the Web Services API. The following diagram shows the basic communication paths involved in converting the vSphere Client session token to a plug-in server SOAP session token.

Figure 161: Plug-in Server Communication Paths for Authentication



Cloning a session consists of three stages of interactions involving the plug-in server:

1. The plug-in user interface retrieves its session ID and the GUID of the context object, then sends them to the plug-in server.
2. The plug-in server sends a REST request to vCenter Server to acquire a ticket that allows it to clone the user session.
3. The plug-in server sends a SOAP request to vCenter Server to clone the user session and acquire a new session ID.

These interactions are described in more detail in How to Delegate Session Authority to the Plug-in Server.

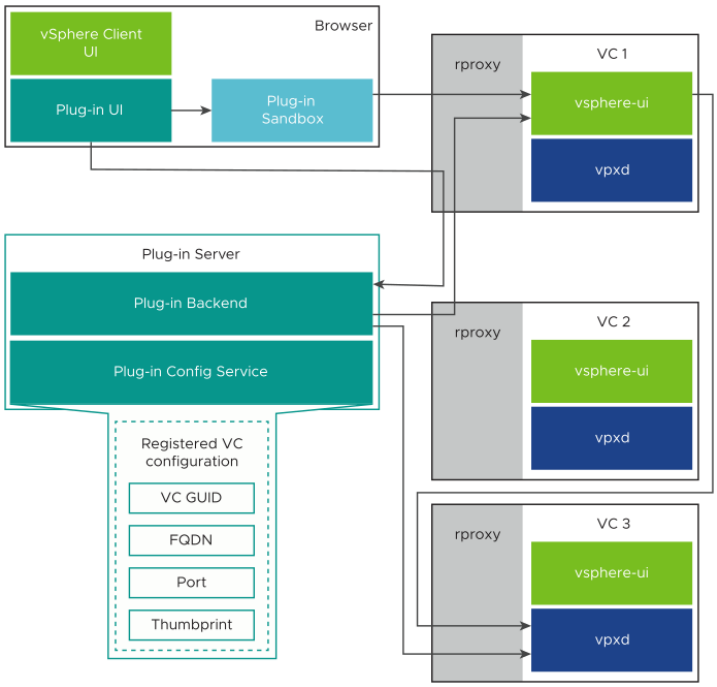
How to Delegate Session Authority to the Plug-in Server

When a plug-in server communicates with vCenter Server, it must authenticate to act on behalf of the user. This procedure shows how to delegate authority from the plug-in client code, running in a browser, to the remote plug-in server.

Before the plug-in server can establish a session with a vCenter Server, it must know the endpoint attributes for that instance. A best practice is to maintain a configuration file, for each plug-in server, that stores the attributes for each vCenter Server instance where the plug-in is registered. For example, a JSON representation of endpoint attributes might look similar to this:

```
{
  "vc1-guid": {
    "fqdn": "vc1.example.com",
    "thumbprint": "vc1-thumbprint",
    "port": 443
  },
  "vc2-guid": {
    "fqdn": "vc2.example.com",
    "thumbprint": "vc2-thumbprint",
    "port": 443
  }
}
```

In its simplest form, the delegation procedure involves communications between a plug-in front end and its sandbox environment, between the plug-in front end and the plug-in back end, and between the back end and an instance of vCenter Server. In some situations, two or three vCenter Server instances participate in the delegation procedure. The following orientation diagram shows all the processes that might work together to delegate session authority securely to the plug-in back end.



Use the following steps to delegate authority from the plug-in front end code to the plug-in server. The plug-in server can use this procedure to establish a session with any vCenter Server instance in the same ELM link group.

1. The plug-in user interface calls the `app.getSessioninfo()` method in the client JavaScript library, which in turn contacts the plug-in sandbox to request session information. The sandbox returns an object containing a `sessionToken` string, which contains a new plug-in session token that can be used for authentication by the plug-in server. The token represents the authority of the session the user established between the vSphere Client and the vCenter Server instance to which it is connected.



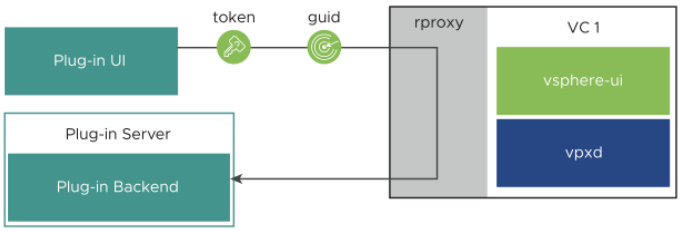
2. The plug-in user interface calls the `app.getContextObjects()` method in the client JavaScript library, which returns the ID of the object currently selected in the vSphere Client. The object ID is appended with the GUID of the vCenter Server instance that manages the object. The vCenter Server GUID is the substring following the last colon separator (:).



NOTE
If a global view is active instead of an object view, `app.getContextObjects()` returns an empty list.

3. The plug-in user interface extracts both the session key and the vCenter Server GUID, then sends them to the plug-in server, using a custom API.

NOTE
If no GUID is available, the plug-in user interface sends only the session key.



4. The plug-in back end builds a REST request to the vsphere-ui service of a vCenter Server instance. The request contains the following:
   - A `POST` verb
   - The clone-ticket endpoint path: `/api/ui/vcenter/session/clone-ticket`
   - The `Content-type` and `Accept` headers both set to `application/json`.
   - A custom header named `vmware-api-session-id`, with the session token as its value
   - A JSON object body, containing a `vc-guid` property whose value is the GUID of the vCenter Server instance



The request will look similar to this:

```
POST /api/ui/vcenter/session/clone-ticket

Content-type: application/json

Accept: application/json

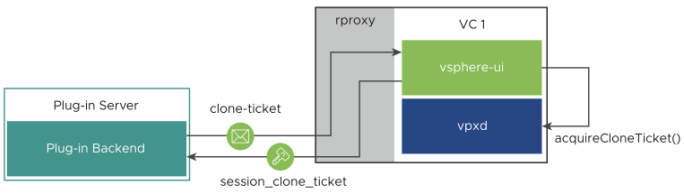
vmware-api-session-id: 12345678


{

  "vc_guid": "223b94f2-af15-4613-5d1a-a278b19abc09"

}
```

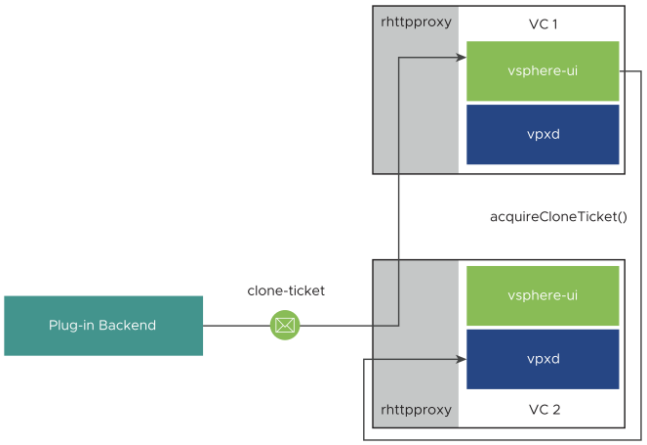
5. The plug-in back end sends the REST request to the clone-ticket REST endpoint of any vCenter Server instance, by using the endpoint attributes in the plug-in server configuration data.



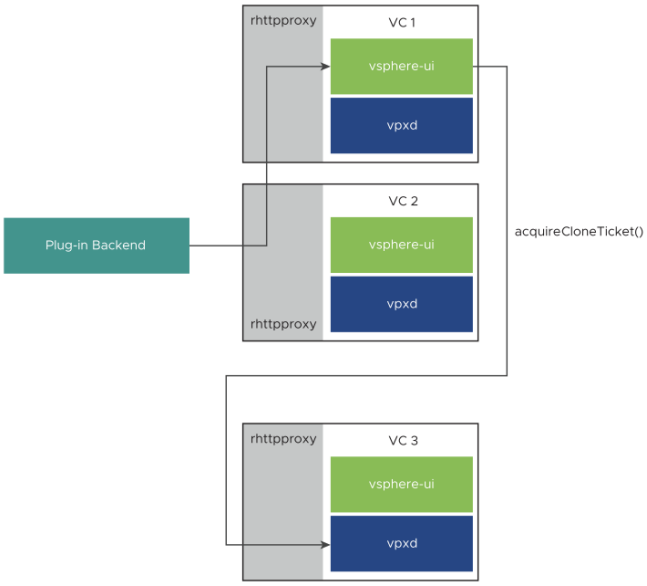
In some cases, this can trigger extra back-end communications before the plug-in back end receives a reply. For example:

a. The plug-in back end might be sending the request to a vCenter Server instance that does not hold the user session, particularly if the user's browser is connected to an instance with which the plug-in is not registered. In that case, the reverse HTTP proxy forwards the request to the vCenter Server instance that holds the user session.
In the following diagram, the browser is connected to VC1, which serves the vSphere Client in the browser. The plug-in back end chooses to contact VC2 with the clone-ticket request. VC2 forwards the request to VC1, which holds the session key.

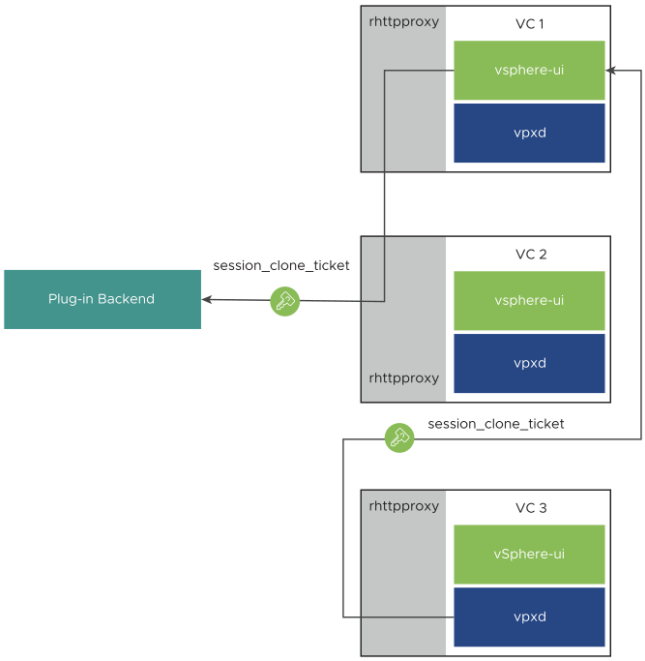
.



b. The context object the user selected might be on a third vCenter Server instance. In that case, the vSphere Client back end (the vsphere-ui process) issues a Web Services API call to the vCenter Server instance identified by the GUID in the clone-ticket request. The vsphere-ui process uses the `acquireCloneTicket()` method of the SessionManager managed object to request delegation of session authority to the process that presents the ticket to the vCenter Server instance.
In the following diagram, VC2 forwards the clone-ticket request to VC1, which holds the session with the vSphere Client in the browser. VC1 discovers that the context object is managed by VC3, so VC1 requests a clone ticket that will allow the plug-in back end to authenticate with VC3.



c. The call chain returns the ticket to the plug-in back end. The ticket gives the plug-in back end a means to establish a Web Services session with VC3.



The clone ticket is valid for the Web Services API of the vCenter Server instance associated with the context object. This is a single-use key to authenticate a call to the SessionManager.

The response will look similar to this:

```
{

  "session_clone_ticket": "cst-VCT-82cbd981-5f52-0a67-fe55-d995a7347f82--tp-B6-BC-CB-B8-59-89-C0-F2-E4-F0-C2-91-8F-28-C1-DE-10-5E-24-69"

}
```

6. The plug-in back end constructs a SOAP request to obtain a regular session ID from the Web Services API, by using the `cloneSession()` operation on the Session Manager.



The code for the SOAP request will be similar to this Java example:

```
VimService vimService = new VimService();

VimPortType client = vimService.getVimPort();

ManagedObjectReference siRef = new ManagedObjectReference();
```

```
siRef.setType("ServiceInstance");

siRef.setValue("Serviceinstance");

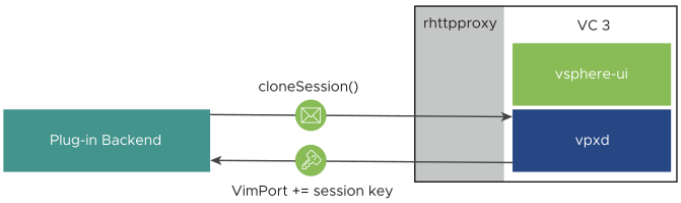
ServiceInstance si = client.createStub(ServiceInstance.class, siRef);

ServiceInstanceContent sic = si.RetrieveContent();

SessionManager mgr = client.createStub(SessionManager.class,
sic.getSessionManager());

UserSession wsSession = mgr.cloneSession(sessionCloneTicket);
```

7. The `cloneSession()` method retrieves a new multi-use session key and applies it to the linked `VimPort` object. Subsequent SOAP requests sent by means of the same `VimPort` object authenticate with the new session key.



Response Codes to session/clone-ticket Request

The REST request for a clone-ticket from the vsphere-ui service can produce the following response codes.

Table 90: Response Codes to clone-ticket Request

Response Code	Description

201	Session clone ticket was successfully created and returned.
	The body type is `application/json`, and the response body has this format:
	`{`
	`"session_clone_ticket": "string"`
	`}`

401	The request is not properly authenticated.
403	The remote plug-in is not registered with the vCenter Server instance identified by the specified UUID.
404	There is no vCenter Server instance identified by the specified UUID.

Best Practices for vSphere Client Remote Plug-ins

Sections below give tips on implementing, localizing, and deploying remote client plug-ins.

Best Practices for Implementing Plug-in Workflows

The following practices are recommended to ensure that your plug-in is stable and performs well.

- If you need to temporarily store data, use the browser cache or your own back-end server.
- Do not send calls to the topmost browser window, `window.top`, or to the parent object of your current window, `window.parent`.

- To increase security of your extensions, limit the users who can access your plug-ins, and control user access to your extensions based on their privileges. For example, you can make your extensions available only to users who have privileges to create or delete Datastore objects. For more information, see Dynamic Extension Use Cases.

Best Practices for Localization

For a partner's remote client plug-in to pass certification for vSphere 8.0, it must support all locales that vCenter supports, as below. Language localizations should be done during the development cycle to prevent delay during the certification process.

- en_US
- de_DE
- es_ES
- fr_FR
- ja_JP
- ko_KR
- zh_CN
- zh_TW
- it_IT

Best Practices for Deploying and Testing Remote Plug-ins

Apply the following advice to minimize difficulties when you deploy your plug-in or plug-ins.

- To prevent deployment issues when you try to deploy a new version of a registered plug-in, make sure that you modify the version property of your plug-in extension.
- To prevent deployment issues when you try to deploy a plug-in with the same version, make sure that you unregister the plug-in by removing the plug-in as a vCenter Server extension. You must also manually delete the cached files of the plug-in that are stored in one of the following locations:

Environment	Location of Cached Packages
vCenter Server Appliance	`/etc/vmware/vsphere-ui/vc-packages/vsphere-client-serenity/`

- To avoid performance issues, make sure that your plug-in has only one version registered with the vCenter Server. You must not change the value of the key property of the vCenter Server Extension data object between releases.
- To verify the deployment of your plug-in and monitor for any issues related to your plug-in, see one or more of the following resources:
  - The **Administration › Client Plug-ins** view in the vSphere Client.
  - The `Download Plugin` task in the Tasks console.
  - The `Deploy Plugin` task in the Tasks console.
  - The Tomcat Server log files.

  You can find the Tomcat server log files in one of the following locations:

Environment	Tomcat Log Files Location
vCenter Server Appliance 8.0 installation vSphere Client	`/var/log/vmware/vsphere-ui/logs/`

  The `vsphere_client_virgo.log` file contains the log information that the Tomcat server generates. Problems usually start with the `[ERROR]` tag. Use your plug-in name or the bundle symbolic name to detect errors caused by your plug-in.

Additional Resources

The following additional resources can help you to design or upgrade your plug-ins for the vSphere Client.

UX Design Guidelines for vSphere Client Plug-ins	Provides details on vSphere Client plug-in extensibility and UX recommendations.
vSphere Client SDK Forum	A community forum for questions on plug-in development and SDK topics.
HOL-1911-07: Hands On Lab on building a vSphere Client Plug-in	Guides the plug-in developer through all the stages of developing a local plug-in.
Upgrading Plug-ins from vSphere 6.5 to vSphere 6.7	Explains Spring upgrade requirements for plug-ins and how to adapt plug-ins to support all release versions.
Virgo Application Server Replacement	Provides information about the replacement of the Virgo application server with a standard Tomcat server.
vSphere Client Remote Plug-in Extensions Reference	Provides details on the remote plug-in manifest JSON format.
vSphere Client Plug-in Manifest Conversion Tool	Migrates a local plug-in XML manifest to a remote plug-in JSON manifest.
JavaScript API Migration Guide	Migrates a plug-in with Bridge JavaScript APIs to a plug-in with new JavaScript APIs.

Troubleshooting Remote Plug–ins for the vSphere Client

These topics describe common problems that users see when deploying and operating remote plug-ins. The topics describe characteristic symptoms and recommend solutions.

Some problems might be hard to distinguish because of similar symptoms. For example, several different underlying problems can cause a situation where a plug-in user interface fails to display. If a troubleshooting topic matches your symptom but the troubleshooting advice does not lead to a matching cause or a working solution, check other topics for similar symptoms.

Plug-in Does Not Appear in vSphere Client

The plug-in does not appear in the vSphere Client.

One or more of these symptoms is evident:

- A plug-in does not appear in the object navigator.
- A plug-in view does not display in the browser.

Several causes can prevent a plug-in from appearing. Try the following troubleshooting topics:

- Plug-in Thumbprint Incorrect
- Plug-in Manifest Cannot Be Parsed
- Plug-in Already Registered
- Wrong Plug-in Type
- Plug-in Version Incompatible
- Plug-in Version Incorrect Format
- Plug-in View Missing

Wrong Plug-in URL

The remote plug-in does not display in the vSphere Client user interface.

The plug-in does not appear in the object navigator, but it is listed in the vSphere Client under **Admin › Client plugins**. This problem can occur if the plug-in URL is not reachable. The plug-in registration command might have supplied an incorrect value for the `extension.url` property, which causes deployment to fail.

NOTE
vCenter Server supports only the HTTP protocol for the plug-in URL. Secure HTTP (HTTPS) is recommended for production environments.

To verify the cause, use one of the following methods:

- In the Management UI, find the plug-in in the list under **Admin › Client plugins**. Next to the list entry, look for a signpost which contains an error message saying that the plug-in failed to download:

  Error downloading plug-in. Make sure that the URL is reachable and the registered thumbprint is correct.
- If you do not find the signpost, you can search for the sample name in the log file at `/var/log/vmware/vsphere-ui/logs/vsphere-client-virgo.log` to find an error message.

1. Connect a web browser to the Managed Object Browser (MOB) of the vCenter Server with which you attempted to register the plug-in.
   The MOB URL is `https://vcenter_server_FQDN/mob`.
2. To view the ServiceContent data object click the **content** link.
3. To view the ExtensionManager managed object click the **ExtensionManager** link.
4. In the **extensionList** values, search for the extension key that you used to register the plug-in.
   The **more...** link enables you to see more of the list of extension keys.
5. Check that the `client.url` property is correct and specifies HTTP or HTTPS.
   Enter the URL in a browser address bar and verify that the browser displays the contents of the `plugin.json` file.
   If the URL used in the registration command was correct, check the thumbprint by using the topic Troubleshooting: Plug-in Thumbprint Incorrect.
6. If the URL used in the registration command was incorrect, reregister the plug-in as follows:
   1. Unregister the plug-in.
      You can use the registration tool in the SDK to unregistor a plug-in. For syntax information, see vSphere Client Plug-in Registration Tool.
   2. Repeat the plug-in registration step with the correct URL.

Troubleshooting: Plug-in Thumbprint Incorrect

The remote plug-in does not display in the vSphere Client user interface.

The plug-in does not appear in the object navigator, but it is listed in the vSphere Client under **Admin › Client plugins**. This problem can occur if the `extension.thumbprint` is not valid. The plug-in registration command might have supplied an incorrect server thumbprint, which causes deployment to fail.

To verify the cause, use one of the following methods:

- In the Management UI, find the plug-in in the list under **Admin › Client plugins**. Next to the list entry, look for a signpost which contains an error message saying that the plug-in failed to download:

  Error downloading plug-in. Make sure that the URL is reachable and the registered thumbprint is correct.
- If you do not find the signpost, you can search for the sample name in the log file at `/var/log/vmware/vsphere-ui/logs/vsphere-client-virgo.log` to find an error message.

1. Connect a web browser to the Managed Object Browser (MOB) of the vCenter Server with which you attempted to register the plug-in.
   The MOB URL is `https://vcenter_server_FQDN/mob`.
2. To view the ServiceContent data object click the **content** link.
3. To view the ExtensionManager managed object click the **ExtensionManager** link.
4. In the **extensionList** values, search for the extension key that you used to register the plug-in.
   The **more...** link enables you to see more of the list of extension keys.

5. Check that the `server.serverThumbprint` property is correct.
   - The thumbprint (fingerprint) should match the characters shown in the plug-in server certificate. To find the plug-in server thumbprint, see Get Thumbprint or Certificate of Remote Plug-in Server. Character pairs should be colon separated.
   - Check for hidden characters.
   - Pairs of digits must be separated by colon separators.
   - You can register the plug-in using a SHA-256 fingerprint or a SHA-1 fingerprint. A best practice is to use SHA-256, which is more secure. SHA-1 is deprecated in favor of SHA-256.

   If the thumbprint is correct, check the URL by using the topic Wrong Plug-in URL.
6. If the thumbprint used in the registration command was incorrect, reregister the plug-in as follows:
   1. Unregister the plug-in.
      You can use the registration tool in the SDK to unregistor a plug-in. For syntax information, see vSphere Client Plug-in Registration Tool.
   2. Repeat the plug-in registration step with the correct thumbprint.

Manifest Cannot Be Parsed

The remote plug-in does not display in the vSphere Client user interface.

The plug-in does not appear in the object navigator, but it is listed in the vSphere Client under **Admin › Client plugins**. The plug-in manifest file is not valid and the plug-in deployment fails due to unsuccessful schema validation.

You can distinguish this problem by searching for this error message in the `vsphere-client-virgo.log`:

`Ignoring plugin extension.key because its JSON manifest could not be parsed.`

1. According to the exception, locate the failure in the plug-in manifest file.
2. Trigger another plug-in discovery/deployment cycle. If the redeployment functionality is not enabled, restart the vSphere Client service, vsphere-ui, to get the new plug-in manifest file changes.
3. Verify that the plug-in has been deployed, if not look for other errors in the plug-in manifest file.

Wrong Plug-in Type

The remote plug-in does not display in the vSphere Client user interface.

The plug-in does not appear in the object navigator, but it is listed in the vSphere Client under **Admin › Client plugins**. The plug-in was not registered correctly as a remote plug-in, which causes the vSphere Client to assume that it is a local plug-in. When the vSphere Client tries to download the plug-in manifest, it expects a `.zip` file but finds a `.json` file instead.

You can verify the cause of the problem by searching the `vsphere-client-virgo.log` file for an error message saying that the vSphere Client did not find a ZIP file at the plug-in manifest location:

`Couldn't open plugin zip file when trying to verify the signature of plugin extension.key:extension.version java.util.zip.ZipException: error in opening zip file.`

1. On the machine that runs the plug-in server, change to the directory that contains the plug-in manifest.
   For example: `cd samples/remote-plugin-sample/target/classes/static`
2. Unregister the plug-in.
   For example: `$tools/extension-registration.sh -action unregisterPlugin -k sample.plugin -url https://vc-svr.example.com/sdk -u administrator@vsphere.local -p secret`
3. Reregister the plug-in, specifying that it is a remote plug-in.
   For example, you can use the plug-in registration tool in the SDK: `$tools/extension-registration.sh -action registerPlugin -remote -k sample.plugin -v 1.0.0 -url https://vc-svr.example.com/sdk -u administrator@vsphere.local -p $secret -pluginUrl $pluginurl`

```
-serverThumbprint $thumbprint -c 'Example Inc.' -n 'Example Plug-in' -s 'This plug-in
is registered with the remote keyword.'
```

Plug-in Marked As Incompatible

The remote plug-in does not display in the vSphere Client user interface.

The plug-in does not appear in the object navigator, but it is listed in the vSphere Client under **Admin › Client plugins**. The plug-in is marked as incompatible in the compatibility matrix and is filtered out.

> **NOTE**
> Plug-ins can be marked as incompatible with wild characters. The plug-in itself might not be marked as incompatible, but it matches a pattern in the compatibility matrix.

In the Management UI there is a signpost for the plug-in which contains an error message saying that the plug-in is marked as incompatible and can not be deployed:

The plug-in does not claim compatibility with the current vSphere Client version. Check plug-in's interoperability matrix.

1. Open the `/etc/vmware/vsphere-ui/compatibility-matrix.xml`.
2. Mark the plug-in with `extension.key` as compatible by adding: *<PluginPackage id="extension.key" status="compatible"/>*.

Plug-in Registered with Incompatible Version

The remote plug-in does not display in the vSphere Client user interface.

The plug-in does not appear in the object navigator, but it is listed in the vSphere Client under **Admin › Client plugins**. The plug-in did not deploy because the `extension.version` is not well formatted.

The `vsphere-client-virgo.log` file contains an error message saying that the plug-in version is not well formatted:

DEPLOYMENT_FAILED: Error deploying plugin package extension.key:extension.version. Reason: Deployment error. java.lang.NumberFormatException: For input string: "some string".

1. Verify that the `extension.version` is in the format "*A.B.C.D*", where:
   - *A*, *B*, *C*, and *D* are numbers.
   - *A* is mandatory.
   - *B*, *C*, and *D* are optional.

   The following examples form valid version strings:

   - "1"
   - "2.3"
   - "5.4.2"
   - "12.3.22.4"

Plug-in View is missing in the vSphere Client

A specific view is missing in the vSphere Client.

Navigating to some extension point shows `Page Not Found`.
The view was not loaded by the vSphere Client.
Verify that the view URL is correctly defined in the plug-in manifest file.

Verify directly that the URL is reachable from the plug-in server, bypassing the vCenter Server reverse proxy.

Missing Entry in the Instance Selector

A plug-in instance is missing from an Instance Selector.

This problem can occur only in an ELM environment, where multiple vCenter Server instances are linked. The plug-in instance entry is missing because the plug-in from the vCenter Server instance did not download or deploy the plug-in correctly.

- Verify that the plug-in is registered with the vCenter Server instance.
- Use other troubleshooting topics in Plug-in Does Not Appear in vSphere Client to find out why the plug-in did not download or deploy.

Unable to Change Plug-in Manifest File

The vSphere Client does not respond to changes in a remote plug-in manifest file.

After a plug-in fails to deploy due to an invalid plug-in manifest file, the vSphere Client marks it as broken and does not attempt to retry the download even if the plug-in manifest file is later updated on the plug-in server.
After download the plug-ins are cached locally and a new download will happen if the cached manifest file is deleted. This is expected production workflow. However, development environments require additional configuration to avoid the cached plug-in manifest files.
Use the Redeploy feature to bypass the plug-in cache during development. For more information, see Redeploying Plug-ins During Development.

Troubleshooting: Problems with Registration Script in SDK

The vSphere Client SDK provides a script that you can use to register plug-ins during development. Errors in the registration command can cause several kinds of plug-in failures. Most plug-in registration failures go undetected until run time, but some failures cause the registration script to report an error.

Plug-in Already Registered

The remote plug-in registration script fails.

The plug-in registration script reports an error. If you overlook the error, the plug-in does not appear in the object navigator and the plug-in is not listed in the vSphere Client under **Admin › Client plugins**.
The extension key was previously registered with the Extension Manager. The Extension Manager rejects a new registration with a duplicate value in the extension key property.

The `extension-registration.sh` or `extension-registration.bat` script reports a message similar to the following:

```
Client received SOAP Fault from server: A specified parameter was not correct:
extension.key
```

1. Connect a web browser to the Managed Object Browser (MOB) of the vCenter Server with which you attempted to register the plug-in.
   The MOB URL is `https://vcenter_server_FQDN/mob`.
2. To view the ServiceContent data object click the **content** link.
3. To view the ExtensionManager managed object click the **ExtensionManager** link.
4. In the **extensionList** values, search for the extension key that you used in your registration command.
   The **more...** link enables you to see more of the list of extension keys.
5. If you find the extension key in use, use a different key or unregister the extension key before you retry the extension registration script.

Unable To Unregister Plug-in

A plug-in unregistration command fails.

You use the plug-in registration tool to unregister a plug-in, but the tool reports a failure to unregister the plug-in. The `--key` argument did not match the `extension.key` property of the registration record in the ExtensionManager.

1. Use the `--action unregisterPlugin` argument with a key that matches the key you used to register the plug-in. If you no longer have access to the registration command you used, you can find the extension key value from the registration record in the ExtensionManager by using the following procedure.
    1. Connect a web browser to the Managed Object Browser (MOB) of the vCenter Server with which you attempted to register the plug-in.
       The MOB URL is `https://`*`vcenter_server_FQDN`*`/mob`.
    2. To view the ServiceContent data object click the **content** link.
    3. Click the **ExtensionManager** link to view the ExtensionManager managed object.
    4. In the **extensionList** values, search for the extension key that you used in your registration command.
       The **more...** link enables you to see more of the list of extension keys.
    5. Locate the `extension.key` property of the registration record for your plug-in.
2. Retry the unregistration command, supplying a corrected `--key` argument.

Developing Local Plug-ins with the vSphere Client SDK

Developing Local Plug-ins with the vSphere Client SDK provides information about developing and deploying HTML-5 extensions to the vSphere Client user interface.

VMware provides many APIs and SDKs for different applications and goals. This documentation provides information about the extensibility framework of the vSphere Client for developers who are interested in extending the web application with custom functionality.



Intended Audience

This information is intended for anyone who wants to extend the vSphere Client with custom functionality. Users typically are software developers who use HTML and JavaScript to create graphical user interface components that work with VMware vSphere®.

VMware Technical Publications Glossary

VMware Technical Publications provides a glossary of terms that might be unfamiliar to you. For definitions of terms as they are used in VMware technical documentation, go to http://www.vmware.com/support/pubs.

About This Book

Developing Local Plug-ins with the vSphere Client SDK provides information about developing and deploying HTML-5 extensions to the vSphere Client user interface.

VMware provides many APIs and SDKs for different applications and goals. This documentation provides information about the extensibility framework of the vSphere Client for developers who are interested in extending the web application with custom functionality.



Intended Audience

This information is intended for anyone who wants to extend the vSphere Client with custom functionality. Users typically are software developers who use HTML and JavaScript to create graphical user interface components that work with VMware vSphere®.

VMware Technical Publications Glossary

VMware Technical Publications provides a glossary of terms that might be unfamiliar to you. For definitions of terms as they are used in VMware technical documentation, go to http://www.vmware.com/support/pubs.

Deprecation of Local Plug-in Architecture

As of the vSphere 8.0 release, we are announcing the deprecation of the local plug-in architecture.

The legacy local plug-in architecture was introduced with vSphere 6.5 and is compatible with versions 6.7, 7.0 and relevant updates. Since vSphere 7.0 VMware is not investing in new features for the local plug-ins as their evolution is completed.

The remote plug-in architecture was introduced with vSphere 6.7U1 and is compatible with any subsequent version. This is the only architecture that will be supported in future vSphere releases. Starting from version 7.0U1 remote plug-ins have full feature parity with local plug-ins and with vSphere 7.0U3 the capabilities they offer already surpass those of the legacy local plug-ins.

Local plug-ins will continue to function in the current release, but certain considerations apply:
- Local plug-ins will no longer be part of the vSphere Client plug-in certification program.
- Local plug-ins must be FIPS-compliant. See https://developer.vmware.com/docs/13385/preparing-local-plug-ins-for-fips-compliance
- Local plug-ins must bundle third-party libraries and install them in a separate directory. See https://vdc-download.vmware.com/vmwb-repository/dcr-public/77dd6491-bb96-47f3-8c2e-a5a4655f078b/b916bc3c-0168-4487-a9e5-c89b50373c18/Local-Plugins-Library-Isolation.pdf

For more information about the deprecation of local plug-in technology, see the following resources:
- https://core.vmware.com/blog/deprecating-local-plugins-next-step-vsphere-client-extensibility-evolution
- https://kb.vmware.com/s/article/87880
- https://blogs.vmware.com/code/2022/07/15/vsphere-client-sdk-changes-in-the-upcoming-major-release-of-vsphere/

Revision History

This *Developing Local Plug-ins with the vSphere Client SDK* is updated with each release of the product or when necessary.

This table provides the update history of *Developing Local Plug-ins with the vSphere Client SDK*.

Revision	Description
18 NOV 2022	Minor cleanup.
11 OCT 2022	• Added notice about deprecation of local plug-ins. • Added notices about upgrade requiremetns for compatibility with vsphere 8.0.
31 MAR 2021	My VMware renamed to Customer Connect.
02 MAR 2021	vSphere Client SDK 7.0 Update 2 release. • Remove confusing compatibility table in chapter 3. Replace with text.
06 OCT 2020	vSphere Client SDK 7.0 Update 1 release. • Add best practices for VimPort use. • Add best practices for SSL connections. • Add information about enabling bash shell in VCSA. • At VMware, we value inclusion. To foster this principle within our customer, partner, and internal community, we are replacing some of the terminology in our content. We have updated this guide to remove instances of non-inclusive language.
14 MAY 2020	Minor clarifications and cosmetic improvements.

Table continued on next page

Continued from previous page

Revision	Description
02 APR 2020	Changes for vSphere 7.0. • Removed support for Eclipse starting Tomcat Server. • Extended and updated JavaScript API. • Support for theme-dependent icons. • Improved discovery of new plug-ins or updates. • Redeploy button in development mode. • Changed subdirectory of `ds.properties` file.
11 APR 2019	Changes for vSphere 6.7U2. • Replaced Virgo server with Tomcat server. • Updated JavaScript API to handle UI themes. • Added information about plug-in caching. • Added information about plug-in handling in linked mode. • Revised SDK directory structure. • Added chapter about integrating Clarity themes.
16 OCT 2018	Renamed for vSphere 6.7U1 release, since remote plug-ins are documented in a parallel book.
17 APR 2018	Initial release.

About the vSphere Web Client and the vSphere Client

VMware vSphere® Web Client and the VMware vSphere® Client provide means for connecting to VMware vCenter Server® systems and managing the objects in the vSphere infrastructure.

The vSphere Client is compatible with vSphere 6.5 and later releases of vSphere. The vSphere Web Client is compatible only with releases before vSphere 7.0. VMware recommends the vSphere Client for all future plug-in development work.

Understanding the vSphere Client Architecture

The vSphere Client architecture consists of three layers: the user interface layer, the Java service layer, and the back end layer.

User Interface Layer

The user interface layer consists of an HTML platform that provides a framework for plug-in extensions displayed in a Web browser. The HTML application contains all user interface elements with which the user interacts, such as menus, commands, home screen shortcuts, and other views. You can use the user interface elements to view information about an object in the vSphere environment and to make changes to your vSphere infrastructure.

The vSphere Client platform ensures that each plug-in view is isolated from the vSphere Client application, which allows you to use the UI technology of your choice when developing HTML plug-ins. You can also use any library to implement the UI components within your views.

At the user interface layer vSphere Client plug-ins use the JavaScript API to communicate with the HTML platform components.

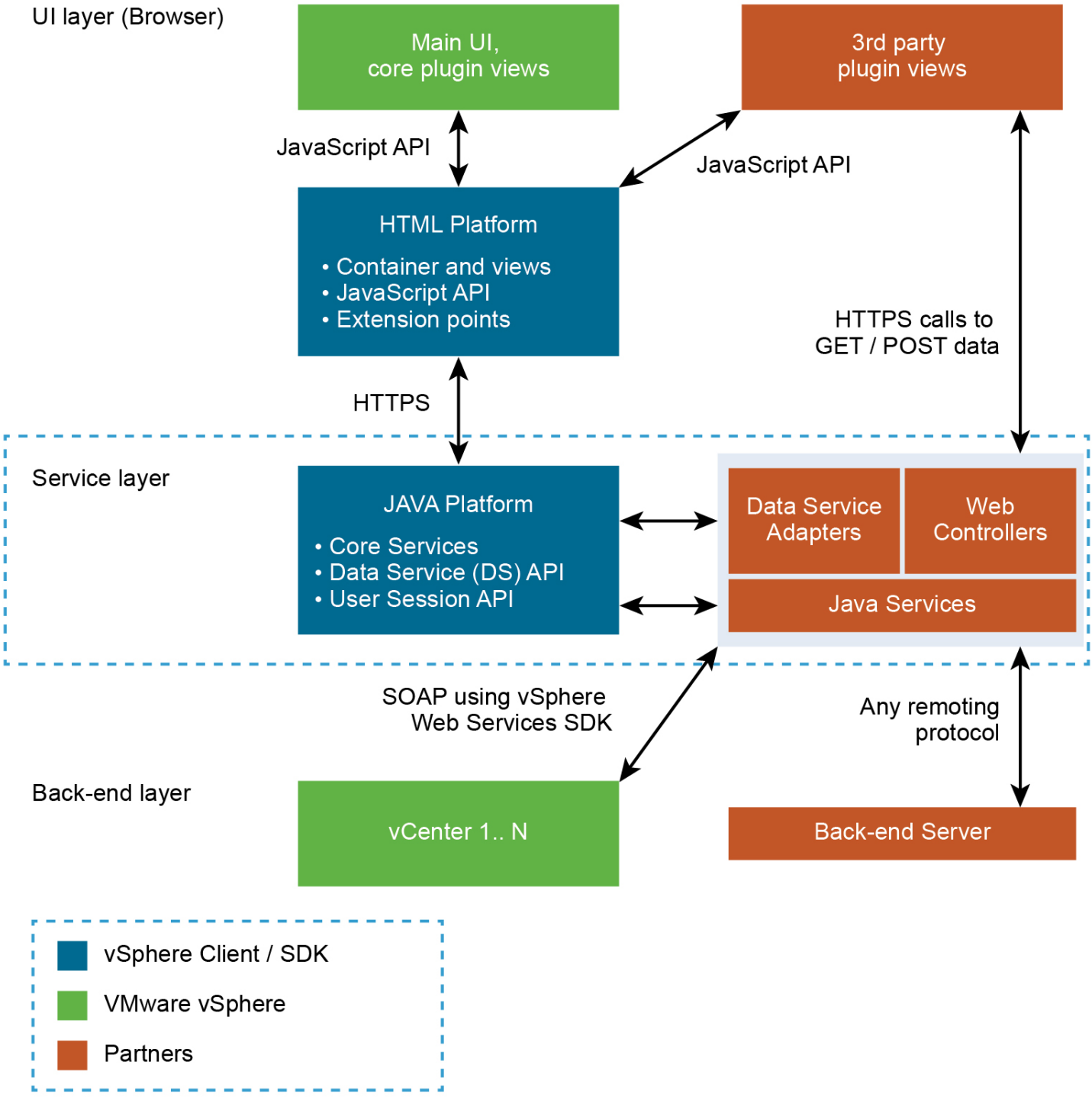
Java Service Layer

The Java service layer provides session management, a data query interface, controller components, and communication with the back end layer. HTML platform components use RESTful API calls over HTTPS to communicate with the Java platform in the service layer.

The service layer is based on the Spring MVC and the OSGI framework. Both the platform services and the vSphere Client plug-ins run in this environment.

Back End Layer

The back end layer consists of services belonging to VMware vCenter Server, and of services created by third parties. Java components in the service layer use the vSphere Web Services SDK to access one or more instances of vCenter Server, or any custom or standard API to access third party services.

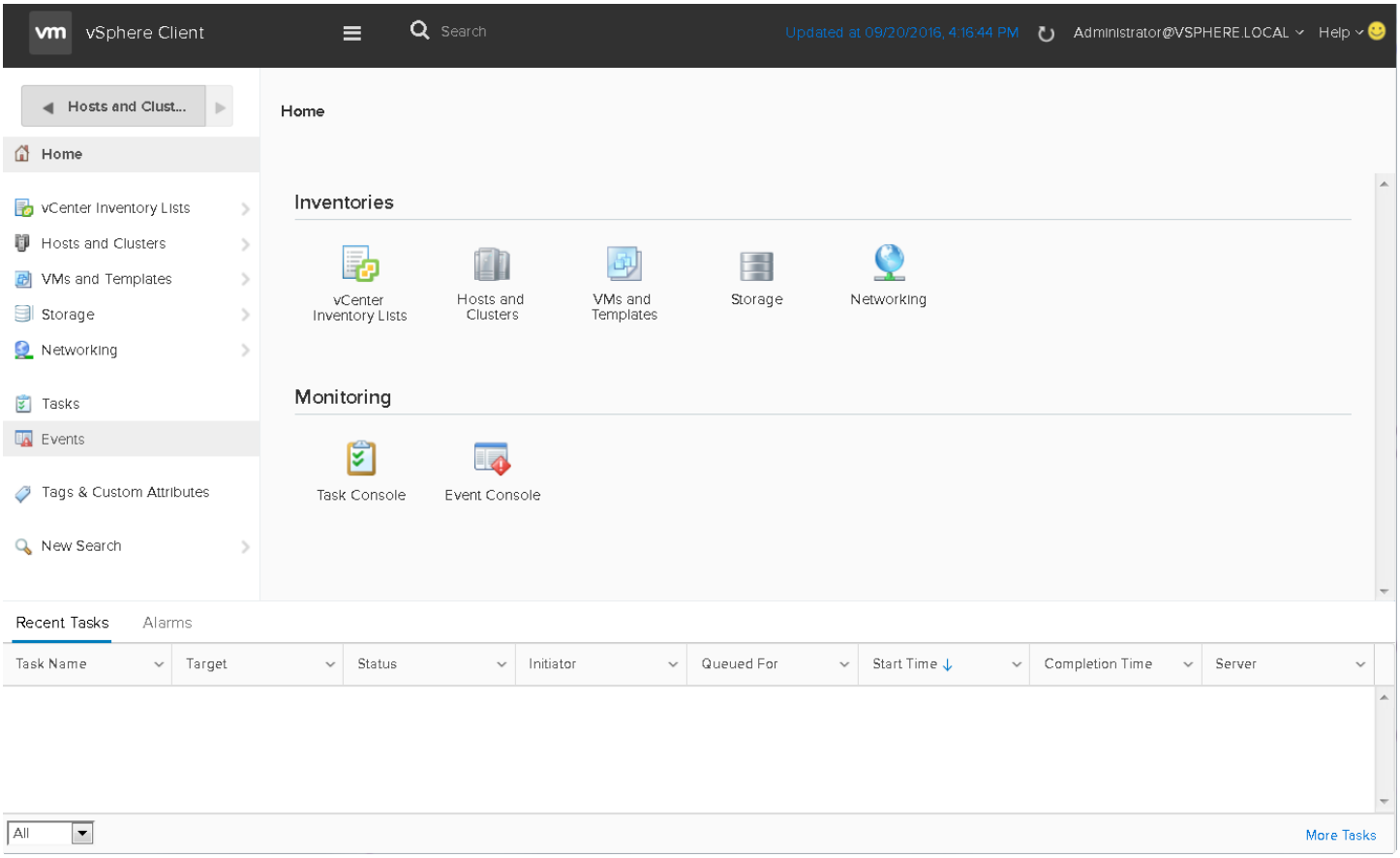


Overview of the User Interface Layer Components

The user interface layer of the Key definition for "h5client" not found in the DITA map. 6.5 contains a limited set of the views and the features that are provided by the vSphere Web Client for managing vSphere objects.

The user interface layer of the Key definition for "h5client" not found in the DITA map. contains HTML views, such as the data views, portlets, navigation options, and search bar. The Key definition for "h5client" not found in the DITA map. provides a vSphere objects navigator, the same top-level tabs for the vSphere objects in the main workspace area, and a panel that displays the recent tasks and events.

You navigate through the user interface of the Key definition for "h5client" not found in the DITA map. application in the same way as you do with the vSphere Web Client.



Understanding Extensibility in thevSphere Client

ThevSphere Client provides a modular architecture that enables plug-in developers to add new user interface elements and business logic to the VMware feature set.

When you add to the user interface layer, you create one or more extensions, which contain the HTML content that you want to visualize. This can be views, menus, or any other UI controls.

When you extend the Java service layer, you create one or more web services that provide data or perform actions on behalf of your UI extensions.

Each plug-in module extends either the user interface layer or the service layer of thevSphere Client. The user interface plug-in modules and service plug-in modules together form a complete solution to add new capabilities to thevSphere Client graphical user interfaces.

In general, you extend thevSphere Client for one of the following reasons.

- You extended the vSphere environment by adding a new type of object to the environment, or by adding more data to an existing object. If you extend vSphere in this way, you can extend thevSphere Client with new user interface elements that allow users to observe, monitor, and control these new objects.
- You extended thevSphere Client without having added new objects or data to the vSphere environment. For example, you might want to collect existing vSphere data on a single screen or location in the user interface. Shortcuts, global views, and object navigator inventory lists are examples of extensions that you can use for these purposes. You can also create a new second-level tab, portlet, or other data view that displays existing vSphere data, such as performance data, as a custom graph or chart.

Extending thevSphere Client can involve creating both user interface plug-in modules and service plug-in modules. For more information about the architecture of thevSphere Client, see Understanding the vSphere Client Architecture.

Extending the User Interface Layer

A user interface plug-in module adds one or more extensions to the Key definition for "h5client" not found in the DITA map. and the vSphere Web Client user interface layer.

Extensions to the user interface layer can include new data views, either in the virtual infrastructure or as global views. When you create a data view extension, you must also create the actual GUI objects in Adobe Flex or in HTML and package them in the plug-in module. These GUI objects rely on data from the Key definition for "h5client" not found in the DITA map. and the vSphere Web Client service layers. You can use the libraries included with the vSphere Web Client SDK to enable communication between your GUI objects and the service layer or if you create an HTML plug-in, you can use a library of your choice.

Other user interface extensions can include new workspaces for custom objects, shortcuts added to the object navigator or home screen, new relations between vSphere objects, and new actions associated with vSphere objects.

Concepts for Extending the User Interface Layer in the vSphere Client

There are three main concepts in vSphere Client UI extensibility.

Extension point
: An integration point on the vSphere Client user interface where a plug-in can hook and add its own capability.

Extension
: The UI content that you want to visualize. This can be views, menus or any other UI controls.

Extension ID
: A unique identifier that you define to refer to your extension.

Extending the Java Service Layer

You can add new Java services to the service layer. The Java services you add can perform any of the functions of a typical Java Web service.

The Java services you add to the Java service layer are used to retrieve data from the vSphere environment and display the data in the user interface layer, or to make changes to the vSphere environment in response to actions in the user interface layer.

Getting Data from the vSphere Environment

Service plug-in modules that gather data from the vSphere environment usually extend the native services on the Key definition for "h5client" not found in the DITA map. and the vSphere Web Client application servers, such as the Data Service. You can create standalone custom Java services for data gathering, but a best practice is to extend the built-in services in the vSphere Web Client SDK. Extensions to the built-in services in the vSphere Web Client SDK are often simple wrappers around existing Java services that you create.

In general, you must extend the Data Service if your extension solution meets any of the following criteria.

- Your extension provides new data about existing vSphere objects. If your extension provides a GUI element to display data that the Key definition for "h5client" not found in the DITA map. or the vSphere Web Client services do not already provide, you must extend the Data Service to provide such data.
- You want to add a new type of object to the vSphere environment. If you are adding a new type of object to the vSphere environment, you can extend the Data Service to provide data for objects of the new type.

The service extensions you create can access data from any source, either inside or outside of the vSphere environment. For example, you can create an extension to the Data Service that retrieves data from an external Web server, rather than from vCenter Server.

Making Changes to the vSphere Environment

Service plug-in modules that make changes to the vSphere environment are standalone Java services that you create. These services are used when the user starts an action in the Key definition for "h5client" not found in the DITA map. or the vSphere Web Client user interfaces. If you create an action extension, you must also create the Java service that performs the action operation on the vSphere environment as a service plug-in module.

About the vSphere Client

The VMware vSphere® Client provides a means for connecting to VMware vCenter Server® systems and managing the objects in the vSphere infrastructure. The VMware vSphere Client is an HTML5-based web application with a modular architecture that allows you to extend the functionality by means of custom plug-ins.

Knowledge Requirements for Using the vSphere Client SDK

Developing extensions for the vSphere Client by using the vSphere Client SDK, requires familiarity with HTML and JavaScript for the user interface components. The server components of remote plug-ins can be written using any technology you choose.

- You can extend the vSphere Client if you have skill with Web application development by using JavaScript and HTML. You can use any user interface technology to create views for the vSphere Client user interface layer. The sample provided within the SDK uses Angular, TypeScript, and the Clarity Design System.
- You can use any coding language you choose for server components. A remote plug-in generally places these functional requirements on the server components:
  - A web server that provides HTML, JavaScript, graphic, and localization resources for the user interface.
  - A custom API that provides controller and model functionality to support the view component.
  - Business logic components that establish Web Services API sessions with vCenter Server instances, and retrieve data to satisfy requests from the view component. The Web Services API is language agnostic, but requesters often use Java, C#, or Python bindings.

SDK Versions and Compatibility

When you write plug-ins intended to interact with different versions of vCenter Server, you need to plan for compatibility between vCenter Server, the vSphere Client SDK, and your plug-in code. Version compatibility can also become an issue when you upgrade from an older version of the vSphere Client SDK.

The vSphere Client SDK team frequently updates the SDK to contain the latest tools and samples, especially to illustrate new features and capabilities. As a result, the latest version of the SDK is not always compatible with earlier versions of vCenter Server.

Plug-in extensibility features and the vSphere Client JavaScript API remain compatible between one version of vCenter Server and the next. Your plug-in can interact with different versions of vCenter Server if it uses only the extension points and API calls supported by each version.

You can refer to the following table for more information about the compatibility of the plug-ins you developed with the different versions of the vSphere Client SDK.

NOTE
* If you have HTML-based plug-ins that are created with the vSphere Web Client SDK 6.0, you must follow the steps for upgrading your plug-in to ensure compatibility with the 6.5 versions of the vSphere Web Client and the vSphere Client.

Table 91: Compatibility Between JavaScript APIs and vSphere Client Versions

Version of the vSphere Client	vSphere 7.0/6.7/6.5 U2 JavaScript API	vSphere 6.0/6.5 Bridge API
version 6.0	No	Yes
version 6.5	No	Yes
version 6.5 U2	Yes	Yes
version 6.7	Yes	Yes
version 7.0	Yes	Yes
version 8.0	Yes	Yes

NOTE
The Bridge API is deprecated in the vSphere 6.7 release and the vSphere 6.5 U2 release. VMware will discontinue support for the Bridge API in a future vSphere Client release.

vSphere Client SDK Contents

The vSphere Client SDK contains the following directories to aid plug-in developers.

docs

Documentation for JavaScript API used by UI components of plug-ins.

libs

Run-time libraries for Spring framework and vSphere API RPC.

samples

Samples that demonstrate both client-side and server-side modules, as well as accompanying metadata.

tools

Scripts to assist with development tasks.

vsphere-ui

A complete version of the vSphere Client, both client and server modules. This directory is not relevant for remote plug-ins.

vSphere Client SDK Setup

To develop HTML plug-ins for the vSphere Client, you must first set up your development environment.

Software Requirements

You can set up your development environment for developing HTML-based plug-ins by using specific software components.

To set up your development environment, you can use the following software components with their respective versions.

Software Component	Minimum Required Version	Description
Java Standard Edition Development Kit (JDK)	1.8.x	For information about the required setup for Java development, see Setup for Java Development. The local application server runtime requires JDK 1.8.x to work with the vCenter Server 6.5 instance.
Apache Ant	1.9.x	For more information about how to use Ant to automate the build process of your plug-ins, see Automate the Plug-in Build Process.
Eclipse IDE for Java EE Developers or Spring Tool Suite	For developing HTML plug-ins, download Eclipse Neon.	For more information about how to set up the Eclipse IDE, see Set up the Eclipse Integrated Development Environment.
IntelliJ IDEA	Standard Edition	You can use the IntelliJ IDEA as an alternative to the Eclipse IDE for developing your Java and JavaScript code.

Development Environment Requirements Overview

Before you start setting up your development environment, you must download the vSphere Client SDK to your working machine and have access to a vCenter Server for Windows or a vCenter Server Appliance instance.

To create a vSphere Client plug-in, your development environment must include the following items.

- A development environment capable of developing Web applications by using JavaScript and HTML.
- A development environment capable of developing Java-based Web applications. You can use the Eclipse IDE or IntelliJ IDEA.
- Access to an instance of vCenter Server for Windows or a vCenter Server Appliance instance to register your plug-in. Plug-in registration allows the vSphere Client to download and install the plug-in.

You can set up the vSphere Client SDK on a machine with Windows or Mac OS operating systems. Before you begin the SDK setup, you can set up your Java environment and Apache Ant, and install and configure the Eclipse IDE or IntelliJ IDEA.

Setting Up for HTML-Based Plug-In Development

The vSphere Client SDK contains libraries, sample plug-ins, and various SDK tools that help you develop and build plug-ins for the vSphere Client.

Setting up your development environment for creating HTML plug-ins for the vSphere Client involves several tasks.

Set Up for Java Development

You must set up your Java development environment to create extensions to the Service Layer.

You might already have the Java platform installed on your development machine. To check the version of your Java installation, open a command prompt and enter **`java -version`**.

1. From the Oracle Web site at http://www.oracle.com/technetwork/java/javase/downloads/index.html, download the Java SE Development Kit installer.

   For developing HTML plug-ins, download JDK 1.8.x.

   Download the 64-bit version of the JDK installer if you need to allocate more memory.

2. Install the JDK following the instructions of Oracle for the operating system of your development machine.
3. Specify the location of the JDK.

Operating System	Java Location
Windows	Use the `JAVA_HOME` environment variable to specify the location of the JDK. For example, set the environment variable to `C:\Program Files\Java\jdk1.8.0_10`.
Mac OS	Open the Terminal application and enter the following command: `echo export "JAVA_HOME=\$(/usr/libexec/java_home)" >> ~/.bash_profile`. In case you have more than one Java Development Kits installed, you can specify only the version you want by using a command like the following: `echo export "JAVA_HOME=\$(/usr/libexec/java_home -v 1.8.0_17)" >> ~/.bash_profile`.

Set the Java compiler compliance level to Java 1.7 in your automation build scripts or in Eclipse, so that your plug-in will generate code compatible with older versions of vCenter Server .

Automate the Plug-In Build Process

Apache Ant is used by the scripts in the SDK to generate plug-in project templates and to build plug-ins.

Verify that you have a Java environment installed on your development machine. See Set Up for Java Development.

You can set up Apache Ant in your development environment to generate plug-in project templates and build plug-ins out of the projects. You can also build the samples provided with the vSphere Client SDK.
To use the SDK build scripts inside Eclipse, you can use the Apache Ant version provided with the Eclipse package. The following procedure sets up Apache Ant for running scripts by using the command line console.

1. From the Apache Ant site at http://ant.apache.org/bindownload.cgi, download the Apache Ant binary distribution.

   For developing HTML plug-ins, download Apache Ant 1.9.x.

2. Install Apache Ant by following the provided instructions for the operating system of your development machine.
3. Set the `ANT_HOME` environment variable to the directory on your development machine where you installed Apache Ant.

Download thevSphere Client SDK

Download the `.zip` file that contains all components of thevSphere Client SDK.

Create a Customer Connect account at https://my.vmware.com/web/vmware/.

1. Download thevSphere Client SDK from the VMware Web site at https://my.vmware.com/web/vmware/downloads.

   ThevSphere Client SDK is part of the VMware vCloud Suite and VMware vSphere, listed under Datacenter & Cloud Infrastructure.

2. Confirm the md5sum is correct.

   See the VMware Web site topic Using MD5 Checksums at http://www.vmware.com/download/md5.html.

3. Extract the content of the SDK in a directory on your development machine.

   NOTE
   The name of the directory where you extract thevSphere Client SDK must be short and without spaces.

4. Set the `VSPHERE_SDK_HOME` environment variable to the directory on your development machine where you extracted thevSphere Client SDK.
   For example:
   `VSPHERE_SDK_HOME=C:\sdk\html-client-sdk`
5. Set up the `VMWARE_CFG_DIR` environment variable on your local machine to point to one of the following directories:
   - For a Windows development environment, set `C:\ProgramData\VMware\vCenterServer\cfg\` as a value to the variable.
   - For a Mac OS development environment, set `/var/lib/vmware/vsphere-ui` as a value to the variable.

Open the `README.html` file and review the information about the other files and directories in thevSphere Client SDK.

Set Up the Eclipse Integrated Development Environment (optional)

You can use an IDE of your choice to develop custom plug-ins for the vSphere Client.

1. From the Eclipse Web site at http://www.eclipse.org/downloads/eclipse-packages/, download the Eclipse IDE for Java EE Developers package.
2. Extract the contents of the downloaded file into an appropriate location on your development machine.
3. If you do not have the minimum and maximum heap size automatically set up for Eclipse, edit the `eclipse.ini` file before you start Eclipse. You must add the location to the JDK you installed and increase the heap space and the maximum permanent space used by the JVM.

   You must add or edit the Eclipse initialization file to contain the following lines:

   ```
   -vm

   C:/<your JAVA_HOME directory>/bin/java.exe

   -Xmx1024m

   -XX:MaxPermSize=512m
   ```

4. Start Eclipse and edit the Eclipse preferences to set up your workspace for developing plug-ins for the vSphere Client.
   a) Open the Preferences dialog.
      - On a Microsoft Windows platform, choose **Window › Preferences**.

- On a Macintosh platform, choose **Eclipse › Preferences**.

The Preferences dialog opens.

b) From the General page, select the **Show heap status** option to display information about the current Java heap usage.
c) From **General › Network Connections**, configure the proxy settings to be used when opening a connection.
d) From **General › Workspace**, select the **Build automatically** and **Refresh using native hooks or polling** check boxes.
e) From **Java › Code Style › Formatter**, configure your code and naming conventions.
f) From **Java › Installed JREs**, add the location of the JDK you installed. See Set Up for Java Development.
g) From **General › Workspace › Linked Resources**, set the location of your SDK.

Set the path to the `html-client-sdk` folder as a value of the `VSPHERE_CLIENT_SDK` path variable.

h) From **Java › Build Path › Classpath Variables**, set the location of your SDK.

Set the path to the `html-client-sdk` folder as a value of the `VSPHERE_CLIENT_SDK` classpath variable.

Register Your Local vSphere Client with the vCenter Server Instance

If you want to verify your custom plug-ins, you can deploy the plug-ins first on your local vSphere Client. You must register your local instances of the Web browser applications with the vCenter Server Appliance or vCenter Server for Windows to be able to deploy your plug-ins locally.

There are two options available to register your local vSphere Client. The automatic registration option applies only to the vCenter Server Appliance. The manual registration option applies to either the vCenter Server Appliance or to vCenter Server for Windows.

Manual Registration of Local vSphere Client

You can register your local vSphere Client with the vCenter Server instance using a manual method that applies to either the vCenter Server Appliance or to vCenter Server for Windows.

- Verify that you have access to a vCenter Server instance.
- Verify that the bash shell is enabled on the vCenter Server Appliance.
  See *`Toggling the vCenter Server Appliance default shell`*, https://kb.vmware.com/s/article/2100508.

The SDK provides a registration script that you can run in the vCenter Server instance. The files generated by this script connect your local Web browser application to the remote vCenter Server system.

1. Navigate to the `vCenter registration scripts` folder under `tools` in your SDK installation.
2. Copy the `dev-setup` script to one of the following locations on the vCenter Server system depending on your vSphere deployment.
   – On the vCenter Server Appliance, use the `root` directory to copy the script. You must make the file executable.
   – On the vCenter Server for Windows, use the `C:\Users\Administrator` directory to copy the script.
3. Run the `dev-setup` script in the corresponding directory.
   The script generates the following files: `webclient.properties`, `store.jks`, and `ds.properties`.
4. Copy the generated files on your development machine in one of the following locations.

   NOTE
   On a Windows operating system, you might not be able to see the `ProgramData` folder. To change the way items are displayed on a Windows machine, use **Folder Options** from **Control Panel**.

Operating System	Generated File	Location on Your Development Machine
Windows	webclient.properties	C:\ProgramData\VMware\vCenterServer\cfg\vsphere-client\
Mac OS		/var/lib/vmware/vsphere-client/vsphere-client/
Windows	store.jks	C:\ProgramData\VMware\vCenterServer\cfg\
Mac OS		/var/lib/vmware/vsphere-client/
Windows	ds.properties	C:\ProgramData\VMware\vCenterServer\cfg\vsphere-ui\config\
Mac OS		/var/lib/vmware/vsphere-client/vsphere-ui/config/

5. If you use a Mac OS development environment, edit the `webclient.properties` file and set the `keystore.jks.path` property to point to the `/var/lib/vmware/vsphere-client/store.jks` file.
6. If you use a Mac OS development environment, edit the `ds.properties` file and set the `solutionUser.keyStorePath` property to point to the `/var/lib/vmware/vsphere-client/store.jks` file.
7. If you use a Mac OS development environment, before you connect your local Web browser application to the vCenter Server system for the first time, edit the `server.xml` file. Change the value of the `certificateKeystoreFile` attribute of the `<Certificate>` element to `/var/lib/vmware/vsphere-client/store.jks`.

   You can locate the file at *`your_SDK_folder`*`/html-client-sdk/vsphere-ui/server/conf`.

8. Start the local vSphere Client by running the `startup` script located at the `bin` directory of the `server` folder.

   For example, if you use a Mac OS development environment, the script for starting the vSphere Client is located at *`your_SDK_folder`*`/html-client-sdk/vsphere-ui/server/bin`.

   NOTE
   You might need to make the script executable: `chmod +x startup.sh`

9. Open a Web browser and log into your local vSphere Client at `https://localhost:9443/ui`.
   Your local vSphere Client connects to the vCenter Server instance and displays the vSphere inventory.

You can deploy your custom plug-ins to the local vSphere Client and verify whether the plug-ins function properly in your development environment before deploying them on the remote Web browser applications.

Automated Registration of Local vSphere Client

You can register your local vSphere Client with the vCenter Server instance using an automated method that applies only to the vCenter Server Appliance.

- Verify that you have access to the vCenter Server Appliance.
- Verify that SSH is enabled on the vCenter Server Appliance.
- Verify that the bash shell is enabled on the vCenter Server Appliance.
  See *`Toggling the vCenter Server Appliance default shell`*, https://kb.vmware.com/s/article/2100508.

The SDK provides a registration script that you can run on your development machine. The script connects to the vCenter Server Appliance and configures your local Web browser application to interact with it.

1. Set the environment variable `VMWARE_CFG_DIR` to specify the directory where the script will place the configuration files it creates.

On a Windows development machine, set the variable to `C:\Program Data\VMware\vCenter Server\cfg`.

On a MacOS development machine, set the variable to `/var/lib/vmware/vsphere-client/`.

2. In a command window, navigate to the `vCenter registration scripts` folder under `tools` in your SDK installation.
3. Run the registration script with the following parameters:
   - `-vcip` *`vc server ip`* is the IPv4 address of the vCenter Server instance where you want to register your local vSphere Client.
   - `-u` *`SSH username`* is the user account to authenticate the SSH connection with the vCenter Server instance.
   - `-pw` *`SSH password`* is the password for the SSH username.
   - `-p` *`vc server ssh port`* is the port on which the vCenter Server instance serves the SSH connection. The parameter is optional. The default value is `22`.

   – On a Windows development machine, the script is `server-registration.bat`.
   – On a MacOS development machine, the script is `server-registration.sh`.

   ./server-registration.sh -vcip 192.0.2.1 -u myUser -pw myPassword -p 22

   NOTE
   To view the full list of parameters for the script, use the `--help` option.

4. Start the local vSphere Client by running the startup script located in *`your_SDK_folder`*`/html-client-sdk/vsphere-ui/server/bin`.
   – On a Windows development machine, the script is `startup.bat`.
   – On a MacOS development machine, the script is `startup.sh`.

   NOTE
   You might need to make the script executable: `chmod +x startup.sh`

5. Open a Web browser and log into your local vSphere Client at `https://localhost:9443/ui`.
   Your local vSphere Client connects to the vCenter Server instance and displays the vSphere inventory.

You can deploy your custom plug-ins to the local vSphere Client and verify whether the plug-ins function properly in your development environment before deploying them on the remote Web browser applications.

Using the vSphere Client SDK Samples

The vSphere Client SDK provides a sample to illustrate ways you can extend the vSphere Client. You can use the scripts provided in the SDK to rebuild and run the sample.

Location of Sample Plug-in in the vSphere Client SDK

When you download the vSphere Client SDK, you can find the sample plug-in in the *`your_SDK_folder`*`/html-client-sdk/samples` directory. The sample code is already built and deployed in the application server location: *`your_SDK_folder`*`/html-client-sdk/vsphere-ui/plugin-packages`.

The sample plug-in code demonstrates how you can create a custom object and a customized workspace for this object based on your business needs. The sample demonstrate best practices for single entry point, plug-in structure

organization such as a Welcome page, a Settings page, and navigation between views. The sample uses Clarity Components and a design similar to the theme of the vSphere Client. The sample also demonstrates the usage of internationalization for different locales. For more information about the Clarity Design System, see https://vmware.github.io/clarity/.

vSphere HTML SDK Sample

The vSphere Client SDK contains a sample plug-in that demonstrates the use of the following extension points.

- vise.navigator.nodespecs
- vise.global.views
- vsphere.core.menus.solutionMenus
- vise.action.sets

Build and Deploy the vSphere HTML SDK Sample Plug-in

The vSphere Client SDK contains a sample plug-in that demonstrates the use of several extension points. Use this procedure after you modify the sample.

Before you modify the sample, see vSphere Client SDK Setup for instructions to install and configure all required components. In particular, you need the following tools installed:

- `nodejs`, versions 6.9.x to 11.15.x, inclusive
- `npm` 5.x.x
- Angular-CLI

> **NOTE**
> For the vSphere 8.0 release, local plug-ins must be upgraded as follows:
>
> 1. Local plug-ins must be FIPS-compliant. See https://developer.vmware.com/docs/13385/preparing-local-plug-ins-for-fips-compliance
> 2. Local plug-ins must bundle third-party libraries and installthem in a separate directory. See https://vdc-download.vmware.com/vmwb-repository/dcr-public/77dd6491-bb96-47f3-8c2e-a5a4655f078b/b916bc3c-0168-4487-a9e5-c89b50373c18/Local-Plugins-Library-Isolation.pdf

Also you must set the following environment variables:

- Set the environment variable `ANT_HOME` to your Apache Ant folder.
- Set the environment variable `VSPHERE_SDK_HOME` to your vSphere Client SDK folder.

After you modify the sample plug-in, you must re-build and re-deploy it. To build the html-sample, do the following steps from the command line.

1. In a command shell, change to the `html-sample-ui` directory.
   `cd samples/html-sample-ui`
2. Build the modified sample.
   On a MacOS system, run `./build-plugin-package.sh`.
   On a Windows system, run `build-plugin-package.bat`.
   The build output is in `samples/html-sample-ui/target`.
3. Copy the output folders to the plugin-packages directory, confirming that you want to replace existing files.
   `cp -r samples/html-sample-ui/target vsphere-ui/plugin-packages`

   The `vsphere-ui/plugin-packages` directory is where local plug-ins are picked up automatically when the server starts.

Running the vSphere HTML SDK Sample

Use this procedure to run the vSphere HTML SDK sample.

1. Start or restart the application server from the command line
   On a MacOS system, run `vsphere-ui/server/bin/startup.sh [-debug]` in a shell window.

   The `-debug` option allows you to specify a debug port, if desired. The default, if no port is specified, is `8000`.

   On a Windows system, run `vsphere-ui/server/bin/startup.bat -debug` in a shell window.
2. Check that there are no errors in the console or in the logs.
3. Log in to your local HTML Client at https://localhost:9443/ui/.
4. Click the vSphere HTML SDK Sample entry in the **Policies and profiles** section to explore the sample functionality.

If you need to modify the logging level, modify the file `html-client-sdk/vsphere-ui/server/conf/serviceability.xml`. The XML file contains instructions to modify the logging level.

To stop running samples in your local client, delete the sample folders from the `vsphere-ui/plugin-packages` directory and restart the server.

> **NOTE**
> Do not delete other plugin packages. Do not delete the entire `plugin-packages` directory.

Creating a vSphere Client SDK Solution

After you successfully install and configure your vSphere Client development environment, you can easily create an HTML plug-in project, then build and test your plug-in with a local or remote vSphere Client.

Before Creating an HTML Plug-In

Before you create a plug-in, you must set up your development environment to use the vSphere Client SDK.

To set up your development machine, see Setting Up for HTML-Based Plug-In Development.

Create an HTML Plug-In Project

HTML plug-ins for the vSphere Client have two components. User interface components run in the Web browser and Java service components run on the Tomcat server. The vSphere Client SDK provides tools for creating an HTML plug-in project template for each of these components. You can run the plug-in project generation scripts to create an HTML plug-in project template and build a plug-in out of the project.

- Verify that you set up the correct paths for the `ANT_HOME` and `VSPHERE_SDK_HOME` environment variables. See Automate the Plug-In Build Process.

The vSphere Client SDK provides two project generation scripts which you can use depending on the operating system of your development environment. Create the project template by using the scripts provided in the *`your_SDK_folder`*`/html-client-sdk/tools/Plugin generation scripts` directory.

1. In your development environment, open a command prompt or launch the Terminal application.
2. Navigate to the `Plugin generation scripts` folder.
   On a Windows machine, the generation scripts are located at *`SDK_folder`*`\html-client-sdk\tools\Plugin generation scripts`.
3. Run the `create-html-plugin.bat` or the `create-html-plugin.sh` script depending on your OS.

4. When prompted, enter the plug-in name, the directory on your machine where the project template folder structure will be created, and the plug-in package name.

   If you do not specify a value when prompted, the generation script uses predefined default values.

The script generates two folders, `myplugin-service` and `myplugin-ui`. For more information about the contents of each folder, see Contents of the HTML Plug-In Project Template.

After you generate the HTML plug-in project template, you can build the plug-in package and test whether your plug-in works by deploying the plug-in on the vSphere Client. For detailed information, see Building a Plug-In Package from the Project Template and Testing the Generated Plug-Ins .

Contents of the HTML Plug-In Project Template

Once you create a template project for your HTML plug-in, you must be familiar with the folder structure of the project and the purpose of each file inside the project. This knowledge will help you to easily create your custom plug-ins for the vSphere Client.

The following tables contain detailed information about the structure of the UI and Java service components of the HTML plug-in project template.

UI Project Template Structure

- **myplugin-id/src/**
  - **app/**

    Main source files for the plug-in user interface, including Javascript and Typescript files.
  - **assets/**
    - **css/**

      CSS files used in the plug-in. The `css` folder contains the `plugin-icons.css` file that you can use to define the external icons.
    - **i18n/**

      Localized resources used in the plug-in.
    - **images/**

      Default images used in the plug-in.
    - **images-dark/**

      I:mages applied when the dark theme is active.
  - **main/locales/**

    String resources specific to different locales.
  - **main/webapp/**
    - **plugin.xml**

      Manifest file of the plug-in. Defines extensions and resources.
    - **META-INF/**
      - **MANIFEST.MF**

Manifest file of the WAR bundle.

- **WEB-INF/spring/**
  Spring configuration.
- **bundle-context.xml**
  Declares the service that this UI bundle uses.

Java Service Project Template Structure

- **myplugin-service/**
  - **build-java.bat**
    Windows script to generate the Java service bundle.
    build-java.sh
    MacOS script to generate the Java service bundle.
    build-java.xml
    `ant` script to generate the Java service bundle.
  - **src/main/**
    - **resources/META-INF/**
      - **MANIFEST.MF**
        The bundle manifest file.
      - **spring/**
        Spring configuration files.
    - **java/com/mycompany/myplugin/**
      - **services/**
        Interfaces and implementations for plug-in services.
      - **model/**
        Data models for plug-in services.
      - **controllers/**
        Controllers for plug-in services.

Building a Plug-In Package from the Project Template

You build an HTML plug-in from the plug-in project template by using the automation scripts provided with the SDK.

To build a plug-in package from the project template, run the `build-plugin-package.bat` or the `build-plugin-package.sh` script depending on your operating system. You can locate these scripts in the *`plugin_name`*`-ui` folder of the project template.

After you run the script, you see the *`plugin_name`* folder that contains the `plugin-package.xml` manifest file and the `plugins` folder with the WAR and JAR files generated for the UI and service components.

Plug-In Package Manifest File

The following example shows the contents of the `plugin-package.xml` manifest file that is generated for the template HTML plug-in.

```
<pluginPackage id="com.mycompany.myplugin" version="1.0.0"
      type="html" name="myplugin"
      description="Add plugin description" vendor="Add vendor"
   <dependencies>
      <pluginPackage id="com.vmware.vsphere.client" version="6.5.0" />
      <pluginPackage id="com.vmware.vsphere.client.html" version="6.5.0" />
   </dependencies>
   <bundlesOrder>
      <!-- Include a 3rd-party library (for example gson) -->
      <bundle id="com.google.gson" />
      <!-- Include my plug-in modules -->
      <!-- These are example IDs; prefix should match the package ID -->
      <bundle id="com.mycompany.myplugin.myplugin-service" />
      <bundle id="com.mycompany.myplugin.myplugin-ui" />
   </bundlesOrder>
</pluginPackage>
```

Follow these recommendations for the `plugin-package.xml` file to ensure that your plug-in can be deployed on the vSphere Client:

- Specify a unique plug-in package ID for the `id` attribute of the `pluginPackage` XML element.
- Add the `type="html"` attribute to the `pluginPackage` elements. This attribute is required if you want your plug-in to be deployed on the vSphere Client.
- Specify that your plug-in depends on the `com.vmware.vsphere.client` package and the `com.vmware.vsphere.client.html` package, version `6.5.0`. This dependency ensures that your plug-in can be deployed on the vSphere Client 6.5.
- To specify an Update release as the minimum version supported by your plug-in, you need to use a special numbering system. For example, to specify that your plug-in supports only 6.5 Update 2 or above, use the version `6.5.0.20000` for the dependency.

Testing the Generated Plug-Ins

You can verify that your plug-in packages work correctly with the vSphere Client by deploying the plug-ins on a local and remote vSphere Client.

NOTE
For the vSphere 8.0 release, local plug-ins must be upgraded as follows:

1. Local plug-ins must be FIPS-compliant. See https://developer.vmware.com/docs/13385/preparing-local-plug-ins-for-fips-compliance
2. Local plug-ins must bundle third-party libraries and installthem in a separate directory. See https://vdc-download.vmware.com/vmwb-repository/dcr-public/77dd6491-bb96-47f3-8c2e-a5a4655f078b/b916bc3c-0168-4487-a9e5-c89b50373c18/Local-Plugins-Library-Isolation.pdf

Deploy the Plug-In on a Local vSphere Client

This procedure describes how you can use the `pickup` directory to speed up your development process. You can repeat the steps for each new version of the UI and Java service components of your plug-in.

- Register the local vSphere Client with the vCenter Server instance. See Register Your Local vSphere Client with the vCenter Server Instance.
- Verify that you run successfully the automation script for generating the plug-in package folder for your plug-in. See Building a Plug-In Package from the Project Template.
- Set the option `pickup.deployer=true` in the `webclient.properties` file.

Using the `pickup` during development is convenient for debugging, but it imposes a performance penalty in production.

1. Navigate to the `plugin` folder where the WAR and JAR files of your plug-in are generated.
   For example, on a Windows machine if you used the default settings of the plug-in generation script, go to *`your_SDK_location`*`\html-client-sdk\tools\Plugin generation scripts\plugin-packages\myplugin\plugin`.
2. Start the vSphere Client application server by running the `startup` script under `bin`.
   For example, on a Windows machine you can find the `startup` script at *`your_SDK_location`*`\html-client-sdk\vsphere-ui\server\bin`.
   The string resources are reloaded when you restart the Tomcat server.
3. Copy JAR files to the `pickup` folder on the server. If the JAR files are deployed successfully, copy the WAR files to the same folder.
   For example, on a Windows machine you can paste the files in the *`your_SDK_location`*`\html-client-sdk\vsphere-ui\server\pickup` directory.
   The Tomcat server console is updated when the bundles are deployed on the local vSphere Client.
4. Refresh your Web browser at `https://localhost:9443/ui` to see the changes.

To complete the verification of your plug-in, deploy the plug-in on a remote vSphere Client.

Deploying Your Plug-In on a Remote vSphere Client

You can verify whether your custom plug-in runs as expected by deploying the plug-ins on a remote vSphere Client.

You can register your plug-ins with the remote Web browser applications by using one of the following options:
- Create an `Extension` data object and register the data object with the `ExtensionManager` by using the Managed Object Browser (MOB) of your vCenter Server instance.
- Use the vCenter Server plug-in registration tool provided with the vSphere Client SDK.

vCenter Server Plug-In Registration Tool

The vSphere Client SDK provides a tool to ease the registration of custom plug-ins with the vSphere Client. You can locate the tool at the `vCenter plugin registration` folder under `html-client-sdk\tools`.

The `prebuilt` folder contains the `extension-registration` script that allows you to register and unregister your plug-ins as extensions to the vCenter Server instance. You can also update the registration of an existing plug-in extension to vCenter Server.

The `project` folder contains the source code and build scripts for the plug-in registration tool which you can use to extend the logic of the tool.

To use the plug-in registration tool, run the script from the command line by providing the following command-line options:

```
extension-registration -action <action> [-c <company>] [-k <key>]
 [-n <name>] [-p <vc pass>] [-pu <plugin url>] [-s <summary>]
 [-show] [-st <server thumbprint>] [-u <vc user>] [-url <vc url>]
 [-v <version>]
```

Table 92: Command-Line Options for the Plug-In Registration Tool

Command-Line Option	Description
`-action` *`<action>`*	The action that the tool must perform. You can choose from the following options: • `registerPlugin` • `unregisterPlugin` • `isPluginRegistered` • `updatePlugin`

`-c`
or
`--company`
`<company>`
	The company that developed the plug-in.

`-k`
or
`--key`
`<key>`
	The unique extension key that must be the same as the plug-in package ID of your plug-in.

`-n`
or
`--name`
`<name>`
	The name of your plug-in.

`-url` *`<vc url>`*	The URL of the vCenter Server instance where you want to register your plug-in. The URL must end with `/sdk`.

`-p`
or
`--password`
`<vc pass>`
	The credentials for logging into the vCenter Server instance.

`-u`
or
`--username`
`<vc user>`	

`-pu`
or
`--pluginUrl`
`<plugin url>`
	The URL from which your plug-in package ZIP file is downloaded.

Table continued on next page

Continued from previous page

Command-Line Option	Description

`-s`
or
`--summary`
`<summary>`
	The short description of your plug-in.

`-show`
or
`--showInSolutionManager`
	The plug-in is available under **Administrator › Solutions › vCenter Server Extensions**.
	NOTE
	This option is not supported by the vSphere Client.

`-st`
or
`--serverThumbprint`
`<server thumbprint>`
	The thumbprint of the Web server hosting your plug-in package. This option is required when your plug-in package ZIP file location is a secure URL (HTTPS).

`-v`
or
`--version`
`<version>`
	The dot-separated version number of the plug-in package that is defined in the `plugin-package.xml` manifest file.

For example, to register the `com.acme.myplugin` plug-in with version `1.0.0` that is located at `https://150.20.23.254/MyPluginpackage.zip`, use the following command on a Mac OS development machine:

```
./extension-registration.sh -url https://10.23.222.35/sdk -username
administrator@vsphere.local -password administrator -action registerPlugin -key
com.acme.myplugin -version 1.0.0 -pluginUrl https://150.20.23.254/MyPluginpackage.zip
-serverThumbprint 99:FD:2B:0D:12:85:37:AA:DA:A0:08:E1:F4:3B:4A:E6:08:AC:49:CD
```

After you register your custom plug-in, log in the vSphere Client to verify that the plug-in is visible in the remote vSphere Client. You can also use the MOB of your vCenter Server instance to view all registered plug-ins.

Extension Points in the vSphere Client

The vSphere Client publishes extension points that you can use to create your extensions. The following sections contain a list of the currently supported extension points, including a brief description of each extension point and the required extension definition type.

Global Extension Points

Global extension points allow you to extend the home screen, to add a global view to the main workspace, or to control application-wide settings.

vise.global.views		
Adds a global UI view to the main area that is not related to vSphere objects. Requires a data object of type `GlobalViewSpec` with available properties:		

Table continued on next page

Continued from previous page

vise.global.views		

- `name` - user-visible name of the global view.
- `contentSpec`
  - `url` - relative URL to the HTML page that loads the view content.
  - `metadata` (optional)
    - `key` - "hasTitle"
    - `value` - "false" opens an empty iframe, without a title.

Accessibility: can be a target of any navigation request.

Example:

```
<extension id="com.vmware.samples.h5.globalview.mainView">
   <extendedPoint>vise.global.views</extendedPoint>
   <object>
      <name>My Global View</name>
      <contentSpec>
         <url>/ui/globalview/resources/mainView.html</url>
         <metadata><entry><key>hasTitle</key><value>false</value></entry></metadata>
      </contentSpec>
   </object>
</extension>
```

vise.home.shortcuts		**deprecated**

Adds a home screen shortcut to a global view or other data view.

Requires a data object of type `ShortcutSpec` with available properties:

- `name` - user-visible name of the shortcut.
- `icon` - (optional) resource ID of 32x32 shortcut icon.
- `categoryUid` - ID of the category this shortcut will be displayed in. Supported values are "`vsphere.core.controlcenter.inventoriesCategory`" and "`vsphere.core.controlcenter.monitoringCategory`".
- `targetViewUid` - identifier of the extension to navigate to when the shortcut is clicked.

Accessibility on vSphere Client: Shortcuts.

Example:

Table continued on next page

Continued from previous page

vise.home.shortcuts		**deprecated**

```
<extension id="com.vmware.samples.h5.globalview.shortcut">
   <extendedPoint>vise.home.shortcuts</extendedPoint>
   <object>
      <name>My Shortcut</name>
      <icon>#{appIcon}</icon>
      <categoryUid>vsphere.core.controlcenter.monitoringCategory</categoryUid>
      <targetViewUid>com.vmware.samples.h5.globalview.mainView</targetViewUid>
   </object>
</extension>
```

vsphere.core.objectTypes		**deprecated**

Declares UI information that is associated with a custom object type.

Requires a data object of type `com.vmware.core.specs.ObjectTypeSpec` with available properties:

- `types` - list of type names applicable to the same type info.
- `icon` - resource ID of a 18x18 icon associated with this object type.
- `label` - localized type name.
- `labelPlural` - plural of the localized type name.
- `listViewId` - (optional) ID of the list view extension used to display multiple objects of this object type. If missing or null, the default ${namespace}.list is used.

Accessibility: Not directly displayed, just declares the new object type.

Example:

```
<extension id="com.vmware.samples.chassisa.objectType">
   <extendedPoint>vsphere.core.objectTypes</extendedPoint>
   <object>
      <types>
         <String>samples:ChassisA</String>
      </types>
      <label>Chassis</label>
      <labelPlural>ChassisA's</labelPlural>
      <icon>#{chassis.icon}</icon>
```

Table continued on next page

Continued from previous page

vsphere.core.objectTypes		deprecated

```
   </object>
</extension>
```

Object Navigator Extension Points

You can extend the object navigator by creating new nodes and categories on each page. You can customize also any object collection node that you create by adding a new icon and label.

vise.navigator.nodespecs		

Adds an object collection node, category, or pointer node extension to the object navigator.

Requires a data object of type `ObjectNavigatorNodeSpec` with available properties:

- `title` - user-visible node title.
- `icon` - (optional) 18x18 node icon resource ID.
- `navigationTargetUid` - (optional) ID of the view extension to navigate to when the node is selected.
- `viewOpenedUponFocus` - (optional) open a new empty object navigator for this view.
- `parentUid` - ID of the parent extension this node will be displayed in. This can be another `vise.navigator.nodespecs` extension ID defined by your plug-in or it can be "`vsphere.core.navigator.solutionsCategory`". Accessibility: Object Navigator root.

Example:

```
<extension id="com.vmware.samples.entryPoint">
   <extendedPoint>vise.navigator.nodespecs</extendedPoint>
   <object>
      <title>ChassisA Category</title>
      <parentUid>vsphere.core.navigator.solutionsCategory</parentUid>
      <navigationTargetUid>com.vmware.samples.htmlsample.welcomeView</
navigationTargetUid>
   </object>
</extension>
```

vise.inventory.representationspecs		deprecated

Defines one or more new icon and label sets for an object collection node in the object navigator, along with the conditions under which the icon and label sets appear.

Table continued on next page

Continued from previous page

vise.inventory.representationspecs		**deprecated**

Requires a data object of type `ObjectRepresentationSpec` with available properties:

- `objectType` - type of objects to which the specs apply.
- `specCollection` - array of IconLabelSpec objects, each of which contains:
  - `iconId` - (optional) 18x18 icon resource ID.
  - `labelId` - (optional) label or its resource ID.
  - `conditionalProperties` - (optional) array of property names. The icon and label are applicable only if the values of all properties evaluate to "true". Note: To test for "false" use the negation operator "!" in front of the property name.
  - `conditions` - (optional) array of PropertyConstraint-s. The icon and label are applicable only if all constraints are satisfied.

Accessibility: Object Navigator → Global Inventory Lists.

Example:

```
<extension id="com.vmware.samples.chassisa.iconLabelSpecCollection">
   <extendedPoint>vise.inventory.representationspecs</extendedPoint>
   <object>
      <objectType>samples:ChassisA</objectType>
      <specCollection>
         <com.vmware.ui.objectrepresentation.model.IconLabelSpec>
            <iconId>#{chassis}</iconId>
         </com.vmware.ui.objectrepresentation.model.IconLabelSpec>
      </specCollection>
   </object>
</extension>
```

Object Workspace Extension Points

Each vSphere object type's object workspace provides a set of extension points. Each extension point corresponds to a specific data view, such as the **Summary** tab view or the **Configure** tab view. Every object workspace extension point requires a data object of type `com.vmware.ui.views.ViewSpec`.

Most object workspace extension points follow the format `vsphere.core.${objectType}.${view}`. The `${objectType}` placeholder corresponds to the type of vSphere object, and the `${view}` placeholder corresponds to the specific view. For example, the extension point `vsphere.core.cluster.manageViews` is the extension point for the **Configure** tab view for Cluster objects. The following names are valid `${objectType}` values.

- `cluster`: `ClusterComputeResource` object
- `datacenter`: `Datacenter` object
- `dscluster`: `StoragePod` object

- `dvs`: `DistributedVirtualSwitch` object
- `dvPortgroup`: `DistributedVirtualPortgroup` object
- `folder`: `Folder` object
- `host`: `HostSystem` object
- `hp`: `HostProfile` object
- `network`: `Network` object
- `resourcePool`: `ResourcePool` object
- `datastore`: `Datastore` object
- `vApp`: `VirtualApp` object
- `vm`: `VirtualMachine` object
- `template`: Virtual Machine template object

vsphere.core.${objectType}.summarySectionViews.html		

Adds an HTML portlet to the **Summary** tab view.

Requires a data object of type `ViewSpec` with available properties:

- `name` - user-visible name of the global view.
- `icon` - (optional) 18x18 portlet icon resource ID.
- `contentSpec`
  - `url` - relative URL to the HTML page that loads the view content.
  - `dialogTitle` - portlet title.
  - `size` - width and height of the portlet content area:
    - `width` - can only be `1`, the default width.
    - `height` - can only be `1` or `2`, multiples of the default height.

NOTE
The portlet content area does not include the title size. The title size cannot be modified.

Accessibility: {vSphere object} → Summary page.

Example:

```
<extension id="com.vmware.samples.vspherewssdk.vm.summary2">
   <extendedPoint>vsphere.core.vm.summarySectionViews</extendedPoint>
   <object>
      <name>#{summaryView.title}</name>
      <contentSpec>
         <url>/ui/vspherewssdk/resources/vm-summary.html</url>
         <dialogTitle>WSSDK Summary Sample</dialogTitle>
         <size>
            <width>1</width>
```

Table continued on next page

Continued from previous page

vsphere.core.${objectType}.summarySectionViews.html

```
            <height>2</height>
         </size>
      </contentSpec>
   </object>
</extension>
```

vsphere.core.${objectType}.monitorCategories

Adds a sub-view category to the **Monitor** tab view.

Requires a data object of type `CategorySpec` with available properties:

- `label` - user-visible name of the Monitor view category.

Accessibility: {vSphere object} → Monitor page

Example:

```
<extension id="com.vmware.samples.vspherewssdk.vm.monitor.category">
   <extendedPoint>vsphere.core.vm.monitorCategories</extendedPoint>
   <object>
      <label>WSSDK Category</label>
   </object>
</extension>
```

vsphere.core.${objectType}.monitorViews

Adds a sub-view to the **Monitor** tab view.

Requires a data object of type `ViewSpec` with available properties:

- `name` - user-visible name of the Monitor view.
- `categoryUid` - (optional) ID of the category this Monitor view belongs to.
- `contentSpec`
  - `url` - relative URL to the HTML page that loads the view content.

Accessibility: {vSphere object} → Monitor page

Table continued on next page

Continued from previous page

vsphere.core.${objectType}.monitorViews

Example:

```
<extension id="com.vmware.samples.vspherewssdk.vm.monitor">
   <extendedPoint>vsphere.core.vm.monitorViews</extendedPoint>
   <object>
      <name>Monitor view</name>
      <categoryUid>com.vmware.samples.vspherewssdk.vm.monitor.category</categoryUid>
      <contentSpec>
         <url>/ui/vspherewssdk/resources/vm-monitor.html</url>
      </contentSyec>
   </object>
</extension>
```

vsphere.core.${objectType}.manageCategories

Adds a sub-view category to the **Configure** tab view.

Requires a data object of type `CategorySpec` with available properties:

- `label` - user-visible name of the Configure view category.

Accessibility: {vSphere object} → Configure page

Example:

```
<extension id="com.vmware.samples.vspherewssdk.vm.manage.category">
   <extendedPoint>vsphere.core.vm.manageCategories</extendedPoint>
   <object>
      <label>WSSDK Category</label>
   </object>
</extension>
```

vsphere.core.${objectType}.manageViews

Adds a sub-view to the **Configure** tab view.

Table continued on next page

Continued from previous page

vsphere.core.${objectType}.manageViews		

Requires a data object of type `ViewSpec` with available properties:

- `name` - user-visible name of the Configure view.
- `categoryUid` - (optional) ID of the category this Configure view belongs to.
- `contentSpec`
  - `url` - relative URL to the HTML page that loads the view content.

Accessibility: {vSphere object} → Configure page

Example:

```
<extension id="com.vmware.samples.vspherewssdk.vm.manage">
   <extendedPoint>vsphere.core.vm.manageViews</extendedPoint>
   <object>
      <name>Configure view</name>
      <categoryUid>com.vmware.samples.vspherewssdk.vm.manage.category</categoryUid>
      <contentSpec>
         <url>/ui/vspherewssdk/resources/vm-configure.html</url>
      </contentSpec>
   </object>
</extension>
```

vise.relateditems.specs		**deprecated**

Creates a new relation between object types, either vSphere objects or custom objects

Requires a data object of type `ObjectRelationSetSpec` with available properties:

- `type` - vSphere/Custom object type.
- `relationViewId` - ID of a view that can display object relations.
- `conditionalProperty` - (optional) property name to introduce additional constraints on the object type for a relation.
  Note: To test for "false" use the negation operator "!" in front of the property name.
- `relationSpecs`
  - `id` - relation ID
  - `label` - user-visible label of the relation.
  - `icon` - 18x18 relation icon resource ID.

Table continued on next page

Continued from previous page

vise.relateditems.specs		**deprecated**

- `listViewId` - ID of a view that can display relation items.
- `relation` - (optional) property name wrapped into a `RelationalConstraint`.
- `inverseRelation` - (optional) property name used to check if an object applies to the relation.
- `conditionalProperty` - (optional) property name wrapped into a `PropertyConstraint`.
  Note: To test for `"false"` use the negation operator `"!"` in front of the property name.
- `targetType` - (optional) target type name used in any kind of Constraint.
- `constraint` - (optional) general constraint used in case of relations that cannot be expressed in terms of `targetType`, `relation` and `conditionalProperty`

Accessibility: {vSphere object} → {related object type} in case of single relation; {vSphere object} → More objects in case of multiple relations.

Example:

```
<extension id="com.vmware.samples.relateditems.specs.host">
   <extendedPoint>vise.relateditems.specs</extendedPoint>
   <object>
      <type>HostSystem</type>
      <relationsViewId>vsphere.core.host.related</relationsViewId>
      <relationSpecs>
         <com.vmware.ui.relateditems.model.RelationSpec>
            <id>chassisForHost</id>
            <icon>#{chassis}</icon>
            <label>Chassis relation</label>
            <relation>chassis</relation>
            <targetType>samples:ChassisB</targetType>
            <listViewId>com.vmware.samples.chassisb.list</listViewId>
         </com.vmware.ui.relateditems.model.RelationSpec>
      </relationSpecs>
   </object>
</extension>
```

vsphere.core.${objectType}.monitor.performanceViews		**deprecated**

Adds a view under the **Performance** second-level tab of the **Monitor** tab view.

Table continued on next page

Continued from previous page

vsphere.core.${objectType}.monitor.performanceViews		**deprecated**
Accessibility: {vSphere object} → Monitor → Performance		

vsphere.core.${objectType}.manage.settingsViews		**deprecated**
Adds a view under the **Settings** second-level tab of the **Configure** tab view. Accessibility: {vSphere object} → Configure → Settings		

vsphere.core.${objectType}.manage.alarmDefinitionsViews		**deprecated**
Adds a view to the Alarm Definitions element in the Issues second-level tab of the Configure tab view. Accessibility: {vSphere object} → Monitor → Alarm Definitions		

vsphere.core.${objectType}.list.columns		**deprecated**
Creates a new column in the list of vSphere objects of type `${objectType}` in the object workspace. Requires a data object of type `com.vmware.ui.lists.ColumnSetContainer`. Note: Only the XML representation is supported. Accessibility: {vSphere object list}		

Actions Extension Points

Actions are invoked in the vSphere Client from menus or toolbars. The actions extension points allow you to add actions to global or contextual menus, and to prioritize the placement of actions within menus and toolbars.

vise.actions.sets		
Defines a set of actions, each of which is represented by the class `ActionSpec`. Requires a data object of type `ActionSpec` with available properties: • `uid` - action ID. • `label` - user-visible action label. • `actionUrl` - URL of the action target. • `dialogTitle` - target dialog title. • `dialogSize` - target dialog width and height. • `className` - accepts the following classes:		

Table continued on next page

Continued from previous page

vise.actions.sets

- `com.vmware.vsphere.client.HtmlPluginModalAction` - opens a modal dialog by using the JavaScript API method `modal.open()`.
- `com.vmware.vsphere.client.HtmlPluginHeadlessAction` - initiates a function call with no associated UI view.
- `com.vmware.vsphere.client.htmlbridge.HtmlActionDelegate` - opens a modal dialog or initiates a headless function using the deprecated `htmlbridge` JavaScript API.

Accessibility: {object} → {menu} → {plugin sub-menu}

Example:

```
<extension id="com.vmware.samples.htmlsample.vmActionSet">
  <extendedPoint>vise.actions.sets</extendedPoint>
  <object>
    <actions>
      <com.vmware.actionsfw.ActionSpec>
        <uid>com.vmware.samples.htmlsample.vm.action</uid>
        <label>#{action1.label}</label>
        <delegate>
          <className>com.vmware.vsphere.client.HtmlPluginModalAction</className>
            <object><root>
              <actionUrl>/ui/html-sample/index.html?view=vm-action-modal</actionUrl>
              <dialogTitle>#{actionModelTitle}</dialogTitle>
              <dialogSize>500,250</dialogSize>
            </root></object>
        </delegate>
      </com.vmware.actionsfw.ActionSpec>
    </actions>
  </object>
  <metadata>
    <objectType>VirtualMachine</objectType>
  </metadata>
</extension>
```

vmware.prioritization.listActions		deprecated

Defines and prioritizes global list actions (not related to a particular object).

Requires a data object of type `ActionPriorityGroup` with available properties:

- `prioritizedIds` - list of action IDs to declare as global.
  Note: The vSphere HTML Client does not support action prioritization.
- `regionId` - ID of the extension that contains the global actions.

Accessibility: {object list} → {action button bar} and {list menu}

Example:

```
<extension id="com.vmware.sample.chassis.listAction">
   <extendedPoint>vmware.prioritization.listActions</extendedPoint>
   <object>
      <prioritizedIds>
         <String>com.vmware.samples.chassisa.createChassis</String>
      </prioritizedIds>
      <regionId>com.vmware.samples.chassisa.list</regionId>
   </object>
</extension>
```

vsphere.core.menus.solutionMenus		

Defines a custom sub-menu including actions, separators, and nested menus.

Requires a data object of type `ActionMenuItemSpec` with available properties:

- `uid` - menu item ID.
- `type` - type of menu item. Supported values are `"menu"`, `"action"` and `"separator"`.
- `label` - (optional) user-visible label of the menu item.
  If omitted and the type is `"action"`, the label defined in the action declaration will be used.
- `icon` - (optional) 18x18 icon resource ID.
  If omitted and the type is `"action"`, the icon defined in the action declaration will be used.
- `children` - (optional) array of child menu items (`ActionMenuItemSpec`) if the type is `"menu"`.

Accessibility: {object} → {menu}

Example:

Table continued on next page

Continued from previous page

vsphere.core.menus.solutionMenus

```
<extension id="com.vmware.samples.vspherewssdk.vmMenu">
   <extendedPoint>vsphere.core.menus.solutionMenus</extendedPoint>
   <object>
      <label>WSSDK menu</label>
      <children>
         <Array>
            <com.vmware.actionsfw.ActionMenuItemSpec>
               <type>action</type>
               <uid>com.vmware.samples.vspherewssdk.myVmAction1</uid>
            </com.vmware.actionsfw.ActionMenuItemSpec>
            <com.vmware.actionsfw.ActionMenuItemSpec>
               <type>separator</type>
            </com.vmware.actionsfw.ActionMenuItemSpec>
            <com.vmware.actionsfw.ActionMenuItemSpec>
               <type>action</type>
               <uid>com.vmware.samples.vspherewssdk.myVmAction2</uid>
            </com.vmware.actionsfw.ActionMenuItemSpec>
         </Array>
      </children>
   </object>
   <metadata>
      <objectType>VirtualMachine</objectType>
   </metadata>
</extension>
```

Extension Templates

When you add custom vSphere objects, use the extension templates to make the vSphere Client user interface consistent.

vsphere.core.inventory.objectViewTemplate		deprecated

Creates a complete object workspace for a given custom object type. When you create an instance of the `objectViewTemplate`, the vSphere Client generates an extension point for each of the standard object workspace tabs, second-level tabs, and views.

Requires the following variables:

- `namespace` - plugin-specific prefix to use in all extension IDs of the template. A best practice is to use reverse domain naming, such as `com.myCompany`, to start the namespace name, followed by a unique extension name. For example, if your company name is Acme, and you create a plug-in for a custom object called Rack, you could use the namespace `com.acme.plugin01.rack`.
- `objectType` - custom object type of the instance. Should be qualified with its own namespace to avoid collisions.

The `objectViewTemplate` creates extension points in the format *`namespace.extension-point-name`*. To continue the previous example, one extension point might be `com.acme.plugin01.rack.monitorViews`.
For the full list of object workspace extension points, see Custom Object Extension Points. A given tab does not appear in the vSphere Client user interface unless you explicitly create an extension that references that tab's extension point.

Example:

```
<templateInstance id="com.vmware.samples.chassisb.viewTemplateInstance">
   <templateId>vsphere.core.inventory.objectViewTemplate</templateId>
   <variable name="namespace" value="com.vmware.samples.chassisb"/>
   <variable name="objectType" value="samples:ChassisB"/>
</templateInstance>
```

vsphere.core.inventorylist.objectCollectionTemplate		deprecated

Creates an object collection node in the object navigator for a given custom object type.

Requires the following variables:

- `namespace` - plugin-specific prefix to use in all extension IDs of the template. It must be different than the one in `objectViewTemplate`.
- `title` - custom object title or its resource ID.
- `icon` - 18x18 custom object icon resource ID.
- `objectType` - custom object type of the instance. Should be qualified with its own namespace to avoid collisions.
- `listViewId` - container view ID for the current object collection.
- `parentUid` - extension ID of the category node which the current node belongs to.

Example:

```
<templateInstance id="com.vmware.samples.lists.allChassis">
```

Table continued on next page

Continued from previous page

vsphere.core.inventorylist.objectCollectionTemplate		deprecated

```
   <templateId>vsphere.core.inventorylist.objectCollectionTemplate</templateId>
   <variable name="namespace" value="com.vmware.samples.chassisb_collection"/>
   <variable name="title" value="Chassis"/>
   <variable name="icon" value="#{chassis}"/>
   <variable name="objectType" value="samples:ChassisB"/>
   <variable name="listViewId" value="com.vmware.samples.chassisb.list"/>
   <variable name="parentUid" value="com.vmware.samples.chassisBCategory"/>
</templateInstance>
```

Custom Object Extension Points

When you instantiate a `objectViewTemplate` for your custom object, the template creates a number of extension points that you can use to fill out the user interface for the object.

The extension points that are created for a custom object include some of the listed extension points in Object Workspace Extension Points. In addition, the `objectViewTemplate` creates the following list of extension points at runtime for a particular namespace.

You can use the extension points to define views and tabs for the custom object workspace. If you want a specific view or tab to appear in the vSphere Client user interface for a custom object, you must explicitly create an extension that references the extension point of the view or tab.

${namespace}.views		deprecated

Adds a top-level tab view for custom objects.

Requires a data object of type `ViewSpec` with available properties:

- `name` - user-visible name of the Getting Started view.
- `contentSpec`
  - `url` - relative URL to the HTML page that loads the view content.

Accessibility: {custom object root}
Example:

```
<extension id="com.vmware.samples.chassisa.MainView">
   <extendedPoint>com.vmware.samples.chassisa.views</extendedPoint>
   <object>
      <name>Chassis Main</name>
      <contentSpec>
```

Table continued on next page

Continued from previous page

${namespace}.views		**deprecated**

```
         <url>/ui/chassisa/resources/chassis-main.html</url>
      </contentSpec>
   </object>
</extension>
```

${namespace}.summaryViews		**deprecated**

Adds a Summary tab view for custom objects.

Requires a data object of type `ViewSpec` with available properties:

- `name` - user-visible name of the global view.
- `icon` - (optional) 18x18 portlet icon resource ID.
- `contentSpec`
  - `url` - relative URL to the HTML page that loads the view content.
  - `dialogTitle`- portlet title.
  - `dialogSize`- portlet width and height.

Accessibility: {custom object root}
Example:

```
<extension id="com.vmware.samples.chassisa.SummaryView">
   <extendedPoint>com.vmware.samples.chassisa.summaryViews</extendedPoint>
   <object>
      <name>Chassis Summary</name>
      <contentSpec>
         <url>/ui/chassisa/resources/chassis-summary.html</url>
      </contentSpec>
   </object>
</extension>
```

${namespace}.monitorViews		**deprecated**

Adds a sub-view to the Monitor tab view for custom objects.

Table continued on next page

Continued from previous page

${namespace}.monitorViews		**deprecated**

Requires a data object of type `ViewSpec` with available properties:

- `name` - user-visible name of the Monitor view.
- `categoryUid` - (optional) ID of the category this monitor view belongs to.
- `contentSpec`
  - `url` - relative URL to the HTML page that loads the view content.

Accessibility: {custom object root} → Monitor page
Example:

```
<extension id="com.vmware.samples.chassisa .monitor">
   <extendedPoint>com.vmware.samples.chassisa.monitorViews</extendedPoint>
   <object>
      <name>Monitor view</name>
      <categoryUid>com.vmware.samples.chassisa.monitor.category</categoryUid>
      <contentSpec>
         <url>/ui/vspherewssdk/resources/vm-monitor.html</url>
      </contentSpec>
   </object>
</extension>
```

${namespace}.manageViews		**deprecated**

Adds a sub-view to the Configure tab view for custom objects.

Requires a data object of type `ViewSpec` with available properties:

- `name` - user-visible name of the Configure view.
- `categoryUid` - (optional) ID of the category this Configure view belongs to.
- `contentSpec`
  - `url` - relative URL to the HTML page that loads the view content.

Accessibility: {custom object root} → Configure page
Example:

```
<extension id="com.vmware.samples.chassisa.manage">
   <extendedPoint>com.vmware.samples.chassisa.manageViews</extendedPoint>
   <object>
      <name>Configure view</name>
```

Table continued on next page

Continued from previous page

${namespace}.manageViews		**deprecated**

```
      <categoryUid>com.vmware.samples.chassisa.manage.category</categoryUid>
      <contentSpec>
         <url>/ui/vspherewssdk/resources/vm-configure.html</url>
      </contentSpec>
</extension>
```

${namespace}.list.columns		**deprecated**

Creates a new column in the list of custom objects.

Requires a data object of type `com.vmware.ui.lists.ColumnSetContainer` which is a collection of columns with available properties:

- `headerText` - column header text.
- `requestedProperties` - object properties whose value representation will be displayed in the column (commonly a 1-element array).
- `requestedParameters` - parameters of the requested object properties.
- `sortProperty` - enables column sorting by header selection.
- `exportProperty` - enables exporting column data.

> **NOTE**
> Only the XML representation is supported.

Accessibility: {custom object list}
Example:

```
<extension id="com.vmware.samples.chassisa.list.sampleColumns">
   <extendedPoint>com.vmware.samples.chassisa.list.columns</extendedPoint>
   <object>
      <items>
         <com.vmware.ui.lists.ColumnContainer>
            <uid>com.vmware.samples.chassisa.column.name</uid>
            <dataInfo>
               <com.vmware.ui.lists.ColumnDataSourceInfo>
                  <headerText>Name</headerText>
                  <requestedProperties>
```

Table continued on next page

Continued from previous page

${namespace}.list.columns		**deprecated**

```
                    <String>name</String>
                 </requestedProperties>
                 <sortProperty>name</sortProperty>
                 <exportProperty>name</exportProperty>
              </com.vmware.ui.lists.ColumnDataSourceInfo>
           </dataInfo>
        </com.vmware.ui.lists.ColumnContainer>
        ...
     </items>
  </object>
</extension>
```

${namespace}.gettingStartedViews		**deprecated**

Adds a Getting Started tab view for custom objects.

Requires a data object of type `ViewSpec` with available properties:

- `name` - user-visible name of the Getting Started view.
- `categoryUid` - (optional) ID of the category this Getting Started view belongs to.
- `contentSpec` - parameters of the requested object properties.
  - `url` - relative URL to the HTML page that loads the view content.

Accessibility: {custom object root}

${namespace}.monitor.issuesViews		**deprecated**

Adds a sub-view under the Issues second-level tab of the Monitor tab view for custom objects.

Requires a data object of type `ViewSpec`.

Accessibility: {custom object root} → Monitor → Issues

${namespace}.monitor.performanceViews		**deprecated**

Adds a sub-view under the Performance second-level tab of the Monitor tab view for custom objects.

Table continued on next page

Continued from previous page

${namespace}.monitor.performanceViews		**deprecated**
Requires a data object of type `ViewSpec`. Accessibility: {custom object root} → Monitor → Performance		

${namespace}.monitor.performance.overviewViews		**deprecated**
Adds a sub-view under the Performance/Overview section of the Monitor tab view for custom objects. Requires a data object of type `ViewSpec`. Accessibility: {custom object root} → Monitor → Performance → Overview		

${namespace}.monitor.performance.advancedViews		**deprecated**
Adds a sub-view under the Performance/Advanced section of the Monitor tab view for custom objects. Requires a data object of type `ViewSpec`. Accessibility: {custom object root} → Monitor → Performance → Advanced		

${namespace}.monitor.taskViews		**deprecated**
Adds a sub-view under the Tasks second-level tab of the Monitor tab view for custom objects. Requires a data object of type `ViewSpec`. Accessibility: {custom object root} → Monitor → Tasks		

${namespace}.monitor.eventsViews		**deprecated**
Adds a sub-view under the Events second-level tab of the Monitor tab view for custom objects. Requires a data object of type `ViewSpec`. Accessibility: {custom object root} → Monitor → Events		

${namespace}.manage.settingsViews		**deprecated**
Adds a sub-view under the Settings second-level tab of the Configure tab view for custom objects. Requires a data object of type `ViewSpec`. Accessibility: {custom object root} → Configure → Settings		

${namespace}.manage.alarmDefinitionsViews		**deprecated**
Adds a sub-view under the Issues/Alarm Definitions section of the Configure tab view for custom objects. Requires a data object of type `ViewSpec`. Accessibility: {custom object root} → Configure → Alarm Definitions		

${namespace}.manage.permissionsViews		**deprecated**
Adds a sub-view to the Permissions tab view for custom objects. Requires a data object of type `ViewSpec`. Accessibility: {custom object root} → Configure → Permissions		

Ordering Extensions

You can use the `<precedingExtension>` element to specify the order in which the vSphere Client renders the extensions in your plug-in module.

Within each `<extension>` element, you can specify a `<precedingExtension>` element that contains the ID of another extension that is to be rendered before the current extension. Setting the value of `<precedingExtension>` to `NULL` causes that extension to be rendered first.

If no `<precedingExtension>` value is specified, the extensions are rendered in the order they appear in the `plugin.xml` module manifest file. If you specify the same value for the `<precedingExtension>` element for several extensions, the extensions are rendered in the order in which they appear in the manifest.
The following XML fragment shows how the `<precedingExtension>` element might appear in the extension definitions in the plug-in module manifest file.

```
<extension id = "com.MyPluginPackage.MyPlugin.PerformanceView">
   <extendedPoint>vsphere.core.vm.views</extendedPoint>
   <precedingExtension>NULL</precedingExtension>
   ...(extension data)...
</extension>


<extension id = "com.MyPluginPackage.MyPlugin.UtilityView">
   <extendedPoint>vsphere.core.vm.views</extendedPoint>
   <precedingExtension>com.MyPluginPackage.MyPlugin.PerformanceView</precedingExtension>
   ...(extension data)...
</extension>
```

The `<precedingExtension>` elements in the example ensure that the `PerformanceView` extension is rendered first, followed by the `UtilityView` extension.

Filtering Extensions

In your extension definition, you can use filtering metadata to control when the extension appears in the vSphere Client GUI.

You can filter extensions based on the selected object type, on the value of any property associated with the selected object, or on the user's privilege level. You set the filter type and the specific filter values by using the appropriate XML elements inside the `<metadata>` element of your extension definition.

Filtering Based on Selected Object Type

You can filter your extension to appear only when the user selects one or more specific types of vSphere objects. You specify the types of objects for which the extension is valid by creating an `<objectType>` element in the `<metadata>` element in the extension definition. The extension appears only when the user selects an object whose type matches the value of the `<objectType>` element.

You can use any vSphere or custom object type name as the value for the `<objectType>` element. To specify multiple object types, include two or more object type names in the `<objectType>` element, separated by commas. You can also use the * symbol to specify all object types.

Extension Filtered by Entity Type

The following example filters the extension action to appear only when the user has selected a virtual machine object.

```
<extension id="com.vmware.samples.actions.vmActionSet">
    <extendedPoint>vise.actions.sets</extendedPoint>
    <object>
      <actions>
        <com.vmware.actionsfw.ActionSpec>
          <uid>com.vmware.samples.actions.myVmAction1</uid>
          <label>#{action1.label}</label>
          <command className="com.vmware.samples.actions.VmActionCommand"/>
        </com.vmware.actionsfw.ActionSpec>
      </actions>
    </object>
    <metadata>
        <!-- This filters the action to be visible only on VMs -->
      <objectType>VirtualMachine</objectType>
    </metadata>
</extension>
```

Filtering Based on the Value of a Property of the Selected Object

You can filter your extension to appear or not depending on the value of a property of the selected object. You must use the property value filter together with the object type filter.

You create the property value filter by describing one or more property value comparisons to be made on the selected object by using the <propertyConditions> element. You include the <propertyConditions> element in the <metadata> element in the extension definition. You can define a single comparison, or define multiple comparisons and conjoin those comparisons together.

In the <propertyConditions> element, you must describe a data object of type com.vmware.data.query.CompositeConstraint using MXML syntax. Using the CompositeConstraint data object, you specify the names of the object properties used in the filter, the desired value for each object property, and the comparison operator. You can also specify a conjoiner if your CompositeConstraint data object has multiple comparisons.

In the CompositeConstraint data object, you describe each property value comparison using the <nestedConstraints> element. The <nestedConstraints> element contains an array of data objects of type com.vmware.data.query.PropertyConstraint. Each PropertyConstraint data object represents one comparison between a given object property and a value you specify.

When you create a PropertyConstraint data object, you specify the object property to compare by using the <propertyName> element, the value to compare against using the <comparableValue> element, and the comparison operator using the <comparator> element.

The value of the <propertyName> element must match the name of the object property to compare. You can set the value of the <comparableValue> element depending on the type of property you are comparing, but the value must be a primitive type. You can use a string value, an integer value, or a Boolean value of true or false in the <comparableValue> element.

You use the <comparator> element to choose how the property is compared against the value you specify in the filter. You can use values of EQUALS, NOT_EQUALS, GREATER, SMALLER, CONTAINS, EQUALS_ANY_OF or CONTAINS_ANY_OF. If you use the CONTAINS operator, you must provide an array of values. If you use the operators EQUALS_ANY_OF or CONTAINS_ANY_OF, you must provide a string containing multiple values in the <comparableValue> element, each separated by a comma.

If your CompositeConstraint data object defines multiple comparisons, you can choose how those comparisons are conjoined by using the <conjoiner> element. You can use a value of AND or OR in the <conjoiner> element.

Example Property Value Filter

The following example filters an action extension only when the value of the isRootFolder property is true and the selected object contains child objects that are virtual machines.

```
<extension id="com.vmware.samples.actions.vmActionSet">
   <extendedPoint>vise.actions.sets</extendedPoint>
   <object>
      <actions>
         <com.vmware.actionsfw.ActionSpec>
            <uid>com.vmware.samples.actions.myVmAction1</uid>
            <label>#{action1.label}</label>
            <command className="com.vmware.samples.actions.VmActionCommand"/>
         </com.vmware.actionsfw.ActionSpec>
      </actions>
```

```
   </object>
   <metadata>
      <objectType>Folder</objectType>
      <propertyConditions>
         <com.vmware.data.query.CompositeConstraint>
            <nestedConstraints>
               <com.vmware.data.query.PropertyConstraint>
                  <propertyName>isRootFolder</propertyName>
                  <comparator>EQUALS</comparator>
                  <comparableValue>
                     <String>true</String>
                  </comparableValue>
               </com.vmware.data.query.PropertyConstraint>
               <com.vmware.data.query.PropertyConstraint>
                  <propertyName>childType</propertyName>
                  <comparator>CONTAINS</comparator>
                  <comparableValue>
                     <String>VirtualMachine</String>
                  </comparableValue>
               </com.vmware.data.query.PropertyConstraint>
            </nestedConstraints>
            <conjoiner>AND</conjoiner>
         </com.vmware.data.query.CompositeConstraint>
      </propertyConditions>
   </metadata>
</extension>
```

Filtering Based on User Privilege Level

You can filter your extension to appear only for users that have specific privileges. You can base your filter on global privilege settings in the vSphere Client, such as settings or licenses. For example, you can use a filter to make your extension available only to users that have global privileges to change settings.

You can also filter your extension based on privileges related to specific types of vSphere objects. For example, you can use a filter to make your extension available only to users who have privileges to create or delete datastore objects.

You specify the privilege for which the object is valid by creating a `<privilege>` element inside the `<metadata>` element in the extension definition. The extension appears only for users whose privileges include the value specified by the `<privilege>` element. You can specify multiple privilege values in the `<privilege>` element, separated by commas. If you specify multiple privileges, the user must have all specified privilege values for the extension to appear.

Limitations on Filtering Based on User Privilege Level

Filtering has no effect on nodespecs and shortcut items. To avoid impacting the load time of the object navigator, home screen, and main menu, the vSphere Client does not check permissions for nodespecs and shortcut items.

The vSphere Client shows plug-in views only if the user has the appropriate privilege. If not, the user interface displays a message informing the user of the lack of privilege.

Extension Filtered by User Privileges

The following example filters the extension to appear only when the user privileges include **Global.Licenses**.

```
<extension id="vsphere.core.hosts.sampleMonitorView">
  <extendedPoint>vsphere.core.hosts.monitorViews</extendedPoint>
  <object>
    <name>Sample Monitor View Title</name>
    <componentClass className="com.vmware.vsphere.client.sampleplugin.SampleObjectView"/>
  </object>
  <metadata>
    <privilege>Global.Licenses</privilege>
  </metadata>
</extension>
```

Using the vSphere Client JavaScript API

The vSphere Client provides several interfaces that your plug-in can use to communicate with the HTML5 platform. These JavaScript methods are documented here as if they have TypeScript signatures, but they run as pure JavaScript, and all complex types are plain old Javascript objects.

Each plug-in runs in an iframe that has the same origin as the vSphere Client.

> **NOTE**
> Do not access the internal JavaScript or CSS resources of the vSphere Client. Such access is unsupported and could cause your plug-in to fail in a future release of the vSphere Client.

vSphere Client JavaScript API: Modal Interface

The `modal` interface enables your plug-in to manage modal dialog windows.

NOTE
This SDK uses the Structured Clone Algorithm, which has limitations as described in the Mozilla Developer Network (MDN) web doc. Limitations affect Modal, Confirmation Modal, and Shared Modal APIs.

modal.AlertLevel

Description	Enum: • SUCCESS • INFO • WARNING • DANGER Clarity adds a standard icon, depending on the alert level. See https://v2.clarity.design/alerts.

Used by:
modal.ConfirmationModalConfig

modal.ButtonStyle

Description	Enum: • SUCCESS • INFO • WARNING • DANGER Clarity adds default CSS tags, depending on the action importance: • btn-success • btn-primary • btn-warning • btn-danger For examples of Clarity button styling, see the Solid Buttons illustrations at https://v2.clarity.design/buttons/#examples.

Used by:
modal.ModalButton

modal.ButtonType

Description	Enum: • PRIMARY • SECONDARY Clarity styles PRIMARY buttons with greater emphasis. See https://v2.clarity.design/buttons.

Used by:
modal.ModalButton

modal.close()

Signature	Closes the modal dialog box in the parent iframe.
Description	Optional data that will be passed to callback function specified by `onClosed` property at dialog open.
Parameter: `data`	Optional data that will be passed to callback function specified by `onClosed` property at dialog open.

NOTE
No data is passed to the callback function if the user clicks the dialog close box.

modal.ConfirmationModalConfig

Description	Specifies the properties of a confirmation modal dialog box.

Property	Type	Required?	Notes
`content`	`string`	yes	Confirmation message displayed in the dialog.
`buttons`	modal.ModalButton `[]`	yes	Buttons displayed in the dialog footer. (min 1, max 4)
`title`	`string`	no	Dialog main title. May not contain an icon. (default=`''`)
`AccessibilityTitle`	`string`	no	Used when dialog title is not displayed, such as in a wizard dialog.
`size`	modal.ModalSize	no	Width of the dialog box. (Default width is chosen by Clarity. Height is fixed.)
`closable`	`boolean`	no	Whether the dialog displays a close button. (default=`true`)
`onClosed`	`function(result:any): void`	no	Function runs when user closes the dialog. If modal.close() is called, its parameter is in turn passed to the `onClosed()` function.
`secondaryTitle`	`string`	no	Optional subtitle for the dialog.
`alertLevel`	modal.AlertLevel	no	Causes Clarity to add an icon to the dialog.

Used by:
modal.openConfirmationModal()

modal.DynamicModalConfig

Description	Specifies values for some properties of a modal dialog box.

Property	Type	Required?	Notes
title	string	no	Dialog title. May not contain an icon. (If not present, no change to dialog title.)
accessibilityTitle	string	no	Used when dialog title is not displayed, such as in a wizard dialog.
height	number	no	Dialog height, specified in pixels. (If not present, no change to dialog height.)

Used by:

modal.setOptions()

modal.getCustomData()

Signature	modal.getCustomData():any
Description	Returns the customData object provided when a modal dialog box was opened, or null if no customData object was provided.

modal.getSharedModalsMap()

Signature	modal.getSharedModalsMap(targetPluginId:string, callback:function):void
Description	Creates listings of modal dialog boxes shared by a specific plug-in and accessible to the calling plug-in. The callback function must have the following signature: function callback(sharedModalsMap:vsphere-client-javascript-api-modal-interface.dita#O2_wont_generate_ID_202208201257-en):void

modal.ModalButton

Description	Describes button properties.

Property	Type	Required?	Notes
label	string	yes	
type	modal.ButtonType	no	
style	modal.ButtonStyle	no	
callback	function(): void	no	

Used by:
modal.ConfirmationModalConfig

modal.ModalConfig

Description	Specifies the properties of a modal dialog box.

Property	Type	Required?	Notes
`url`	`string`	yes	Location of HTML content for the dialog.
`title`	`string`	no	Dialog title. May not contain an icon. (default=`''`)
`accessibilityTitle`	`string`	no	Used when dialog title is not displayed, such as in a wizard dialog.
`size`	`(width:number, height:number)`	no	Specify in pixels. Default is chosen by Clarity. See https://v2.clarity.design/modals.
`closable`	`boolean`	no	Whether the dialog displays a close button. (default=`true`)
`onClosed`	`function(result:any):void`	no	Function runs when the dialog closes. If `modal.close()` is called, its parameter is in turn passed to the `onClosed()` function. Function runs when the dialog closes. If modal.close() is called, its parameter is in turn passed to the `onClosed()` function.
`customData`	`any`	no	Data the calling module passes to the dialog.
`contextObjects`	`any[]`	no	IDs of relevant objects the calling module passes to the dialog.

Used by:
modal.open()

modal.ModalSize

Description	Width of a modal dialog box.

Property	Type	Required?	Notes
`width`	`number`	yes	Specified in pixels.

Used by:
modal.ConfirmationModalConfig

modal.open()

Signature	`modal.open(configObj:` modal.ModalConfig `):void`
Description	Opens a modal dialog box specified by the `configObj` parameter.
Parameter: `configObj`	Specifies the properties of this modal dialog box.

modal.openConfirmationModal()

Signature	`modal.openConfirmationModal(configObj:` modal.ConfirmationModalConfig `):void`
Description	Opens a lightweight modal dialog box designed to present information and confirmation buttons.
Parameter: `configObj`	Specifies the properties of this modal dialog box.

modal.openSharedModal()

Signature	`modal.openSharedModal(configObj:`vsphere-client-javascript-api-modal-interface.dita#GUID-7A73F8A7-B0DD-4D6D-B39E-ACC27CD521F5-en`):void`
Description	Opens a modal dialog box that is defined and shared by another plug-in.

modal.SharedModalConfig

Description	Specifies the properties for opening a shared modal dialog.

Property	Type	Required?	Notes
`pluginId`	`string`	yes	ID of the plug-in that defines the dialog.
`sharedModalId`	`string`	yes	ID of the shared modal dialog.
`vcGuid`	`string`	yes	The GUID of the vCenter Server instance where the plug-in that defines the dialog is registered.
`objectId`	`string`	no	
`onClosed`	`function(result:any):void`	no	Function runs when user closes the dialog. If modal.close() is called, its parameter is in turn passed to the `onClosed()` function.
`contextObjects`	`any[]`	no	IDs of relevant objects the calling module passes to the dialog.
`customData`	`any`	no	Additional data the calling module passes to the dialog.

Used by: vsphere-client-javascript-api-modal-interface.dita#SECTION_07BB474C-C87E-4295-B0DB-BEC83BB99D0B-en

modal.setOptions()

Signature	modal.setOptions(configObj:vsphere-client-javascript-api-modal-interface.dita#section_100902d7-9c2f-453e-9418-5bcbef1de3da-en):void
Description	Called by the parent view to modify some properties for a modal dialog box in the parent iframe.
Parameter: configObj	Specifies values for some dialog box properties.

modal.SharedModalsMap

Description	Information about modal dialog boxes shared by a specific plug-in instance registered with one or more vCenter Server instances.

Property	Type	Required?	Notes
sharedModalsInfoByVcGuid	[VcGuid:string]:vsphere-client-javascript-api-modal-interface.dita#SECTION_B7F375B5-C49A-4117-A881-AC80B268F1A0-en	yes	Map of modal dialog boxes shared by a plug-in instance, listed by each vCester Server instance where the plug-in is registered.
sharedModalsInfoByVcObjects		no	Reserved for future use.

Used by: vsphere-client-javascript-api-modal-interface.dita#SECTION_4BDA1279-509B-427C-BDED-74272C8B0EBC-en

modal.VcSharedModalsInfo

Description	Information about modal dialog boxes shared by a plug-in instance registered with a specific vCenter Server instance.

Property	Type	Required?	Notes
pluginVersion	string	yes	The plug-in version registered with the vCenter Server instance.
sharedModalsIds	string[]	yes	List of modal dialog boxes shared by the plug-in.

Used by: vsphere-client-javascript-api-modal-interface.dita#O2_wont_generate_ID_202208201257-en

vSphere Client JavaScript API: Application Interface

The app interface provides context object information and helps your plug-in navigate and control the vSphere Client user interface.

app.ApiEndpoints

Description	Holds a nested object that contains the parsed elements of the plug-in URL.

Property	Type	Required?	Notes
uiApiEndpoint	app.UiApiEndpoint	info only	

Used by: app.getApiEndpoints()

app.ClientInfo

Description	Documents type and version of vSphere Client.

Property	Type	Required?	Notes
type	string	info only	The vSphere Client type (must be HTML).
version	string	info only	The vSphere Client version string.

Used by: app.getClientInfo()

app.ClientViewNavigationOptions

Description	Specifies a destination view that is owned by the vSphere Client.

Property	Type	Required?	Notes
targetviewid	string	no	Navigation ID of the destination view. • administration.ceip • cluster.configure.settings.drs • cluster.configure.settings.ha • cluster.configure.settings.quickstart • cluster.monitor.allIsues • folder.configure.keyProviders • host.configure.networking.physicalAdapters • host.summary • namespace.storage.persistentVolumeClaims • network.summary • profilesAndPolicies.vmStoragePolicies • vm.monitor.performance.overview • vm.summary • folder.configure.settings.general • workload.platform.supervisor.summary
objectid	string	no	ID of any object associated with the view. (For a global view, this field is not required.).

Used by app.navigateToClientView()

app.formatDateTime()

Signature	app.formatDateTime(instant:number, options: app.PluginDateTimeFormatOptions):string
Description	Returns a formatted string that contains a human-readable representation of the instant parameter. The instant parameter holds the number of milliseconds since the start of 1 January 1970 UTC.

app.getApiEndpoints()

Signature	app.getApiEndpoints(): app.ApiEndpoints
Description	Returns the URLs of the vsphere-ui service API endpoints available to plug-ins. For an example, see the sample remote plug-ins available in the vSphere Client SDK.

app.getClientInfo()

Signature	app.getClientInfo(): app.ClientInfo
Description	Returns type and version info for the vSphere Client.

app.getClientLocale()

Signature	app.getClientLocale():string
Description	Returns the current locale of the vSphere Client.

app.getContextObjects()

Signature	app.getContextObjects():any[]
Description	Returns the IDs of the current context objects, depending on the view or dialog from which the method is called.
Return value:	**for global view** Returns empty array. Global views have no associated vSphere objects. **for vSphere object** Returns a context item for the associated vSphere object. **for dialog opened by modal.open()** If dialog opened by htmlClientSdk.modal.open(), returns value of configObj.contextObjects (or empty array, if contextObjects undefined) **for dialog opened by plugin.json actions** If dialog opened by action defined in plugin.json, returns an array of action targets. A context item is a JavaScript object containing a single property, id:string. This is the ID of the associated vSphere object.

app.getNavigationData()

Signature	app.getNavigationData():any
Description	Returns the custom data passed to the view by the app.navigateTo() . (If no custom data passed, returns null.)

app.getPluginBackendInfo()

NOTE
Applies to remote plug-ins only.

Signature	app.getPluginBackendInfo(callback:function):void
Description	Creates listings of service endpoints registered for the running plug-in instance, enabling plug-in front ends to use the vCenter Server ExtensionManager as a service registry. The callback function must have the following signature: Function callback(backendInfo: app.PluginBackendInfo):void If the method throws an exception, the callback parameter will be null.

app.getRemotePluginNavigationMap()

Signature	app.getRemotePluginNavigationMap(targetPluginId:string, callback:function):void
Description	Returns view IDs that belong to a plug-in specified by the targetPluginId parameter. If the targetPluginId is not the ID of the caller, the method returns IDs of only the public views for the target plug-in. Returns information for all vCenter Server instances where the target plug-in is registered and enabled. The callback function must have the following signature: function callback(navigationInfo: app.RemotePluginNavigationMap):void If the method throws an exception, the callback parameter will be null.

app.getSessionInfo()

Signature	app.getSessionInfo(callback:function):void
Description	Retrieves and processes information about the client's authentication session. The callback function must have the following signature: function callback(info: app.SessionInfo):void If the method throws an exception, the callback parameter will be null.

app.getTheme()

Signature	app.getTheme(): app.PluginTheme
Description	Retrieves information about the UI theme that is currently selected.

app.navigateTo()

Signature	app.navigateTo(configObj: app.NavigationOptions):void
Description	Navigates to a specified view, and optionally passes custom data to the view.
Parameter: configObj	Specifies the destination view and custom data.

app.navigateToClientView()

Signature	app.navigateTo(configObj: app.ClientViewNavigationOptions):void
Description	Navigates to a specified vSphere Client view.
Parameter: configObj	Specifies the destination view.

app.navigateToRemotePluginView()

Signature	app.navigateTo(configObj: app.RemotePluginViewNavigationOptions):void
Description	Navigates to a view implemented by another remote plug-in, or by another instance of the same plug-in deployed by a different vCenter Server instance. Optionally passes custom data to the view.
Parameter: configObj	Specifies the destination view and custom data.

NOTE
A best practice is to call app.getRemotePluginViewNavigationMap() before this procedure, to ensure that the navigation target is registered and enabled.

app.NavigationOptions

Description	Specifies a destination view and custom data for the view.

Property	Type	Required?	Notes
targetViewId	string	no	Navigation ID of the destination view. (For a remote plug-in, this property must identify a view created by the same plug-in.) Omit this to navigate to the last-used view of the object specified by objectId.
objectId	string	no	ID of any object associated with the view. Omit this to navigate to a global view.

Table continued on next page

Continued from previous page

Property	Type	Required?	Notes
customData	any	no	A custom data structure passed to the view.

Used by: app.navigateTo()

app.PluginBackendInfo

NOTE
Applies to remote plug-ins only.

Description	Contains two objects that list endpoint descriptors available to a given plug-in instance.

Property	Type	Required?	Notes
allPluginBackendServers	Array< app.PluginBackendServerInfo >	info only	A list of plug-in server endpoint descriptors registered for a given plug-in instance. The list includes all registrations within the same link group. The list is unordered and contains no duplicates.
backendServersPerVc	{[vcGuid:string]:Array< app.PluginBackendServerInfo >}	info only	A one-to-many mapping: vCenter Server GUID to a list of plug-in server endpoint descriptors registered with the vCenter Server instance on behalf of the running plug-in instance.

Used by: app.getPluginBackendInfo()

app.PluginBackendServerInfo

NOTE
Applies to remote plug-ins only.

Description	A descriptor for a plug-in server endpoint registered for a plug-in instance. Choose the server by the type value, then form a resource URL from the endpoint descriptor. To access a resource belonging to the server, prefix the proxiedBaseUrl value to the root-relative path of the resource on the plug-in server: /<proxiedBaseUrl>/<path to resource from server root>

Property	Type	Required?	Notes
proxiedBaseUrl	string	info only	The path component of the server root URL, as seen on the reverse proxy service port.
type	string	info only	The type of the server, as specified in its registration record (Extension data object) with a vCenter Server instance. Plug-ins can use this property to identify auxiliary

Table continued on next page

Continued from previous page

Property	Type	Required?	Notes
			servers that are part of the same plug-in instance. For more information, see Using Auxiliary Plug-in Servers.

Used by: app.PluginBackendInfo

app.PluginDateTimeFormatOptions

Description	Specifies which part of a timestamp to format.

Property	Type	Required?	Notes
format	string	no (default: DATE_AND_TIME)	Possible values: • DATE • TIME • DATE_AND_TIME

Used by: app.formatDateTime()

app.PluginTheme

Description	Indicates a choice of UI theme.

Property	Type	Required?	Notes
name	string	info only	Possible values: light or dark.

Used by: app.getTheme()

app.QueryParam

Description	Holds a single query parameter of a URL.

Property	Type	Required?	Notes
name	string	info only	Name of query parameter, as in ?name=value.
value	string	info only	Value of query parameter, as in ?name=value.

Used by: app.UiApiEndpoint

app.refreshPluginItemsState()

Signature	app.refreshPluginItemsState():void
Description	Triggers the refresh of all dynamic UI content hosted by the current plug-in. For example, the plug-in introduces a view that is dynamically filtered based on a property value of some object. If the property value is changed by the plug-in using the public vSphere APIs, then the plug-in must call this API to signal the UI to re-evaluate the state of the initiator's plug-in related items.

app.RemotePluginNavigationMap

Description	Lists remote plug-in destination views, belonging to a single vCenter Server instance, that are exposed by plug-ins using the `isPublic` property in the plug-in manifest. Use the map key as the value of the `vcGuid` property in the parameter to app.navigateToRemotePluginView() .

Property	Type	Required?	Notes
`navigationInfoByVcGuid`	`Map<string,` app.RemotePluginVcNavigationInfo `>`	info only	A one-to-many mapping of vCenter Server instances to view IDs that are served by a specified plug-in instance registered with each vCenter Server instance. Each map entry contains the GUID of a vCenter Server instance within the same link group, and a corresponding list of view descriptors.

Used by: app.getRemotePluginNavigationMap()

app.RemotePluginVcNavigationInfo

Description	Descriptor for views served by a given plug-in. Use a view ID as the value of the `targetViewId` property in the parameter to app.navigateToRemotePluginView()

Property	Type	Required?	Notes
`pluginVersion`	`string`	info only	The version of a plug-in instance registered with a given vCenter Server instance.
`viewIds`	`string[]`	info only	All the view IDs, served by a given plug-in instance, that are accessible to the current plug-in instance.

Used by: app.RemotePluginNavigationMap

app.RemotePluginViewNavigationOptions

Description	Specifies a destination view served by a different plug-in instance, and custom data for the view.

Property	Type	Required?	Notes
`pluginId`	`string`	yes	ID of the plug-in that owns the destination view.
`targetViewId`	`string`	yes	Navigation ID of the destination view. Use only values returned by `htmlClientSdk.app.getRemotePluginNavigationMap()` in the `viewsIds` array.
`vcGuid`	`string`	yes	The GUID of the vCenter Server instance that determines the context for the target view. If `objectId` is specified,

Table continued on next page

Continued from previous page

Property	Type	Required?	Notes
			the object must be managed by the specified vCenter Server instance.
`objectId`	`string`	no	ID of any object associated with the view. (For a global view, this field is not required.)
`customData`	`any`	no	A custom data structure passed to the view.

Used by: app.navigateToRemotePluginView()

`app.SessionInfo`

Description	Holds information about the current session of the vSphere Client.

Property	Type	Required?	Notes
`sessionToken`	`string`	info only	Identifier of the plug-in authentication session with vCenter Server.
`nodeId`	`string`	info only	Reserved for internal use only.

Used by: app.getSessionInfo()

`app.UiApiEndpoint`

Description	Holds the parsed elements of a plug-in URL.

Property	Type	Required?	Notes
`origin`	`string`	info only	<protocol>://<hostname><port>
`pathname`	`string`	info only	
`queryParams`	`Array<` app.QueryParam `>`	info only	<name>=<value>
`fullUrl`	`string`	info only	<origin>/<pathname>?<queryParams>

Used by: app.ApiEndpoints

vSphere Client JavaScript API: Event Interface

The `event` interface helps your plug-in with event management.

`event.onDateTimeFormatChanged()`

Signature	`event.onDateTimeFormatChanged(callback:function):void`
Description	Registers an event handler for changes to the datetime format preferences in the vSphere Client.
Parameter: `callback`	A reference to a function that responds to changes in datetime preferences. The callback function must have the following signature:

Table continued on next page

Continued from previous page

	`function callback():void`

`event.onGlobalRefresh()`

Signature	`event.onGlobalRefresh(callback:function):void`
Description	Registers a global refresh handler that the vSphere Client will call when the **Global Refresh** button is clicked.
Parameter: `callback`	A reference to a global refresh handler. The callback function must have the following signature: `function callback():void`

`event.onThemeChanged()`

Signature	`event.onThemeChanged(callback:function):void`
Description	Registers an event handler that the vSphere Client will call when the vSphere Client changes the current theme.
Parameter: `callback`	A reference to a theme change handler. The callback function must have the following signature: `function callback(theme:app.PluginTheme):void` The `theme` parameter identifies the new vSphere Client theme.

Example Using the `modal` API

This example shows some basic features of the `modal` interface of the Client API.

modal.html

```
<html>
  <head>
    <script src="http://code.jquery.com/jquery-latest.min.js"
            type="text/javascript"></script>
    <script src="/api/ui/htmlClientSdk.js"
            type="text/javascript"></script>
    <script type='text/javascript'>
      function handler(event)
      {
        var choice = $('input[name=heads_or_tails]:checked').val();
        htmlClientSdk.modal.setOptions({title: choice});
        settimeout(function(){htmlClientSdk.modal.close(choice);}, 3000);
```

```
      }
    </script>
  </head>
  <body>
    <form name='flip' onSubmit='return handler()'>
      <p><input type='radio' name='heads_or_tails' value='HEADS' />HEADS</p>
      <p><input type='radio' name='heads_or_tails' value='TAILS' />TAILS</p>
      <input type='submit' name='submit' value='Submit' />
    </form>
  </body>
</html>
```

modal.js

```
flipper = function(){
  # Select correct answer.
  correct = ['heads', 'tails'][2*Math.random()-1];

  # Create callback function.
  checker = function(choice){
    var correct = htmlClientSdk.modal.getCustomData();
    if (choice === correct) {
      alert('You chose wisely.');
    } else {
      alert('Sorry, you lose.');
    }}

  # Configure modal dialog.
  var config ={
    url: "example/dialog.html",
    title: 'Choose!',
    size: { width: 490, height: 240 },
    onClosed: checker,
    customData: correct}
```

```
  # Open modal dialog.
  htmlClientSdk.modal.open(config);
}

# Initialize Javascript API.
$(document).ready(htmlClientSdk.initialize(flipper));
```

Using Themes with vSphere Client Plug-ins

The vSphere Client SDK provides the means for a plug-in to integrate with the themes supported by the vSphere Client. Modifying a plug-in to support themes requires changes to the plug-in style sheets and front-end code to switch style sheets whenever the user changes the theme in the vSphere Client.

To integrate with the vSphere Client themes, a plug-in uses these methods of the JavaScript API:

- `app.getTheme()`
- `event.onThemeChanged(callback)`

To prepare your code for a theme change, you must identify and isolate theme-dependent styles and icons, then create variables with which to manage the style changes and overrides.You can merge your styles with the standard Clarity styles to improve performance. Finally, you can use examples in this book to load new styles in response to user theme changes.

The following procedures assume that the plug-in's front-end code is built using Angular and Clarity Design System. For other frameworks and build tools, the approach is similar but you will need to adapt the approach to suit the chosen tools. The examples in this guide are based on the HTML Plug-in Sample provided as part of the vSphere Client SDK.

Using Style Variables in Plug-In CSS

If a plug-in uses custom styles that depend on the theme colors, the plug-in style sheets (CSS or SASS or LESS) need to be parameterized. This enables the plug-in to adapt when the user switches themes in the vSphere Client user interface.

Ensure that the plug-in's Clarity version supports the `dark` theme. The first Clarity version to support the `dark` theme is 0.10.16.

In this procedure you copy any custom colors that depend on the current theme into variables in separate style sheets that are specific to the `light` or `dark` theme. You replace the colors in the original style sheets with instances of CSS variables. This is done to avoid style sheet duplication and to easily integrate theming with any custom Angular components the plug-in has defined. For more information about CSS variables, see https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_variables.

1. Identify any theme-dependent colors or styles in your plug-in.
2. Factor out theme-dependent colors or styles into two new style sheets as CSS variables.

   The SDK includes the following sample file at `html-client-sdk/samples/remote-plugin-sample/src/main/ui/src/styles-light.css`.

   ```
   :root {
           --border-color: rgb(204, 204, 204);
           --overlay-color: rgba(255, 255, 255, 0.2);
   ```

```
        --info-icon-color: darkblue;
    }
```

The SDK includes the following sample file at `html-client-sdk/samples/remote-plugin-sample/src/main/ui/src/styles-dark.css`.

```
:root {
        --border-color: rgb(72, 87, 100);
        --overlay-color: rgba(0, 0, 0, 0.2);
        --info-icon-color: darkblue;
    }
```

3. Replace the theme-dependent colors or styles in the original style sheets with variable references.

   The SDK includes the following code in the sample file at `html-client-sdk/samples/remote-plugin-sample/src/main/ui/src/app/views/list/list.component.scss`.

```
.splitter {
            flex: 0 0 auto;
            width: 1px;
            margin: 0 20px;
            background-color: var(--border-color);
          }
```

4. For Internet Explorer 11, which does not include support for CSS variables, include a polyfill library to provide support for CSS variables.

   The vSphere Client SDK includes a remote plug-in sample that uses `css-vars-ponyfill`. The following example is borrowed from `html-client-sdk/samples/remote-plugin-sample/src/main/ui/src/index.html`.

```
<script type="text/javascript" src="scripts/css-vars-ponyfill.js"></script>
```

- Configure theme-dependent icons in the plug-in manifest file.
- Use the modified input style sheets to build the output style sheets for your plug-in.

Building Output Style Sheets for vSphere Client Plug-Ins

After you isolate theme-dependent colors or styles as CSS variables, you can merge the resulting style sheets with the standard Clarity styles to produce a set of output style sheets for optimized performance.

Refactor the input style sheets for the plug-in so that they isolate theme-dependent colors and styles in separate style sheets as CSS variables.

Angular applications which use `webpack` and `angular-cli` place the style sheet declarations inline by default, when in development mode. Inline style declarations interfere with dynamic CSS loading. When you build the output style sheets, always configure the build to output and use external CSS:
To build external style sheets, add the `--extract-css` parameter to the `ng build` command. The vSphere Client SDK has examples of this usage in `Key definition for "package.json" not found in the DITA map.`.

You must disable any output file name hashing in the development and production builds. Otherwise the names of the style sheet files will change whenever the code changes, and the plug-in will not be able to load them.
To disable file name hashing when you build style sheets, use this syntax: `ng build --prod --output-hashing none`.

1. Create a base output style sheet that is independent of the themes.

   The base style sheet contains the Clarity icons style sheet and the base input style sheet for the plug-in, which uses CSS variables. The vSphere Client SDK builds this output style sheet by using Angular to compile the SCSS.

   The following example comes from the vSphere Client SDK file `Key definition for ".angular-cli.json" not found in the DITA map.`.

   ```
   "styles": [
     {
       "input": "../node_modules/clarity-icons/clarity-icons.min.css",
       "output": "styles",
       "lazy": true
     },
     {
       "input": "styles.css",
       "output": "styles",
       "lazy": true
     }
     ...
   ]
   ```

   This step combines the contents of `Key definition for "clarity-icons.min.css" not found in the DITA map.` and `Key definition for "styles.css" not found in the DITA map.` into `Key definition for "styles.bundle.css" not found in the DITA map.`.
2. Create an output style sheet file for the `light` theme.

   This style sheet includes the Clarity style sheet for the `light` theme and the plug-in style sheet for the `light` theme, which contains the CSS variable definitions for the `light` theme.

   The following example comes from the vSphere Client SDK file `Key definition for ".angular-cli.json" not found in the DITA map.`.

   ```
   "styles": [
     ...
     {
       "input": "../node_modules/clarity-ui/clarity-ui.min.css",
       "output": "theme-light",
       "lazy": true
   ```

```
  },
  {
    "input": "styles-light.css",
    "output": "theme-light",
    "lazy": true
  }
  ...
]
```

This step combines the contents of `Key definition for "clarity-ui.min.css" not found in the DITA map.` and `Key definition for "styles-light.css" not found in the DITA map.` into `Key definition for "theme-light.bundle.css" not found in the DITA map..`

3. Create an output style sheet file for the `dark` theme.

   This style sheet includes the Clarity style sheet for the `dark` theme and the plug-in style sheet for the `dark` theme, which contains the CSS variable definitions for the `dark` theme.

   The following example comes from the vSphere Client SDK file `Key definition for ".angular-cli.json" not found in the DITA map..`

```
"styles": [
  ...
  {
    "input": "../node_modules/clarity-ui/clarity-ui.min.css",
    "output": "theme-dark",
    "lazy": true
  },
  {
    "input": "styles-dark.css",
    "output": "theme-dark",
    "lazy": true
  }
  ...
]
```

   This step combines the contents of `Key definition for "clarity-ui.min.css" not found in the DITA map.` and `Key definition for "styles-dark.css" not found in the DITA map.` into `Key definition for "theme-dark.bundle.css" not found in the DITA map..`

Write front-end code to load style sheets that match the theme selected by the user.

Configuring and Loading Theme Style Sheets in vSphere Client Local Plug-Ins

After you compile the output style sheets for your plug-in user interface, you write front-end code to load the style sheets that cause your plug-in to conform to the style selected in the vSphere Client.

- Refactor the input style sheets for the plug-in so that they isolate theme-dependent colors and styles in separate style sheets as CSS variables.
- Build output style sheets into a base style sheet and a style sheet for each theme.

1. Load and configure polyfill libraries to provide CSS variable support in Internet Explorer 11.

   If you use `css-vars-ponyfill`, consider whether to configure options to create a MutationObserver and whether to remove CSS rulesets and declarations that do not reference a CSS custom property value. For more information about configuring `css-vars-ponyfill`, see https://github.com/jhildenbiddle/css-vars-ponyfill/tree/v1.17.1#optionswatch and https://github.com/jhildenbiddle/css-vars-ponyfill/tree/v1.17.1#optionsonlyvars.

   The vSphere Client SDK includes a remote plug-in sample that uses `css-vars-ponyfill`. The following example is borrowed from the file `html-client-sdk/samples/html-sample/html-sample-ui/src/index.html`.

   ```
   <script type="text/javascript" src="scripts/css-vars-ponyfill.js"></script>

   // Initialize CSS vars to configure polyfill.
   cssVars({
             watch: true,
             onlyVars: true
   });
   ```

   The following example is borrowed from the file `html-client-sdk/samples/html-sample/html-sample-ui/.angular-cli.json`.

   ```
     "assets": [
       "assets",
       {
         "glob":
         "css-vars-ponyfill.js",
         "input": "../node_modules/css-vars-ponyfill/dist/",
         "output": "scripts/
       },
      …
     ]
   ```

2. Load the base style sheet initially.

The following example is borrowed from `html-client-sdk/samples/html-sample/html-sample-ui/src/index.html`.

```
<link rel="stylesheet" type="text/css" href="styles.bundle.css">
```

3. Load the style sheet for the current theme initially and whenever the style changes.

   The following example is borrowed from `html-client-sdk/samples/html-sample/html-sample-ui/src/index.html`.

```
function() {
  function loadTheme(isFirstLoad, theme) {
    var themeName = theme.name;
    var supportedThemeNames = ["light", "dark"];
    if (supportedThemeNames.indexOf(theme.name) === -1) {
      themeName = supportedThemeNames[0];
    }

    var themeStyleSheetLinkId = "theme-stylesheet-link";
    var themeUrl = "theme-" + themeName + ".bundle.css";
    var styleSheetLinkElement;
    if (isFirstLoad) {
      styleSheetLinkElement = document.createElement("link");
      styleSheetLinkElement.setAttribute("rel", "stylesheet");
      styleSheetLinkElement.setAttribute("type", "text/css");
      styleSheetLinkElement.setAttribute("id", themeStyleSheetLinkId);
      styleSheetLinkElement.setAttribute("href", themeUrl);
      document.write(styleSheetLinkElement.outerHTML);
    } else {
      styleSheetLinkElement = document.getElementById(themeStyleSheetLinkId);
      styleSheetLinkElement.setAttribute("href", themeUrl);
    }

    document.documentElement.setAttribute("data-theme", themeName);
  }

  var htmlClientSdk = window.frameElement.htmlClientSdk;
```

```
    if (htmlClientSdk.app.getTheme && htmlClientSdk.event.onThemeChanged) {
      loadTheme(true, htmlClientSdk.app.getTheme());
      htmlClientSdk.event.onThemeChanged(loadTheme.bind(this, false));
    } else {
      loadTheme(true, { name: 'light' });
    }
  })();
```

Configuring Theme-Dependent Icons for vSphere Client Local Plug-ins

When you refactor style sheets for your plug-in to accommodate theme changes, you can also specify icons suited for alternative themes. You specify theme-dependent icons in the plugin icons style sheet.

Configuring theme-dependent icons is an optional step that can improve the user experience after a theme change.

1. Prepare alternative icons as needed to accommodate theme changes.
2. For each alternative icon, add a CSS theme selector to override the default icon after a theme change, similar to the image overrides in the following example.

CSS For `dark` Theme Icons

```
.com_vmware_samples_htmlsample-icon-plugin {
  background: none;
  background-image: url("../images/plugin.svg");
  background-size: contain;
  background-repeat: no-repeat;
  width: 18px;
  height: 18px;
  display: inline-block;
  vertical-align: text-bottom;
  margin: 1px 4px 0;
}

[data-plugin-theme="dark" .com_vmware_samples_htmlsample-icon-plugin {
  background-image: url("../images-dark/plugin.svg");
}
```

```
.com_vmware_samples_htmlsample-icon-list {
  background: none;
  background-image: url("../images/list.svg");
  background-size: contain;
  background-repeat: no-repeat;
  width: 16px;
  height: 16px;
  display: inline-block;
  vertical-align: text-bottom;
  margin: 1px 4px 0;
}

[data-plugin-theme="dark" .com_vmware_samples_htmlsample-icon-list {
  background-image: url("../images-dark/list.svg");
}
```

Write front-end code to load style sheets that match the theme selected by the user.

Developing HTML-Based User Interface Extensions

The vSphere Client is a Web browser-based application that provides an extensible plug-in architecture. The user interface layer contains every visual component of the application, including data views, portlets, and navigation controls.

You can add UI features by creating user interface extensions. A UI plug-in contains one or more extensions, which add UI elements to the vSphere Client user interface.

Overview

The vSphere Client provides an extensible plug-in architecture which you can use to create custom solutions for your environment. Use the vSphere Client SDK to develop HTML plug-ins that are compatible with the vSphere Client.

The vSphere Client SDK provides the following features:

- You can use the JavaScript libraries of your choice to develop the user interface components of your extensions.
- You can examine the sample HTML plug-in provided with the vSphere Client SDK. The sample demonstrates how you can add different extensions to the vSphere Client.
- You can create extensions to the Java service layer by using Java APIs provided in the SDK.

Accessing Data

Each global view extension is an independent HTML element that must communicate with the plug-in back-end service or the vSphere environment to retrieve data, or send commands, that the view requires. The vSphere Client SDK includes a JavaScript library that you can use when creating UI extensions. The JavaScript API provides access to user interface data such as a list of objects selected in the UI, locale of the vSphere Client, and navigation to a specified view.

The JavaScript code can use REST-based Ajax queries to retrieve data from the plug-in back-end service to retrieve data that the plug-in displays in the UI view. Alternatively, a user interface plug-in can redirect to another view by submitting HTML forms.

Global View Extensions

In the vSphere Client, you can create global view extensions to create custom solutions for the user interface.

A global view extension can have nearly any function, including aggregating data about different types of vSphere objects onto a single screen, or displaying data from sources outside the vSphere environment. A global view can be a simple single-level data view that uses the entire vSphere Client main workspace, or a complex nested view with its own internal navigation structure and organization. Creating a global view extension has a few restrictions:

- Global views are displayed in the vSphere Client main workspace, but exist outside of the virtual infrastructure hierarchy. The user selects a global view directly, either through a pointer in the object navigator or a shortcut on the vSphere Client home screen.
- To create a global view extension, you must define the extension by using the XML elements in the plug-in module manifest file, and create the HTML code that appears in the main workspace.

Use Cases

You can use global view extensions to create dashboard-style data views or console-style applications.

A dashboard aggregates data from different sources in the vSphere environment together in one unified data view. For example, you can create a dashboard that brings together status information about vSphere objects from different vCenter Servers.

Console-style applications are displayed in the vSphere Client main content area. For example, the vSphere Client Task Console and Event Console are console-style applications.

Creating Global View Extensions

You create global view extensions by using the `vise.global.views` extension point. To define a global view extension, you need only the view name and the content URL.

Since there is no context object for a global view extension, the global view document is opened with a request that contains only the `locale` parameter.

Extending the vCenter Object Workspace

The vSphere Client displays a standard object workspace for each type of vSphere object. Your plug-ins can extend the object navigator with new categories, such as **Settings**, **Custom Objects List**, and so on.

The object workspace is a collection of data views with a tabbed navigation structure and detailed views with table of contents entries. The workspace for a given vSphere object appears in the vSphere Client main workspace for a selected object from the virtual infrastructure.

Each vSphere object type has **Summary**, **Monitor**, **Configure**, and categorized object relations top-level tabs , and may contain additional detailed views within each tab. You can add extensions to create your own sub-views and detailed views within the **Monitor** and **Configure** tab views. You can also create new object workspaces with the default top-level tab, sub-tabs and detailed views structure.

Use Cases

You can either add a new data view to the existing object workspace for any type of vSphere object, or you can create an object workspace for a plug-in specific object navigator item or entry point.

In general, you add a data view extension to an existing object workspace to convey additional information about a vSphere object that is not included in the standard workspace of the object.

To implement a new workspace, you add an object navigator item that links to a global view extension. Within the global view extension, you have the freedom to implement any view structure you want, including a view with tabs and nested views.

When you create an object workspace, use XML extension templates as demonstrated in the sample code.

Extending an Existing Object Workspace

To add HTML extensions to the **Monitor** and **Configure** tabs in the vSphere Client, use the following generic extension points. These extension points generate a subordinate view inside that tab.

- `vsphere.core.${objectType}.monitorViews`
- `vsphere.core.${objectType}.manageViews`

For example, if you define an extension that extends the `vsphere.core.vm.manageViews` extension point, your extension appears as an entry in the table of contents under the **Configure** tab in the object workspace for virtual machine objects.

For a complete list of object workspace extension points available for the vSphere Client, see Object Workspace Extension Points.

Types of Data Views

A data view extension appears differently depending on the vSphere object that you specified with the extension point. Data views can appear in the object workspace having one of the following structures.

- Table of contents entry - If you define an extension to a top-level tab, such as **Monitor**, or **Configure**, a data view extension appears as an entry in the table of contents on the left in the object workspace.
- Portlet - If you define the portlet extension point, a data view extension appears as a portlet in the object workspace.

Configure and Monitor Views Extensions

You can extend a vSphere object view under the **Configure** and **Monitor** tabs by using the generic extension points `vsphere.core.${objectType}.monitorViews` and `vsphere.core.${objectType}.manageViews` to specify the generic HTML class that implements the new data view. You must also specify the URL to the HTML source of the data view.

Adding a Host Monitor View

Following is an example of how you can add an HTML view to the **Monitor** tab of host objects.

```
<extension id="com.vmware.samples.vspherewssdk.host.monitor">
   <extendedPoint>vsphere.core.host.monitorViews</extendedPoint>
   <object>
      <name>#{monitorHtml.label}</name>
      <contentSpec>

         <url>/ui/plugin-name/monitor-view.html</url>
      </contentSpec>
   </object>
```

```
</extension>
```

The value of the `<url>` property is a relative URL that starts with the Web context path of the plug-in, `/ui/`. You must set the same URL without the first slash as a value to the `Web-ContextPath` manifest header of the Web application `MANIFEST.MF` file.

To display content from another domain in the view, you can use HTTPS URLs. Note that the content is not loaded the first time that the user open the view, unless the domain certificate is already verified. You must not use HTTP URLs because the contemporary Web browsers are designed to block any insecure content that you try to display inside the secure vSphere Client domain.

The `monitor-view.html` document view is opened with a REST request that contains the following parameters:

- `objectId` - The context object ID of the view.
- `objectType` - The context object type.
- `locale` - The current locale of the Web browser.

Creating an Object Workspace for a Custom Object

If your vSphere environment contains a custom vSphere object, you can create the object workspace by using the provided extension templates.

For more information about the extension templates, see Extension Templates.

Creating Extensions to the Summary Tab

To create vSphere object views, you add portlets to the **Summary** tab.

You can add a portlet to the **Summary** tab of a vSphere object by using the `vsphere.core.{objectType}.summarySectionViews.html` HTML-specific extension point.

Adding Portlets to the Summary Tab

You create portlet views at the `<namespace>.summarySectionViews.html` extension point by using the generic `HtmlView` component class.

Adding Portlet Views to the Summary Tab

The following example creates a portlet in the **Summary** view of a host.

```
<extension id="com.vmware.samples.vspherewssdk.host.summary2">
   <extendedPoint>vsphere.core.host.summarySectionViews.html</extendedPoint>
   <object>
      <name>#{summaryView.title}</name>
      <contentSpec>
         <url>/ui/vspherewssdk/resources/host-summary.html</url>
         <dialogTitle>WSSDK Summary Sample</dialogTitle>
         <dialogSize>440,400</dialogSize>
      </contentSpec>
```

```
    </object>
  </extension>
```

Creating Data View Extensions

When you create data view extensions for the vSphere Client user interface layer, follow these general recommendations:

- You do not need to change the Data Adapter services running in the service layer.
- You can use the generic `DataAccessController` Java class provided with each generated plug-in project to handle HTTP JSON `GET` data requests.
- You must access data through the vSphere Client server and avoid calling directly your back end services or database.

Common Data Access Pattern

You can use the pattern demonstrated in the `html-sample` in the SDK to access data from the vCenter Server system from your plug-ins:

- The Ajax `GET` request created in your JavaScript code has the following format:
  /plugin_context_path`/rest/data/properties/`*objectId*`?properties=`*properties-list*
  , where *objectId* is the object ID of the currently selected vSphere object, and *properties-list* is the comma-separated list of properties that must be retrieved for that object.
- The `web.xml` deployment descriptor located in the `WEB-INF` folder of the UI bundle of your plug-in contains the `<servlet-mapping>` element that defines the `/rest/*` URL pattern for invoking the `springServlet` servlet.

```
<servlet-mapping>
   <servlet-name>springServlet</servlet-name>
   <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
```

- The `bundle-context.xml` file located in the `WEB-INF\spring` folder declares the `dataAccessController` bean for the `DataAccessController` class that is available in the Java service bundle of your plug-in.

```
<bean name="dataAccessController"
    class="com.vmware.samples.vspherewssdk.mvc.DataAccessController" />
```

- The `DataAccessController` class included in the Java service bundle of your plug-in has the `@RequestMapping` annotation set to process the HTTP JSON `GET` for the `/data` endpoint. The `getProperties()` generic method has the `@RequestMapping` annotation set to the `/properties/{objectId}` value to handle the Ajax `GET` requests created in your JavaScript code.

```
...
@Controller
@RequestMapping(value = "/data", method = RequestMethod.GET)
public class DataAccessController {

...

   @RequestMapping(value = "/properties/{objectId}")
```

```
@ResponseBody
public Map<String, Object> getProperties(
         @PathVariable("objectId") String encodedObjectId,
         @RequestParam(value = "properties", required = true) String properties)


 ...
```

- The `getProperties()` generic method uses the `QueryUtil` class to create a Data Service query for the requested list of vSphere object properties. The query results are returned to the Web browser as JSON data.

```
   ...
    Object ref = getDecodedReference(encodedObjectId);
    String objectId = _objectReferenceService.getUid(ref);


    String[] props = properties.split(",");
    PropertyValue[] pvs = QueryUtil.getProperties(_dataService, ref, props);
    Map<String, Object> propsMap = new HashMap<String, Object>();
    propsMap.put(OBJECT_ID, objectId);
    for (PropertyValue pv : pvs) {
       propsMap.put(pv.propertyName, pv.value);
    }
    return propsMap;
}
```

- The JavaScript code can display the data returned by the Ajax `GET` request as needed.

Creating Actions Extensions

You can extend the vSphere Client by adding actions. You can add actions to existing vSphere objects, or create actions associated with a new type of vSphere object.

In the vSphere Client, actions represent commands that the user can issue to manage, administer, or otherwise manipulate the objects in the vSphere environment. Each action in the vSphere Client is associated with one or more specific vSphere object types. For example, the user might perform an action to change the power state of a selected Virtual Machine object, or to cause a Host object to enter or exit maintenance mode.

When you add an action extension to the vSphere Client user interface layer, you must also extend the vSphere Client service layer with a Java service. The Java service is responsible for performing the action operation on the target vSphere object.

Use Cases

You can extend the vSphere Client by adding actions associated with an existing type of vSphere object, or with a new type of vSphere object. You might add actions to an existing object type if you have created a custom version of that vSphere object, such as a custom host.

In addition to creating the action extension in the user interface layer, you might need to add a Java service to the vSphere Client service layer. This Java service is used to perform the action operation on the target vSphere object.

Actions Framework Overview

The Actions Framework governs all available actions in the vSphere Client. All actions in the Actions Framework are organized into groups called action sets. When you create action extensions to the vSphere Client, you must define one or more action sets in the Actions Framework.

Each action in the Actions Framework is associated with one or more specific types of objects in the vSphere environment. Actions associated with virtual machines, for example, are available only when the user has selected a virtual machine object. Available actions are displayed in the actions drop-down menu at the top of the main workspace, or in a context menu when the user right-clicks on an object in the object navigator.

Action Controllers for HTML Extensions

In the plug-in module that contains your action extension, you must create a Java actions controller. The controller runs on the application server and acts as a dispatcher for commands. The HTML UI component sends commands to the actions controller using a REST API, and the controller routes the commands to services that implement the actions.

Defining an Action Set

An extension that adds one or more actions to the vSphere Client must define an action set. You add each action set extension to a specific extension point in the vSphere Client user interface layer, named `vise.actions.sets`.

Your extension definition must define an action set and the individual actions within that action set. An action set is a data object of type `com.vmware.actionsfw.ActionSetSpec`. The `ActionSetSpec` object contains an `<actions>` property, which is an array of action data objects. You specify each individual action in the set inside the `<actions>` property, using a separate data object for each.

You associate each action set extension with a particular type of vSphere object. A best practice is to use the vSphere Client extension filtering mechanism to ensure that the actions are only visible when the user selects the relevant type of vSphere object. See Filtering Extensions.

> **NOTE**
> If you omit the `<metadata>` element for extension filtering in your action set extension definition, your action is shown for all vSphere objects. Use the `<metadata>` element to ensure that your actions appear only for the correct type of vSphere custom objects.

Defining Individual Actions for HTML-Based Action Extensions

HTML-based extensions do not use the `<command>` property of the `ActionSpec` object. Instead they contain a `<delegate>` object.

The <delegate> object requires a `<className>` property and an `<object>` element that contains only an embedded `<root>` element. The `<className>` property specifies one of the following, depending on whether the action is modal or headless:

- `com.vmware.vsphere.client.HtmlPluginHeadlessAction` for a headless action
- `com.vmware.vsphere.client.HtmlPluginModalAction` for a modal (UI) action

The following table lists the properties that you can use in the `<root>` element.

Property	Type	Description
<actionURL>	string	Identifies the HTML resource to be displayed. The value can be an absolute HTTPS URL or a bundle context path. If the value is a bundle context path, the relative URL must end with the .html extension to enable session authentication. For absolute URLs, the framework does not use session authentication.
<dialogTitle>	string	Specifies the title of the dialog box. Add this property to the <root> element, or the action is treated as headless. Can be localized.
<dialogSize>	string	Indicates the width and height of the dialog box, in pixels, separated by commas.
<dialogIcon>	string	Specifies an optional icon resource for the dialog.
<closable>	boolean	Hides the top right close button for the dialog. The default value of this property is true.

There are two types of HTML-based action extensions. One type, known as a UI action, displays a modal dialog box for user input or confirmation before submitting a service request. The other type, known as a headless action, initiates a request to a service without additional user input. An extension definition for a UI action specifies the size and title of the dialog box, while a headless action definition omits the dialog box properties.

Invoking Headless HTML Actions

Your HTML-based action extension can invoke headless actions on its own initiative by specifying com.vmware.vsphere.client.HtmlPluginHeadlessAction as the delegate class name. of the action extension point.

- The value of the actionUrl parameter has the following form. /ui/html-sample/rest/vm-headless-ction
- The value of the jsonData parameter is a JSON map of parameters passed to the actions controller, or null if no parameters are needed.

HTML-Based Headless Action Extension Definition

The following example shows an extension definition for an HTML-based headless action extension.

```
<!-- Plugin Action set -->
<extension id="com.vmware.samples.htmlsample.vm.actionSet">
  <extendedPoint>vise.actions.sets</extendedPoint>
   <object>
```

```
      <actions>
...
         <!-- Plugin Headless Action -->
         <com.vmware.actionsfw.ActionSpec>
            <uid>com.vmware.samples.htmlsample.vm.headless.action</uid>
            <label>#{vmHeadlessActionLabel}</label>
            <delegate>
               <className>com.vmware.vsphere.client.HtmlPluginHeadlessAction</className>
               <object><root>
                  <actionUrl>/ui/html-sample/rest/vm-headless-action</actionUrl>
               </root></object>
            </delegate>
         </com.vmware.actionsfw.ActionSpec>
...
      </actions>
   </object>
   <metadata>
      <objectType>VirtualMachine</objectType>
   </metadata>
</extension>
```

When the headless action is invoked the JavaScript API makes a `POST` request to the actions controller on the Tomcat server, using the `actionUrl` property. The following parameters are added to the URL.

- `actionUid` - The `<uid>` of the `ActionSpec` object defined in the `plugin.xml` file
- `targets` - A comma-separated list of `objectIds`

By default, the targets parameter takes only one `objectId`. To specify more than one `objectId`, set the flag `acceptsMultipleTargets` to `true`.

In this example, the full URL takes the following form.

```
/vsphere-client/chassis/rest/actions.html?
actionUid=com.vmware.samples.chassis.deleteChassis&targets=objectId
```

UI Actions

You can implement a UI action that displays a modal dialog in response to a menu click or a toolbar button. You can implement also other types of pop-up dialogs that are specific to an object view or a global view.

When you define a UI action, you can supply an additional property for the `<delegate>` class to specify whether the dialog displays an **X** button to close the dialog. The default is to display an **X** button. To suppress the **X** button, add the `closable` property with a value of `false`.

HTML-Based UI Action Extension Definition

The following example shows an extension definition for an HTML-based UI action extension.

```
<!-- Plugin Action set -->
<extension id="com.vmware.samples.htmlsample.vm.actionSet">
  <extendedPoint>vise.actions.sets</extendedPoint>
   <object>
      <actions>
...
         <!-- Plugin UI Action -->
         <com.vmware.actionsfw.ActionSpec>
            <uid>com.vmware.samples.htmlsample.vm.modal.action</uid>
            <label>#{vmUiActionLabel}</label>
            <delegate>
               <className>com.vmware.vsphere.client.HtmlPluginModalAction</className>
               <object><root>
                  <actionUrl>/ui/html-sample/index.html?view=vm-modal-action</actionUrl>
                  <dialogTitle>#{vmActionModalTitle}</dialogTitle>
                  <dialogSize>600.250</dialogSize}
                  <closable>false</closable>
               </root></object>
            </delegate>
         </com.vmware.actionsfw.ActionSpec>
...
      </actions>
   </object>
   <metadata>
      <objectType>VirtualMachine</objectType>
   </metadata>
</extension>
```

When the action is invoked the platform opens a modal dialog containing the HTML document specified in the `actionUrl` property. The following table contains the parameters that are added to the URL.

- `locale` - The current locale that is used.

After the dialog form is submitted or the operation is canceled, the code calls `modal.close(data)`.

Handling Actions for HTML-Based Action Extensions

When you create an HTML-based action extension to the vSphere Client, you must create an actions controller class on the Tomcat server to respond to the REST API requests from the client code.

A best practice is to implement the controller class as a simple dispatcher that maps the action UIDs to Java services. You can invoke custom services or translate REST API requests to Data Manager requests.

Example Java Actions Controller Class

```
...
/**
  * Perform headless action on a VirtualMachine vSphere object.
  */
   @RequestMapping( value="vm-headless-action",
                    method = RequestMethod.POST)
   @ResponseBody
public void vmHeadlessAction() {
   // Implement your logic here to trigger an action on a virtual machine.
            }
...
```

Handling Locales

The default locale is the locale that is set by the Web browser of the user, or the English (United States) locale if the vSphere Client does not support the set locale.

In the vSphere Client locales are usually handled on the user interface layer. In some cases, the HTML plug-in must return text from the Java service layer such as the properties of a vSphere object adm error messages.

Handling Resources in the `plugin.xml` Manifest File

The localized resources for your plug-in are located in the `locales` directory of the WAR file. The `plugin.xml` manifest file contains the `<resources>` element that you must use to specify the location of plug-in resources such as images and localization data. The `defaultBundle` attribute of the `<plugin>` element specifies the name of the main `.properties` file of the plug-in and is added automatically by the Ant build scripts.

To instruct the vSphere Client to use the locale that your Web browser specifies at runtime, set `{locale}` as a value to the `locale` attribute of the `<resource>` element in the `plugin.xml` manifest file. You must avoid hard-coding a specific locale as a value to the `locale` attribute.

The `plugin.xml` manifest file contains the names of views, dialogs, action menus, icons, and other localizable objects. These strings and icons must be localized and not hard-coded in a particular language. If the string or icon is defined in the main properties file specified with the `defaultBundle` attribute, you must use the `#{RESOURCE_KEY}` syntax for the element and attribute values. If the string or icon is defined in a different `.properties` file, use the `#{BUNDLE_NAME:RESOURCE_KEY}` syntax for the element and attribute values.

Localizing Strings and Icons in the `plugin.xml` Manifest File

The following code snippet demonstrates how you can specify the values for strings and icons that must be localized in the vSphere Client depending on the settings of the Web browser. The main properties file of the plug-in is `locale/en_US/com_vmware_samples_chassisa.properties` which is reflected with the value of the `defaultBundle` attribute.

```
<plugin id="com.vmware.samples.chassisa"

   defaultBundle="com_vmware_samples_chassisa">


   <resources>

      <resource locale="{locale}">

         <module uri="locales/chassisa-{locale}.swf"/>

      </resource>

   </resources>

  ...

   <templateInstance id="com.vmware.samples.lists.allChassis">

      <templateId>vsphere.core.inventorylist.objectCollectionTemplate</templateId>

      <variable name="namespace" value="com.vmware.samples.chassisa_collection"/>

      <variable name="title" value="#{chassisLabel}"/>

      <variable name="icon" value="#{chassis}"/>

  ...
```

The English locales for the `chassisLabel` string and the `chassis` icon are defined in the `com_vmware_samples_chassisa.properties` file in the following way:

```
# ------- String properties --------


chassisLabel = ChassisA

summary.title = Chassis main info

...


# ------------- Images -------------


chassis = Embed("../../assets/images/chassis.png")

localizedImage.url = assets/images/localizedImage-en_US.png

...
```

Handling Resources in the HTML and JavaScript Code

You can retrieve the current client locale by using the app.getClientLocale() method from the JavaScript API. You can use the locale information to localize your plug-in UI with a framework of your choice. For an example of localizing a plug-in UI, see the HTML sample plug-in included with the SDK.

Handling Resources at the Service Layer

In some cases your plug-in might return strings from the service layer that must be displayed in the vSphere Client. For example, the service layer can return the properties of a vSphere object that must be displayed in a human-readable format, or an error message that comes from the back end. You must retrieve the current locale of the user and return the translated text for that locale in your Java code.

In case of error messages, your back end server might have the messages localized. In other cases, you can use the standard Java localization APIs and add `.properties` files inside your JAR files. These properties files are used to load the correct strings based on the locale.

Following is an example of how to use the `UserSession` class to access the locale of the current client session.

```
   // see the vsphere-wssdk-service sample for injecting  _userSessionService in your
class

   UserSession userSession = _userSessionService.getUserSession();

   String locale = userSession.locale;

   ...
```

Guidelines for Creating Plug-Ins Compatible with the vSphere Client

You can use the plug-in generation scripts provided with the vSphere Client SDK to create a plug-in that is compatible with both Web browser-based applications.

To develop an HTML plug-in, you must first create a plug-in project that has the required by the plug-in resources and directory structure. Use one of the generation scripts that are available in the `tools\Plugin generation scripts` folder under `html-client-sdk`.

After you create the HTML plug-in project, follow these guidelines to ensure that your plug-in is compatible with the vSphere Client:

- Use relative URLs to set the location to the resources inside your plug-in inside your HTML and JavaScript code. For example, you must avoid adding the `/ui` root path to the URLs.
- Use the `ui` root path only inside the `MANIFEST.MF` and `plugin.xml` files.
- Add Cascading Style Sheets (CSS) classes to the `plugin-icons.css` file for the icons that are displayed outside the views, such as Home screen shortcut icons, menu icons, and vSphere objects list icons. See Handling Icons Outside the HTML Views.
- When you add an extension to an existing object menu or a custom object menu, you must define a custom menu extension referencing the `vsphere.core.menus.solutionMenus` extension point in addition to the actions referencing the `vise.actions.sets` extension point. See Defining Menus and Sub-Menus.

  NOTE
  For the vSphere 8.0 release, local plug-ins must be upgraded as follows:

  1. Local plug-ins must be FIPS-compliant. See https://developer.vmware.com/docs/13385/preparing-local-plug-ins-for-fips-compliance
  2. Local plug-ins must bundle third-party libraries and installthem in a separate directory. See https://vdc-download.vmware.com/vmwb-repository/dcr-public/77dd6491-bb96-47f3-8c2e-a5a4655f078b/b916bc3c-0168-4487-a9e5-c89b50373c18/Local-Plugins-Library-Isolation.pdf

Using the Web Context Path in HTML Plug-Ins

Each HTML plug-in is a separate Web application that has a specific context path defined in the `MANIFEST.MF` file of the WAR bundle. The context path of your application specifies where the Web content is hosted and which requests must be handled by your application. For example, the Web context path for the HTML sample plug-in is defined in the manifest file as follows:

```
Web-ContextPath: ui/html-sample
```

The root path for resources and data requests for the vSphere Client starts with `ui`.

Handling Icons Outside the HTML Views

External icons are the icons displayed outside the HTML views and handled directly by the vSphere Client. Examples of such icons are the Home view shortcut icons, menu icons, and the vSphere object list icons. If you use the generation scripts to generate your HTML plug-in, the `plugin-icons.css` CSS file is added to the plug-in project. The example CSS file contains the definitions of two external icons.

To declare that your plug-in depends on external icons, in the `plugin.xml` manifest file add the `<dependency>` element inside the `<dependencies>` element. The following attributes of the `<dependency>` element contain information about the external icons:

- `type` - The resource type such as `css`.
- `uri` - The URI of the CSS file that contains the external icon declarations.

Following is an example of dependency declaration in the `plugin.xml` file:

```
<dependencies>

   <!-- Allow HTML Client to display icons in menus, shortcuts, lists -->

   <dependency type="css" uri="myplugin/assets/css/plugin-icons.css" />

</dependencies>
```

Defining Menus and Sub-Menus

When you add a custom vSphere object menu or extend an existing object menu, you must define each individual action and add a custom solution menu under the existing menu which might include sub-menus and separators. Use the `vise.actions.sets` extension point to define each action, and the `vsphere.core.menus.solutionMenus` extension point to add the custom solution menu.

The following example demonstrates how you can define custom actions for `VirtualMachine` objects and then add custom solution menus under the existing `VirtualMachine` menu.

```
<extension id="com.vmware.samples.vspherewssdk.vmActionSet">

   <extendedPoint>vise.actions.sets</extendedPoint>

   <object>

      <actions>

         <com.vmware.actionsfw.ActionSpec>

            <!-- UI action: show dialog -->

            <uid>com.vmware.samples.vspherewssdk.myVmAction1</uid>

            <label>#{action1.label}</label>
```

```
               <delegate>
<className>com.vmware.vsphere.client.htmlbridge.HtmlActionDelegate</className>
                  <object><root>
                     <!-- execute the action on client-side (html view in a modal dialog)
-->
                     <actionUrl>/vsphere-client/vspherewssdk/resources/vm-action-
dialog.html</actionUrl>
                     <dialogTitle>#{action1.label}</dialogTitle>
                     <dialogSize>500,250</dialogSize>
                  </root></object>
               </delegate>
            </com.vmware.actionsfw.ActionSpec>
      </object>
      <metadata>
         <!-- Filter this extension only for VirtualMachine objects -->
         <objectType>VirtualMachine</objectType>
      </metadata>
   </extension>
            ...

   <extension id="com.vmware.samples.vspherewssdk.vmMenu">
      <extendedPoint>vsphere.core.menus.solutionMenus</extendedPoint>
      <object>
         <!-- <label> is required here because it is an extension to an existing menu -->
         <label>#{solution.label}</label>
         <children>
            <Array>
               <com.vmware.actionsfw.ActionMenuItemSpec>
                  <!-- UI action example -->
                  <type>action</type>
                  <uid>com.vmware.samples.vspherewssdk.myVmAction1</uid>
               </com.vmware.actionsfw.ActionMenuItemSpec>
```

```
            ...
        </object>
        <metadata>
          <!-- Filter creates this extension only for VirtualMachine objects -->
          <objectType>VirtualMachine</objectType>
        </metadata>
    </extension>
```

Developing for the vSphere Client Service Layer

User interface elements in the vSphere Client interact with Java services that run in the application server. The Java services on the application server communicate with vCenter Server, ESXi hosts, and other data sources within and outside of the vSphere environment.

Developing Extensions to the Service Layer

The principal Java service included in the service layer is the Data Service. The Data Service provides data on objects that vCenter Server manages, using a query-based information model. Components in the and user interface layer, such as HTML data views, send queries to the Data Service for specific objects or attributes. The Data Service processes each query and returns responses.

When you create an extension in the user interface layer that requires data not provided by the Data Service, you must extend the service layer with new providers for the data. This chapter explains how to create Data Service extensions, how to create a custom Java service, how to access data using the vSphere Web Services API or the Data Services interface, and how to import services in a user interface module.

For more information about the relationships between the components in the different layers, see Understanding the vSphere Client Architecture.

Understanding the vSphere Client Data Service

The default Data Service provides a stateless, query-based interface to retrieve information about vSphere objects, as defined by the vSphere Web Client API.

The default Data Service interface can access data from vCenter Server. The Data Service accesses various services on vCenter Server, including the Inventory and Property Collector services.

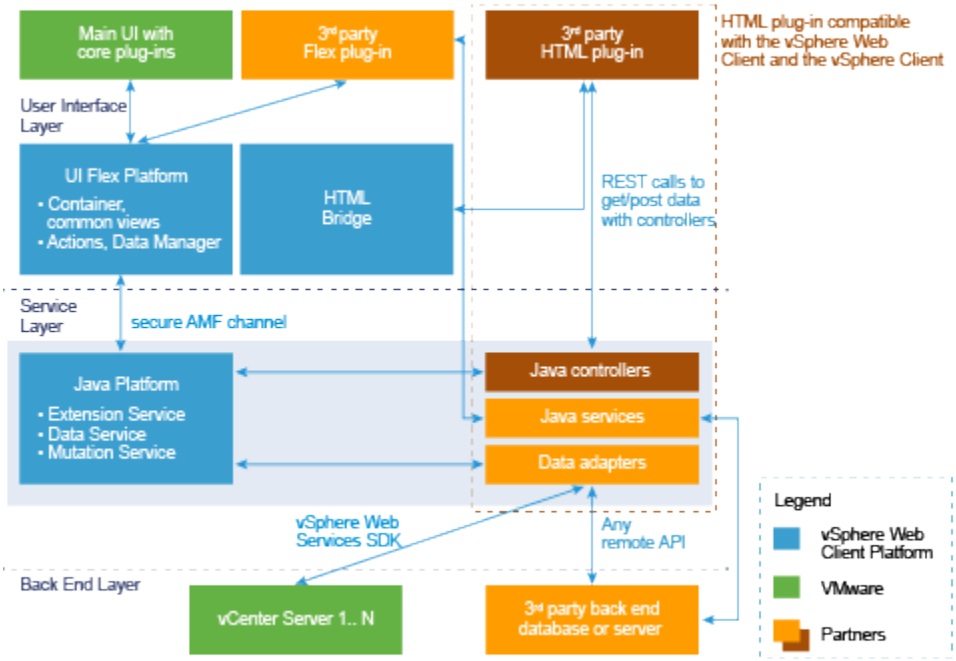
User interface components, such as Flex data views, act as Data Service clients. These clients retrieve information by creating Data Service queries. The Data Service processes each query and returns a set of result objects.

If your vSphere Web Client or vSphere Client extensions require data from a different source, either within vCenter Server or outside vCenter Server, you can extend the Data Service by creating a Data Service Adapter. A Data Service Adapter provides a way for you to use Data Service queries to retrieve a data from custom objects or to extend VMware managed objects.

Extending the Service Layer with Custom Components

The Web Client SDK provides several ways to extend the service layer. Each kind of extension is best suited for certain functions.

- To manage back-end operations in the service layer, you create custom Java plug-ins, which can be of two kinds:
  - Plug-ins that implement REST services that act on behalf of user interface plug-ins written in HTML.
  - Plug-ins that implement custom RPC interfaces on behalf of Flex proxy components.
- To retrieve data from vCenter Server or from external sources, you create custom data adapters in Java. Your data adapters can be of two kinds:
  - Property Provider Adapters can retrieve data from vSphere managed objects.
  - Data Provider Adapters can retrieve data from external sources as well as vSphere managed objects.



Custom Component Types

The following types of custom components belong to or communicate with the service layer.

HTML UI components

HTML components display the visual components of the vSphere Client interface. You can create custom HTML components to add new features to the user interface.

Flex UI components

Flex user interface components display the visual components of the vSphere Web Client interface. You can create custom Flex components to add new features to the user interface.

Data Service Adapters

Data Service Adapters implement query service interfaces designed by VMware for data requests from user interface components. Property Provider Adapters and Data Provider Adapters are the two kinds of Data Service Adapters.

Data Provider Adapters

Data Provider Adapters implement the `DataProviderAdapter` interface. They respond to requests for data from custom vSphere objects or from objects that are not managed by vSphere.

Property Provider Adapters

Property Provider Adapters implement the `PropertyProviderAdapter` interface. They respond to requests for properties of vSphere objects. Property Provider Adapters cannot provide properties for custom objects.

Custom Java services

Custom Java services provide operations on vSphere managed objects or external data sources. Custom services usually dispatch requests to vCenter Server or to external processes that perform extensive operations.

Interfaces to the Service Layer

Components communicating in the service layer can use the following types of interfaces:

Data Service

The Data Service is an interface accessible to the Data Access Manager or to controller services used by HTML UI components.

Data Access Manager

The Data Access Manager is a Flex library provided by VMware to simplify communications between Flex UI components and the service layer.

PropertyProviderAdapter

Property Provider Adapters implement the `PropertyProviderAdapter` interface of the Data Service. This interface is designed to provide properties of VMware managed objects.

DataProviderAdapter

Data Provider Adapters implement the `DataProviderAdapter` interface of the Data Service. This interface is designed to provide properties of custom objects.

Web Services API

The Web Services API is supported by vCenter Server and ESXi systems. It provides access to vSphere managed objects using an XML SOAP protocol.

Custom Service Interfaces

You can design your own service interfaces to use in custom Java services.

Communications with the Service Layer

The service layer contains several providers from VMware and you can extend it with custom providers that you create in Java. Custom providers collect and package data used either by custom user interface components or by existing user interface components.

HTML components in the vSphere Client user interface layer communicate with a controller service in the service layer by using REST APIs. The controller service can use the Data Service or the vSphere Web Services API to access data about vSphere objects, or extend the Data Service to access objects outside vSphere. The controller service can also use other custom or third-party services to access objects outside vSphere.

You can extend the Data Service to process queries for new data sources. The new data can come from other sources inside the vSphere environment, such as specific ESXi hosts, or from external data sources. When you extend the Data Service, your extensions in the user interface layer can communicate with new data sources by using the existing methods and libraries, such as the Data Access Manager.

You extend the Data Service by creating a Java service called a Data Service Adapter. A Data Service Adapter can either retrieve new properties for existing vSphere objects, or it can retrieve information from new custom objects. You must create different types of Data Service Adapters, depending on whether your environment adds new data to existing vSphere objects, or adds custom objects to the virtual infrastructure.

You can create custom Java services to work with your UI components. These custom Java services are typically used for performing action operations that make changes to the vSphere environment. Custom Java services are generally used as pass-throughs to back-end processes or external data sources.

> **NOTE**
> A best practice is to limit your Java service to dispatching requests from the vSphere Client, without passing on requests to other services. You can implement extensive or resource-intensive logic on your own external server.

Overview of Data Service Queries

Data Service is an API used to query data in the application server. You can use the Data Service either from user interface components or from providers in the service layer.

When To Use Data Service Queries

The Data Service is primarily intended for queries from user interface components. However, your service providers also have access to the Data Service.

You can initiate Data Service queries in the Java code of your service providers to fetch data from vCenter Server or from custom service providers. A best practice is to use the vSphere Web Services API to fetch data from vCenter Server, because it is more efficient than Data Services. However, you must use Data Services in the following cases:

- You need to join data from more than one vCenter Server.
- Your query includes properties that are available only from a custom provider.
- Your query includes data objects (complex properties) from vCenter Server, and the client is a UI component from VMware that understands data object encoding.

RequestSpec Data Structure in Data Service Queries

A Data Services client sends a request in the form of a `RequestSpec` object, which contains a list of `QuerySpec` objects.

QuerySpec Structure

The `name` field of a `QuerySpec` is optional. You can assign a name of your choosing, to help you identify the corresponding results. The `name` field is also useful to troubleshoot custom data providers.

A `QuerySpec` also contains a `ResourceSpec` and a `ResultSpec`.

ResourceSpec

The `ResourceSpec` specifies what properties and what objects are to be returned. It contains a list of `PropertySpec` objects and a tree of `Constraint` objects. The `PropertySpec` objects select resources and their properties, while the `Constraint` objects enable you to construct Boolean combinations of conditions to filter the set of resources from which properties are returned.

ResultSpec

The `ResultSpec`, which is optional, enables you to sort the results and to specify a chunk length and a starting index for the `ResultSet`.

The `OrderingCriteria` is a list of `OrderingPropertySpec`. Each list entry specifies the name of a sortable property and whether to sort the values in ascending or descending order.

`OrderingPropertySpec` is a subclass of `PropertySpec`. The subclass adds a `SortType` field.

> **NOTE**
> Sorting on custom properties can degrade performance in the client.

PropertySpec

A `PropertySpec` object is used to identify the properties to return in the `ResultSet`, or the properties used for sorting the results. In the latter usage, you can specify an optional sort order for the property by supplying an instance of `OrderingPropertySpec`, which is a subclass of `PropertySpec`. A `PropertySpec` is required in the `ResourceSpec`, but `OrderingPropertySpec` is optional.

A `PropertySpec` begins with a `type` field which contains the name of a resource type. This is typically the URI of a custom resource type, or the name of a managed object type in the case of a query that joins data across vCenter Servers. For example, to request properties of a `VirtualMachine` managed object, you must set the `type` field of a `PropertySpec` object to `"VirtualMachine"`.

A `PropertySpec` contains an array of strings identifying properties to be returned in the `ResultSet`. To identify nested properties, such as properties of nested data objects, use a period as delimiter. For example, the name of a virtual machine config file is a property of the `files` data object, which is a property of the `config` data object, which is a property of the `VirtualMachine` managed object, so you identify the chosen property with the string `"config.files.vmPathName"`.

To access properties of related resources or managed objects, such as the name of the host on which a virtual machine is currently running, use a `Constraint` object to do a join operation between the two managed object types.

> **NOTE**
> The `relation` field and the `ParameterSpec` array contained in the `propertySpec` object are reserved for internal use.

Constraint

`Constraint` objects enable you to specify arbitrary Boolean expressions that filter the results of your query. You can limit the results by placing conditions on property values and object identities. Your query must include a `Constraint` object.

`Constraint` is an abstract class with four subclasses. You can supply a simple constraint of object identity or property value by using an `ObjectIdentityConstraint` object or a `PropertyConstraint` object. You can use a `RelationalConstraint` object to join data across resource types.

You can use a `CompositeConstraint` wherever a `Constraint` object is allowed. A `CompositeConstraint` enables you to combine a list of other constraint objects, joined by a Boolean operator. You can nest a `CompositeConstraint` within another `CompositeConstraint`, which enables you to create arbitrarily complex Boolean expressions.

A query can contain the following types of constraints, each of which is a subclass of the base `Constraint` class.

- `ObjectIdentityConstraint` - Queries based on this constraint retrieve the properties of a known target object. For example, a query might retrieve the powered-on state of a given virtual machine. The object identifier can be a managed object type or any custom type that implements the `IResourceReference` interface. The identifier in this constraint includes the server GUID.
- `PropertyConstraint` - Queries based on this constraint retrieve all objects with a given property value. For example, a query might retrieve all virtual machine objects with a power state of on. This constraint accepts the property name and comparator as strings, and the property value as an `Object`. This constraint is not bound to a specific server, and can be used to retrieve results from all vCenter Servers known to the client.
- `RelationalConstraint` - Queries based on this constraint retrieve all objects that match the specified relationship with a given object. For example, a query might retrieve all virtual machine objects related to a given host object. The identifier in this constraint includes the server GUID.
- `CompositeConstraint` - Composite queries allow the combination of multiple constraints using the `and` or `or` operator, passed as a string. The combined subconstraints in `CompositeConstraint` are contained in an array of `Constraint` objects.

Each constraint operates relative to a resource type that you specify in its `targetType` field. For instance, if you want to query the names of all virtual machines running on a given host, one way is to create a `PropertyConstraint` that specifies a `targetType` of `"HostSystem"` and a value for the `name` property, then nest that `PropertyConstraint` in the `Constraint` field of a `RelationalConstraint` that specifies a `targetType` of `"VirtualMachine"` and a `relation` field of `"runtime.host"`.

ResultSet Data Structure in Data Service Queries

The response to a `RequestSpec` is a list of `ResultSet` objects. Each `ResultSet` corresponds to a `QuerySpec` object in the `RequestSpec`, with a one-to-one mapping.

The `queryName` field of a `ResultSet` is used to identify the `QuerySpec` that corresponds to the `ResultSet`. If you assigned a name to a query in the `RequestSpec`, the Response contains a `ResultSet` with a matching value in its `queryName` field. If you submitted a `QuerySpec` without a name, the corresponding `ResultSet` has an empty string in the `queryName` field. A best practice is to assign a unique name to each `QuerySpec` whenever you submit a request that contains more than one query.

When you process a `ResultSet`, first check the `error` field. If the `error` is non-empty, the query failed, and the `queryName` field has a valid value but other fields have indeterminate values. If the `error` is empty, the other fields are meaningful.

The `totalMatchedObjectCount` tells you the number of items the query can return. If the query did not specify a chunk size in the `maxResultCount` field, then `totalMatchedObjectCount` is the size of the `ResultSet.items` list. If the query did specify a chunk size, then the `items` list size is the minimum of `QuerySpec.maxResultCount` and `ResultSet.totalMatchedObjectCount - QuerySpec.offset`.

The data payload is `ResultSet.items`, which is a list of `ResourceItem` objects. Each `ResourceItem` object contains a single `resourceObject` field, which holds the identifier of the resource whose properties are returned in this `ResourceItem`. The `ResultItem.properties` field contains a list of name-value pairs for properties requested by the `QuerySpec`.

Extending the Data Service with a Data Service Adapter

You extend the Data Service by creating a Data Service Adapter to provide data to the components in your user interface extensions that require data that is not available through the Data Service.

A Data Service Adapter is a Java service that integrates with the Data Service, and gives the Data Service the ability to process and respond to Data Service queries for new object types or properties. Data Service Adapters can access data sources within vSphere, or outside data sources.

A Data Service Adapter must implement the same interface and information model as the Data Service. When you create a Data Service Adapter, it must handle Data Service queries and return information as a result set consisting of objects with associated properties.

Advantages of Providing a Data Service Adapter

Extending the Data Service by creating a Data Service Adapter has several advantages.

- The Data Service routes queries to the appropriate Data Service Adapters. This mechanism removes any distinction between data sources inside or outside of vSphere, and your extension components can access multiple data sources in a single call.
- The Flex components in your user interface extensions can use the Data Access Manager interface to access the new data. The Data Access Manager provides a consistent data access model throughout the component, easing maintenance and improving code consistency and re-use.
- Centralizing data access through the Data Service lets your extension components take advantage of services such as logging and error handling.

Designing a Data Service Adapter

To create a Data Service Adapter, you must create a Java service that implements one of the adapter interfaces published by the Data Service. The Data Service publishes interfaces for Property Provider Adapters and Data Provider Adapters. The type of Data Service Adapter you must create depends on the information you want to make available through the Data Service.

Property Provider Adapters

You create a Property Provider Adapter to allow the Data Service to access new properties for existing vSphere objects, such as virtual machines or hosts. For example, your vSphere environment might contain custom virtual machines or hosts that provide extra properties not normally available through the Data Service. You can create a Property Provider Adapter to extend the Data Service to fetch these additional properties.

Data Provider Adapters

You can use a Data Provider Adapter to extend the Data Service to fetch data that is not associated with an existing vSphere object. Typically, you create a Data Provider Adapter for one of the following purposes.

- To retrieve information about a new type of object that you have added to the vSphere environment
- To retrieve information from a source outside the vSphere environment

For example, you might create a Data Provider Adapter to handle queries for a new type of vSphere object called Chassis. You might also use a Data Provider Adapter to display data in the vSphere Web Client or the vSphere Client from an external Web source separate from vCenter Server.

Implementing an Adapter

To implement one of the adapter interfaces, your Java service must import the `com.vmware.vise.data.query` package.

After you create the adapter service, you must add the adapter service to the Tomcat server framework and register the adapter with the Data Service. You register an adapter by using the `DataServiceExtensionRegistry` service, typically within your adapter constructor method. See Registering a Property Provider Adapter and Registering a Data Provider Adapter.

The registration process declares what types of objects and properties the Data Service Adapter can provide. When the Data Service receives a query for one of the registered object or property types, the Data Service routes the query to the proper Data Service Adapter.

Processing Data Service Queries

Data Service queries are passed to your Data Service Adapter through the `com.vmware.data.query.RequestSpec` object parameter.

A `RequestSpec` object consists of an array of objects of type `com.vmware.data.query.QuerySpec`, each of which represents an individual query. Each `QuerySpec` object defines the query target, the query constraints, and the expected formatting for the query results.

Query Target

A query target is a resource type for which your `getData()` method must retrieve properties. A `QuerySpec` can specify a number of targets within its `ResourceSpec`, by including an array of objects of type `com.vmware.data.query.PropertySpec`. Each target type is represented as a string in the field `ResourceSpec.PropertySpec[x].type`.

Your `getData()` method can determine what information it must retrieve by using the values in the `PropertySpec` objects. If the target is a VMware managed object, the value of the string is the name of the managed object type. For custom objects, see Resolving a Custom Target Object.

Handling Constraints

Within the `QuerySpec` object, the query constraints are represented as an object of type com.`vmware.data.query.Constraint`. A query can specify the following types of constraints, each of which is a subclass of the base `Constraint` class.

- `ObjectIdentityConstraint` - Queries based on this constraint retrieve the properties of a known target object. For example, a query might retrieve the powered-on state of a given virtual machine. The object identifier can be a managed object type or any custom type that implements the `IResourceReference` interface. The identifier in this constraint includes the server GUID.
- `PropertyConstraint` - Queries based on this constraint retrieve all objects with a given property value. For example, a query might retrieve all virtual machine objects with a power state of on. This constraint accepts the property name and comparator as strings, and the property value as an `Object`. This constraint is not bound to a specific server, and can be used to retrieve results from all vCenter Servers known to the client.
- `RelationalConstraint` - Queries based on this constraint retrieve all objects that match the specified relationship with a given object. For example, a query might retrieve all virtual machine objects related to a given host object. The identifier in this constraint includes the server GUID.
- `CompositeConstraint` - Composite queries allow the combination of multiple constraints using the `and` or `or` operator, passed as a string. The combined subconstraints in `CompositeConstraint` are contained in an array of `Constraint` objects.

When processing constraints, a best practice is to read the entire set of constraints and then determine the most efficient processing order. For example, you can process relational constraints first to retrieve a smaller number of objects that meet any included property constraints.

Specifying Result Sets

In the `QuerySpec` object, the expected formatting for the query results are included in an object of type `com.vmware.data.query.ResultSpec`. The properties of the `ResultSpec` object specify a maximum number of results for the query to return, provide an offset into the returned results, and set ordering for the returned results. Your `getData()` method must use the values of the `ResultSpec` properties to format the information it has retrieved.

> **NOTE**
> When a Data Service query requests a vSphere data object as a whole, rather than its properties, the response contains the data object in an unsupported format that VMware user interface elements understand. If your provider needs to use the Data Service to request a data object on behalf of a client, your provider should copy the data object from its query results into the result set that your provider is building in response to the client, without doing any kind of processing on the data object portion of the results.

Property Provider Adapters

Queries to a Property Provider Adapter accept one or more specific vSphere objects, and return one or more properties for those objects. A Property Provider Adapter registers with the Data Service to advertise which types of properties it can return. When the Data Service receives a query for one of the registered property types, the Data Service routes the query to the appropriate Property Provider Adapter for processing.

> **NOTE**
> You may not register a provider for an existing VMware property or object type. For example, if your solution needs to identify a host by an alternate name, you may create an adapter to implement a property such as `alt_name`, but it may not modify the original `name` property.

`PropertyProviderAdapter` Interface

A Property Provider Adapter must implement the `PropertyProviderAdapter` interface of the `com.vmware.vise.data.query` package. The `PropertyProviderAdapter` interface publishes a single method named `getProperties()`. Your Property Provider Adapter service must provide an implementation of this method. The Data Service calls the `getProperties()` method of your adapter in response to an appropriate query for the properties your adapter is registered to provide.

The method implementation in your service must accept as its parameter an object of type `com.vmware.vise.data.query.PropertyRequestSpec`, and must return an object of type `com.vmware.vise.data.query.ResultSet`.

```
public ResultSet getProperties(PropertyRequestSpec propertyRequest)
```

Your service implementation of the `getProperties()` method can retrieve and format data in any way you choose. However, your implementation must return the results as a `ResultSet` object. You use the `PropertyRequestSpec` object to obtain the query list of target vSphere objects and desired properties. The `PropertyRequestSpec` object contains an objects array and a properties array, which respectively contain the target vSphere objects and requested properties.

For additional information on `ResultSet`, `PropertyRequestSpec`, and other features in the `com.vmware.vise.data.query` package, see the Java API reference included in the SDK.

Registering a Property Provider Adapter

You must register your Property Provider Adapter for the adapter to work with the Data Service. You register your Property Provider Adapter with the Data Service by using the `DataServiceExtensionRegistry` service. The

`DataServiceExtensionRegistry` service contains a method named `registerDataAdapter()` that you must call to register your Property Provider Adapter.

A best practice for registering your adapter is to pass `DataServiceExtensionRegistry` as a parameter to your Property Provider Adapter class constructor, and call `registerDataAdapter()` from that constructor.

Property Provider Adapter

The following example shows a Property Provider Adapter class. The class constructor method registers the adapter with the Data Service.

The class constructor method `MyAdapter()` constructs an array of property types that the adapter can supply to the Data Service in the array named `providerTypes`. The constructor then calls the Data Service Extension Registry method named `registerDataAdapter` to register the Property Provider Adapter with the Data Service. The Data Service calls the override method `getProperties()` when the Data Service receives a query for the kinds of properties that were specified at registration. The `getProperties()` method must retrieve the necessary properties, format them as a `ResultSet` object, and return that `ResultSet`.

```
package com.myAdapter.PropertyProvider;


import com.vmware.vise.data.query;

import com.vmware.vise.data.query.PropertyProviderAdapter;

import com.vmware.vise.data.query.ResultSet;

import com.vmware.vise.data.query.type;


public class MyAdapter implements PropertyProviderAdapter {


  public MyAdapter(DataServiceExtensionRegistry extensionRegistry) {

    TypeInfo vmTypeInfo = new TypeInfo();

    vmTypeInfo.type = "VirtualMachine";

    vmTypeInfo.properties = new String[] { "myVMdata" };

    TypeInfo[] providerTypes = new TypeInfo[] {vmTypeInfo};


    extensionRegistry.registerDataAdapter(this, providerTypes);

  }


  @Override

  public ResultSet getProperties(PropertyRequestSpec propertyRequest) {

    // Logic to retrieve properties and return as result set

    ...
```

```
   }
}
```

Data Provider Adapters

You can use a Data Provider Adapter to retrieve almost any data, including data agnostic to vSphere, provided that you can format it as a set of objects and related properties.

A Data Provider Adapter is responsible for all aspects of data retrieval, including parsing a query, computing the results of access operations, finding the matching objects or properties, and formatting results as responses compatible with the Data Service.

Typically, you use a Data Provider Adapter to retrieve data on custom objects that you added to your vSphere environment. The specific implementation of the Data Provider Adapter's data access depends on the data source for your custom object. Your Data Provider Adapter might query a database for configuration data, or retrieve operational data directly from a particular device.

> **NOTE**
> You may not register a provider for an existing VMware property or object type. For example, if your solution needs to identify a host by an alternate name, you may create an adapter to implement a property such as `alt_name`, but it may not modify the original `name` property.

When designing a Data Provider Adapter, consider the following constraints:

- You must be able to represent the external data by using the same object and property model as the Data Service.
- The Java service that you create to act as the Data Provider Adapter must perform all necessary data fetching operations from your remote data source.
- The service you create must process Data Service queries and return Data Service result sets.
- In general, you should not use a Data Provider Adapter to add properties to an existing resource. if you register a Data Provider Adapter to service a request for any properties of the resource, your provider must be able to provide all properties for the resource. A best practice is to use a Property Provider Adapter to add properties to an existing resource.

`DataProviderAdapter` Interface

A Data Provider Adapter must implement the `DataProviderAdapter` interface in the `com.vmware.vise.data.query` Java SDK package.

The `DataProviderAdapter` interface publishes a single method named `getData()`. Your Data Provider Adapter service must provide an implementation of this method. The Data Service calls the `getData()` method of your adapter in response to the queries your adapter is registered to process.

Your implementation of the `getData()` method must accept an object of type `com.vmware.vise.data.query.RequestSpec` as a parameter, and must return an object of type `com.vmware.vise.data.query.Response`.

```
public Response getData(RequestSpec request)
```

The `RequestSpec` object parameter to the `getData()` method contains an array of Data Service query objects. Each query contains a target object and one or more constraints that define the information that the client requests, as well as the expected format for results.

Your `getData()` method determines what information it must fetch by processing each Data Service query and handling the included constraints. The `getData()` method must then retrieve that information, through whatever means your data source provides, such as a database query or a remote device method.

Your `getData()` method must format the retrieved information as a specific result type for each query, and then return those results as an array, packaged in a `Response` object.

Resolving a Custom Target Object

A custom target object for a query is identified by a Uniform Resource Identifiers (URI) string, which is a unique identifier for a specific custom object type. In your Data Provider Adapter, you must resolve the URI for a query target object to the correct custom object type.

Implementing a Resource Type Resolver

A best practice is to use a Resource Type Resolver to resolve a URI to the correct custom object type. To use a Resource Type Resolver, you must create a Java class that implements the interface `com.vmware.vise.data.uri.ResourceTypeResolver`.

The class you create to implement `ResourceTypeResolver` must support the following methods.

- `String getResourceType(URI uri)` - The `getResourceType()` method must parse a URI and return a String containing the type of custom object to which the URI pertains. For example, for a URI that referred to a custom `Chassis` object, the `getResourceType()` method must return the String `samples:Chassis`.
- `String getServerGuid(URI uri)` - The `getServerGuid()` method must parse a URI and return a String containing the server global unique identifier for the URI target object. For example, for the URI string `urn:cr:samples:Chassis:server1/ch-2`, the `getServerGuid()` method must return the string `server1`.

Registering a Resource Type Resolver

To use your Resource Type Resolver, you must register the resolver with the Data Service. You typically register the Resource Type Resolver in your Data Provider Adapter class constructor by using the Resource Type Resolver Registry service, an OSGi service included within the service layer of the vSphere Web Client and vSphere Client. You must use the Spring framework to pass the Resource Type Resolver Registry OSGi service as an argument to your class constructor method. See Passing Arguments to Your Class Constructor.

Data Provider Adapter Example shows an example of how to register a Resource Type Resolver.

Registering a Data Provider Adapter

You must register your Data Provider Adapter for the adapter to work with the Data Service. You can register an adapter implicitly by declaring the Java service as an OSGi bundle, or you can register an adapter explicitly by using the Data Service Extension Registry service.

Registering Implicitly

You can register your Data Provider Adapter implicitly when you add the adapter to the application server framework. To use implicit registration, you must declare the Java service that implements your Data Provider Adapter as an OSGi bundle when you add the service to the application server framework. The vSphere Web Client and the vSphere Client detect new OSGi bundles as they are added and register the Data Provider Adapters with the Data Service. You must also annotate the adapter class with the object types that the adapter supports.

Declaring the Service as an OSGi Bundle

To declare the service as an OSGi bundle, you must define Java service of your adapter as a Java Bean in the `bundle-context.xml` file. You can find the `bundle-context.xml` file in the `src/main/resources/META-INF/spring` folder of your plug-in module.

To define the Java Bean, you must add the following XML element to the `bundle-context.xml` file.

```
<bean name="MyDataProviderImpl" class="com.example.MyDataProviderAdapter"> </bean>
```

The name attribute is an identifier that you choose for the Java Bean. You must set the value of the class attribute to the fully qualified class name of the Java class you have created that implements the `DataProviderAdapter` interface.

After you define your Data Provider Adapter as a Java Bean, you must modify the `bundle-context-osgi.xml` file to include the Java Bean as an OSGi service. The `bundle-context-osgi.xml` file is in the `src/main/resources/META-INF/spring` folder of your plug-in module.

You must add the following XML element to the `bundle-context-osgi.xml` file.

```
<osgi:service id="MyDataProvider" ref="MyDataProviderImpl"
              interface="com.vmware.vise.data.query.DataProviderAdapter" />
```

The `id` attribute is an identifier that you choose for the Data Provider Adapter. You must set the value of the `ref` attribute to the same value as the `name` attribute that you defined when declaring your Java Bean. The `interface` attribute must be set to the fully qualified class name of the `DataProviderAdapter` interface.

You must update the `src/main/resources/META-INF/MANIFEST.MF` file to reflect any Java packages from the SDK that your Data Provider Adapter imports. You add the imported packages to the `Import-Package` manifest header of the `MANIFEST.MF` file.

In Data Provider Adapter Example, the example Data Provider Adapter imports the packages `com.vmware.vise.data.uri` and `com.vmware.data.query`. The packages are listed by using the `Import-Package` OSGi manifest header in the `MANIFEST.MF` file.

```
Import-Package: org.apache.commons.logging,

com.vmware.vise.data,

com.vmware.vise.data.query,

com.vmware.vise.data.uri
```

Annotating the Adapter Class

You must annotate your Data Provider Adapter class with the object types for which the adapter processes queries. The vSphere Web Client and the vSphere Client use these annotations to route queries for the specific types to the correct adapters. You use the `@type` annotation to define the vSphere object type for which the adapter processes queries.

For example, if you have a custom object of type `WhatsIt`, you annotate the class in the following way.

```
@type("samples:WhatsIt") // declares the supported object types

public class MyAdapter implements DataProviderAdapter {

...
```

Passing Arguments to Your Class Constructor

Most Data Provider Adapters use other OSGi services that the SDK provides. These services include the base Data Service, the Resource Type Resolver Registry, and the vSphere Object Reference Service. You can pass these OSGi services to your Data Provider Adapter as arguments to the Data Provider Adapter class constructor method.

All Data Provider Adapters can include the Data Service. To include the Data Service as an argument to your Data Provider Adapter class constructor, you add the following element to the `bundle-context-osgi.xml` file of your service.

```
<osgi:reference id="dataService" interface="com.vmware.vise.data.query.DataService" />
```

NOTE
Making Data Service queries from within a Data Service provider can impact the performance of your provider. A best practice is to use the vSphere Web Services API to fetch data from vCenter Server, because it is more efficient than Data Services.

If your Data Provider Adapter handles queries for multiple custom object types, you must include the Resource Type Resolver Registry OSGi service and register a Resource Type Resolver. To include the Resource Type Resolver Registry OSGi service as an argument to your Data Provider Adapter class constructor, you add the following element to the `bundle-context-osgi.xml` file of your service.

```
<osgi:reference id="uriRefTypeAdapter"
interface="com.vmware.vise.data.uri.ResourceTypeResolverRegistry" />
```

If your Data Provider Adapter handles queries for built-in vSphere object types, such as Hosts or Virtual Machines, you can include the vSphere Object Reference Service. To pass the vSphere Object Reference Service as an argument to your Data Provider Adapter class constructor, you add the following element to the `bundle-context-osgi.xml` file of your service.

```
<osgi:reference id="vimObjectReferenceService"
interface="com.vmware.vise.vim.data.VimObjectReferenceService" />
```

Your Data Provider Adapter can use the User Session Service to get information about the current user session. To pass the User Session Service as an argument to your Data Provider Adapter class constructor, you add the following element to the `bundle-context-osgi.xml` file of your service.

```
<osgi: reference id="userSessionService"
interface="com.vmware.vise.usersession.UserSessionService" />
```

If you pass OSGi services to your Data Provider Adapter class constructor, you must include those constructor arguments when you declare your Data Provider Adapter as a Java Bean in the `bundle-context.xml file`. See Declaring the Service as an OSGi Bundle.

For each service your Data Provider Adapter includes, you must add a `<constructor-arg>` element to the Bean definition of your adapter. In each `<constructor-arg>` element, you set the `ref` attribute to the same value as the `id` attribute in the `<osgi:reference>` element in the `bundle-context-osgi.xml` file.

If your Data Provider Adapter uses the Data Service, vSphere Object Reference Service, Resource Type Resolver Registry, and User Session Service, the Bean definition might appear as follows.

```
<bean name="MyDataProviderImpl" class="com.example.MyDataProviderAdapter">

  <constructor-arg ref="dataService"/>

  <constructor-arg ref="uriRefTypeAdapter"/>

  <constructor-arg ref="vimObjectReferenceService"/>

  <constructor-arg ref="userSessionService"/>

</bean>
```

Registering Explicitly

You can register your Data Provider Adapter with the Data Service by using the `DataServiceExtensionRegistry` service. `DataServiceExtensionRegistry` contains a `registerDataAdapter()` method that you must call to register your Data Provider Adapter.

A common way to register your adapter is to pass `DataServiceExtensionRegistry` as a parameter to your Data Provider Adapter class constructor, and call `registerDataAdapter()` from within that constructor.

Data Provider Adapter Example

The following example presents an example of a Data Provider Adapter class that supports hypothetical `WhatsIt` objects. In the example, the class constructor method initializes the class member variables for the Data Service and registers a Resource Type Resolver. The example assumes that the Data Provider Adapter is registered implicitly by registering the service as an OSGi bundle. The Data Service and Resource Type Resolver Registry services are passed as arguments to the class constructor.

As a best practice, you can initialize the other services that your Data Provider Adapter requires in your Data Provider Adapter class constructor. These might include the Data Service, the Resource Type Resolver Registry if your adapter handles multiple custom object types, and the vSphere Object Reference Service if your adapter requires data from regular vSphere objects.

For more complete examples of Data Provider Adapters, see the sample extensions included in the SDK.

Example Data Provider Adapter Class

The `getData()` method is called by the Data Service when it receives a query for one of the objects or properties specified at registration. In the `getData()` method, your Data Provider Adapter must parse the query, compute the results, and return that result data as a `Response` object. For a more complete example, see the `ChassisDataAdapter` class in the SDK.

```
package com.MyAdapter.DataProvider;


import java.net.URI;


import com.vmware.vise.data.uri.ResourceTypeResolverRegistry;

import com.vmware.vise.data.query.DataProviderAdapter;

import com.vmware.vise.data.query.QuerySpec;

import com.vmware.vise.data.query.RequestSpec;

import com.vmware.vise.data.query.Response;

import com.vmware.vise.data.query.type;


@type("samples:WhatsIt") // type that the adapter supports

public class MyAdapter implements DataProviderAdapter {


  private final DataService _dataService;


  // Resource resolver, used to resolve the URIs of objects serviced by this adapter
```

```
  private static final ModelObjectUriResolver RESOURCE_RESOLVER = new
ModelObjectUriResolver();


  // constructor method
  public MyAdapter( DataService dataService,
                    ResourceTypeResolverRegistry typeResolverRegistry )
  {


    if ( dataService == null || typeResolverRegistry == null ) {
      throw new IllegalArgumentException("MyAdapter constructor arguments must be non-
null.");
    }
    _dataService = dataService;
    try {
      // Register the Resource Type resolver for multiple custom object types
      typeResolverRegistry.registerSchemeResolver( ModelObjectUriResolver.SCHEME,
                                                   RESOURCE_RESOLVER);
    } catch (UnsupportedOperationException e) {
      _logger.warn("ModelObjectUriResolver registration failed.", e);
    }
  }


  @Override
  // All query requests for the types supported by this adapter are routed here by the
vSphere
  // Web Client Data Service; this method is the starting point for processing
constraints,
  // discovering objects and properties, and returning results
  public Response getData(RequestSpec request) {
    QuerySpec[] querySpecs = request.querySpec;
    List<ResultSet> results = new ArrayList<ResultSet>(querySpecs.length);
    for (QuerySpec qs : querySpecs) {
      // Call your logic for query processing, constraint processing, object discovery:
      ResultSet rs = processQuery(qs);
```

```
      results.add(rs);
    }
    Response response = new Response();
    response.resultSet = results.toArray(new ResultSet[]{});
    return response;
  }
}
```

Creating a Custom Java Service

You can extend the Java service layer with your own Java services.

Typically, you create a Java service if your user interface extensions adds an action to the vSphere Web Client or the vSphere Client, where the Java service performs the action operation on the virtual infrastructure. You can also add a Java service to perform a complex calculation, retrieve data from an external source, or perform other miscellaneous tasks.

To add a Java service, you must provide a Java Archive (JAR) bundle. Inside the JAR bundle, you must add an XML configuration file that declares all of the Java objects that the service adds to the application server framework. The application server uses Spring as the application server framework.

Make Java Services Available to the UI Components in the vSphere Web Client and the vSphere Client

To make a custom Java service available to your extension components in the vSphere Web Client and the vSphere Client, complete the following tasks.

1. Create a Java interface for the service.
2. Create a Java class that implements the interface in Step 1.
3. Add the service to the application server framework.

   You must export and expose the service to the framework by adding it as a `bean` in the Spring configuration application server.

4. Import the service where your extension references it.
   - For Flex-based extensions, import the service into the user interface plug-in module that contains your Flex components.
   - For HTML-based extensions, import the service in the controller module that services your extension data requests.
5. Establish a communication between your service and the user interface layer.
   - For Flex-based extensions, use ActionScript to create a proxy class in your Flex component. The proxy class is used to communicate between the user interface plug-in module and the service.
   - HTML-based extensions access the service by using a REST API that communicates with the controller module on the application server.

Creating the Java Interface and Classes

To integrate with the application server Spring framework, the Java service you create must provide separate interface and implementation classes.

The following example shows a basic interface class and an implementation class.

```
package com.vmware.myService;


public interface MyService {
  String echo (String message);
}


public class MyServiceImpl implements MyService {
  public String echo (String message) {
    return message;
  }
}
```

Persisting Data from Your Plug-Ins to the vCenter Server Appliance and the vCenter Server System

You can store persistently small data files such as configuration changes on the vCenter Server Appliance and the vCenter Server system.

You can use the default data directory on the vCenter Server Appliance and the vCenter Server on Windows for storing small files. If the data you want to persist is complex or requires more storage space, you must use a separate back end server or database.

For more information, you can refer to the `GlobalServiceImpl.getGlobalViewDataFolder()` method from the Global View sample. The sample code demonstrates how you can use your Java services to create folders for storing the data persistently on the vCenter Server Appliance and vCenter Server instances.

> **NOTE**
> Make sure that the directories that you use for storing your data are accessible by the processes running on the Tomcat server.

Packaging and Exposing the Service

To make your Java service available for use with the vSphere Web Client and the vSphere Client, you must export the service and add it to the Spring configuration on the application server. Spring uses the OSGi model to share Java libraries.

Exporting the Service

You must locate the `/src/main/resources/META-INF/MANIFEST.MF` file in your service JAR bundle and ensure that the Java service package is exported. To export the package, the following line must appear in the `MANIFEST.MF` file:

```
Export-Package: com.vmware.myService
```

In the example line, `com.vmware.myService` is the name of the service package you created.

Adding the Service to the Spring Configuration

You add your service to the Spring configuration on the application server by creating a `<bean>` element in the Spring configuration file. In the JAR bundle, locate the `/src/main/resources/META-INF/spring/bundle-context.xml` file. The file contains a `<beans>` XML element containing services in the configuration. Add your service as a new `<bean>` as shown in the following example.

```
<bean name="myServiceImpl" class="com.vmware.myService.MyServiceImpl"/>
```

The name attribute is the name of your service implementation, and the class attribute contains the class you created that implements the service interface.

You must also expose the service interface as an OSGi bundle in the Spring framework. In the JAR bundle, locate the `/src/main/resources/META-INF/spring/bundle-context-osgi.xml` file. This file also contains a `<beans>` XML element. Add your service by using the following line.

```
<osgi:service id="myService" ref="myServiceImpl"
interface="com.vmware.myService.MyService"/>
```

The `id` attribute is the name of your service, the `ref` element specifies the service implementation you added to the `bundle-context.xml` file, and the `interface` element contains the class that defines the service interface.

Importing a Service in a User Interface Plug-In Module

To use a Java service you created and exposed in the service layer, a user interface plug-in module must import the service. You import the service by updating two metadata configuration files within your user interface plug-in module Web Archive (WAR) bundle.

In your user interface plug-in module WAR bundle, locate the `/war/src/main/webapp/META-INF/MANIFEST.MF` file and add the following lines.

```
Import-Package: com.vmware.myService
```

`com.vmware.myService` is the name of the service package you created.

Creating and Deploying Plug-In Packages

Each plug-in package contains both user interface plug-in modules and service plug-in modules, and manages the deployment of those modules. The vSphere Web Client and the vSphere Client extensibility frameworks can perform live hot deployment of the plug-in modules in a package.

Plug-In Package Overview

A plug-in package is a ZIP archive file that contains all of the plug-in modules in your solution along with a package manifest.

The package manifest describes deployment information for each plug-in module using XML metadata. The vSphere Client Extension Manager uses this metadata to install and deploy each plug-in module in the plug-in package.

To create a plug-in package, you must create a ZIP archive file with the following structure:

- At the root level, add a `plugin-package.xml` file to the root folder.
- At the root level, add a `plugins` folder.
- Inside the `plugins` folder, add one WAR files containing the plug-in UI modules.
- Inside the `plugins` folder, add zero or more JAR files, one for each Java service component created for your plug-in.

- Inside the `plugins` folder, add zero or more JAR files, one for each third party Java library used by your plug-in.

You can use any text or XML editor to create the `plugin-package.xml` file.

NOTE
Each WAR file or JAR file must contain an OSGi-compliant `META-INF/MANIFEST.MF` file that describes the bundle.

NOTE
For the vSphere 8.0 release, local plug-ins must be upgraded as follows:

1. Local plug-ins must be FIPS-compliant. See https://developer.vmware.com/docs/13385/preparing-local-plug-ins-for-fips-compliance
2. Local plug-ins must bundle third-party libraries and installthem in a separate directory. See https://vdc-download.vmware.com/vmwb-repository/dcr-public/77dd6491-bb96-47f3-8c2e-a5a4655f078b/b916bc3c-0168-4487-a9e5-c89b50373c18/Local-Plugins-Library-Isolation.pdf

XML Elements of the Plug-In Package Manifest File

The plug-in package manifest file specifies general information about the plug-in package, the deployment order for the plug-in modules in the package, and any dependencies for the plug-in package.

XML Elements in the Manifest File

The metadata in the manifest file follows a specific XML schema. The `<pluginPackage>` root element encapsulates the entire plug-in package manifest. The `<pluginPackage>` element can contain the `<dependencies>` element and the `<bundlesOrder>` element.

The following example shows an example of a `plugin-package.xml` manifest file. The source code that corresponds with this manifest file is available in the HTML sample in the SDK, at `html-client-sdk/vsphere-ui/plugin-packages/`.

```
<pluginPackage id = "com.MyCompany.myPackage"

version="1.0.0"

type="html"

name="My Plugin Name"

description="Demo package version 1"

vendor="My Company"

iconUri="assets/packageIcon.png">


<dependencies>

<pluginPackage id = "com.vmware.vsphere.client" version="6.7.0" />

<pluginPackage id = "com.vmware.vsphere.client.html" version="6.7.0" />

</dependencies>
```

```
<bundlesOrder>

<bundle id="com.mySolution.myUI" />
<bundle id="com.mySolution.myService" />
</bundlesOrder>

</pluginPackage>
```

<pluginPackage> Element

The <pluginPackage> element is the root element of any plug-in package manifest file. The following attributes of the <pluginPackage> contain information about the entire plug-in package.

Attribute Name	Description
id	The unique package identifier that you define. A best practice is to use namespace notation, such as com.myCompany.MyPluginID. Must match the vCenter Server extension key.
version	A dot-separated string containing the plug-in version number, such as 1.0.0. Must match the vCenter Server extension version.
type	Must be html.
description	A short description of the plug-in.
vendor	The name of the plug-in vendor.
iconUri	The URI of an icon to represent the package. The location is specified relative to the manifest file.

<dependencies> Element

The <dependencies> element defines any dependencies upon other packages. In the <dependencies> element, you specify each specific package dependency with a <pluginPackage> element. Each <pluginPackage> element in the <dependencies> element must have the following attributes.

Attribute Name	Description
id	The unique identifier of the package that your package depends on.
version	The version number of the package that your package depends on.
match	The version matching policy. Possible values are *equal*, *greaterThan*, *lessThan*, *greaterOrEqual*, or *lessOrEqual*. The match attribute is optional and defaults to *greaterOrEqual* if omitted.

IMPORTANT
If your vSphere Client plug-in depends on packages with specific versions and might not be compatible with later versions of these packages, make sure that you define correctly the dependencies by using the `match` parameter. Otherwise, your plug-in package will not work and might cause errors.
For example, you can use the following lines in the manifest file of your plug-in package to define the minimum and maximum supported versions of the vSphere Client:

```
...
<dependencies>
   <pluginPackage id="com.vmware.vsphere.client"
                  version="6.7.0" match="greaterOrEqual" />
   <pluginPackage id="com.vmware.vsphere.client.html"
                  version="6.7.0" match="greaterOrEqual" />
</dependencies>
...
```

If your plug-in package is only compatible with a specific version of the vSphere Client, you must use the *`equal`* value of the `match` attribute to specify the version. This ensures that when the vSphere Client is upgraded, your plug-in package will not be deployed, and will not cause any compatibility errors for your users.

`<bundlesOrder>` Element

The `<bundlesOrder>` element specifies the order in which locally hosted plug-in modules are deployed to the vSphere Client. A best practice is to deploy the service plug-in modules first, because the user interface plug-in modules might import those services.

You specify each plug-in module using a `<bundle>` element inside the `<bundlesOrder>` element. The `id` attribute of the `<bundle>` element contains the unique identifier of the plug-in module. The value of the `id` attribute must match the `Bundle-SymbolicName` specified in the plug-in module `MANIFEST.MF` file included in the WAR bundle.

NOTE
Plug-in modules in the package that are not explicitly specified in the `<bundlesOrder>` list are still deployed, but in an undefined order.

Deploying a Plug-In Package

You deploy a plug-in package to the vSphere Client by registering the package as an extension on vCenter Server. When you register your plug-in as an extension on vCenter Server, your plug-in becomes available to any vSphere Client that connects to your vCenter Server.

You must register your plug-in on every vCenter Server where you need to use it. When a vSphere Client connects to a vCenter Server where your plug-in is not registered, the plug-in is not visible to the vSphere Client.

When a vSphere Client establishes a user session to a vCenter Server instance, the vSphere Client application server queries vCenter Server for a list of all available plug-in packages that are registered as vCenter Server extensions. Plug-in packages that are not present on the vSphere Client application server are downloaded and deployed.

The vSphere Client application server can run only one version of each plug-in package. If a plug-in package is present on the application server, but has an older version number than the registered vCenter Server extension, the registered vCenter Server extension replaces the older plug-in package with the newer version.

When vCenter Server instances operate in linked mode, a user login can cause the vSphere Client to upgrade to the latest version of a plug-in. The vSphere Client queries the vCenter Server instance for a list of registered plug-ins, and the vCenter Server in turn queries all other instances in the link group to compare versions of registered plug-ins. The vCenter Server instance then upgrades all its registered plug-ins to the highest version registered within the link group, and presents that version in the user interface. This assures a consistent behavior for plug-ins, regardless of which vCenter Server instance the browser connects to.

> **NOTE**
> When a user views a managed object, the visibility of plug-in object extensions depends on whether the vCenter Server instance that owns the object has an applicable plug-in registered. If so, the vSphere Client always displays the latest version of the plug-in. If not, the vSphere Client does not display the plug-in at all.

Plug-In Caching

When the vSphere Client installs a plug-in, it downloads the plug-in manifest file and caches it. The cached copy is re-used whenever the vsphere-ui process restarts.

After a plug-in has been unregistered, the vsphere-ui service detects the change immediately. The cached copy of the plug-in manifest is deleted from the cache.

When a plug-in is upgraded, the vsphere-ui service detects the change as soon as the plug-in registration entry is updated with a new version number. At that time, the old plug-in version is undeployed, the cached copy of the manifest is removed, and the new plug-in version is deployed.

If plug-in resources are upgraded without changing the version number, such as when developing a plug-in, the vsphere-ui service does not detect the change. To deploy plug-in changes during development, use `pluginDeveloperMode` to enable the `Redeploy` button in the user interface. For more information, see Redeploying Plug-ins During Development.

Deploying a Plug-In Package From a Remote Server

The plug-in package ZIP file that represents a vSphere Client plug-in is typically hosted on a remote Web server. A vCenter Server extension can reference a remotely hosted plug-in package by specifying the Web sever URL in the vCenter Server extension definition. When you register a plug-in as an extension with a vCenter Server instance, the plug-in package ZIP file is downloaded from the remote URL.

The vSphere Client establishes a secure HTTPS connection with the remote Web server that hosts the plug-in packages. Starting with vSphere 6.0 Update 2, you can configure the TLS protocol versions for the vCenter Server Service, VMware vSphere Client Service, VMware Directory Service, Security Token Service and Syslog Collector Service. The TLS protocol versions that you configure for the vCenter Server service must be the same as the protocol versions for all other services.

For more information about supported TLS versions and configurations, see https://kb.vmware.com/s/article/2145796.

> **NOTE**
> Make sure that the Web server that hosts your vSphere Client plug-ins supports the same TLS protocol versions that are configured for the vSphere services. If this requirement is not met, the vSphere Client fails to download the extension plug-ins.

Register a Plug-In Package as a vCenter Server Extension

To register your plug-in package as an extension with vCenter Server, you must create an `Extension` data object and register this data object with the vCenter Server`ExtensionManager`.

You can create and register an `Extension` data object in the following ways:

- Use a utility application or script to create the `Extension` data object programmatically, and register that data object using the vSphere API. You can use the `ExtensionManager.registerExtension()` method to register the data object. For more information about the vCenter Server plug-in registration tool, see Register Your Local vSphere Client with the vCenter Server Instance.
- Use the Managed Object Browser (MOB) application for your vCenter Server system. For more information about how to use the MOB to register your extension, see the procedure bellow.

1. Create the `vim.Extension` data object in an XML file, and place that file in a file system available to the vSphere Client.
2. In a Web browser, navigate to the Managed Object Browser of your vCenter Server.

   `https://<vcenter_server_ip_address_or_fqdn>/mob/?moid=ExtensionManager`

3. Log in with your vCenter Server credentials.
4. On the `ManagedObjectReference:ExtensionManager` page, under Methods, click **RegisterExtension**.
5. On the `void RegisterExtension` page, in the text box inside the Value column, enter the XML data of your vSphere Client plug-in.
6. Click **Invoke Method** to register the plug-in as a vCenter Server extension.

For an example about how to define your `Extension` data object, see Creating the Extension Data Object.

Check whether your extension is registered successfully with the vCenter Server instance by using one of the following approaches:

- Log in to the vSphere Client, go to **Administration**, and under **Solutions**, select **Client Plug-Ins**.
- Log out and log in again to the vSphere Client. The vSphere Client checks for new plug-ins for each new user session.

> **NOTE**
> If you try to upgrade an existing plug-in with a new version and you do not follow the best practices and recommendations for developing vSphere Client plug-ins, you might need to restart the vSphere Client service to see your plug-in. This additional step is required in the following two cases:
> - The new version of your plug-in has a different plug-in ID.
> - The `plugin-package.xml` manifest file and the vCenter Server extension data object have different plug-in IDs or versions specified.

For more information about verifying the deployment of your plug-in package, see Verifying Your Plug-In Package Deployment.

Creating the vCenter Server`Extension` Data Object

Regardless of the registration method you choose, you must set the properties of the `Extension` data object.

You use the following properties to define the `Extension` data object.

Property Name	Description
`<key>`	The plug-in package ID that you defined in your plug-in package manifest file, `plugin-package.xml` file.
`<client>`	This property must contain one `ExtensionClientInfo` data object, with the following properties.

Table continued on next page

Continued from previous page

<table>
<tr><th>Property Name</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><th>Property Name</th><th>Description</th></tr>
<tr><td><version></td><td>The dot-separated version number of the plug-in package that is defined in plugin-package.xml.</td></tr>
<tr><td><type></td><td>Must be set to vsphere-client-serenity.</td></tr>
<tr><td><url></td><td>The location of the plug-in package ZIP file that is accessible on a Web server.</td></tr>
<tr><td><server></td><td>Optional. If the URL uses HTTPS, you must define a <server> property in your extension data object. The <server> property must contain the SHA1 thumbprint for the server where your plug-in package ZIP file is stored. For information about the <server> property, see the following example.</td></tr>
</table>
</td></tr>
</table>

Example `vim.Extension` XML Definition

The following example shows an example `Extension` object defined in an XML file.

```
<extension>
   <description>
      <label>My plugin</label>
      <summary>My first vSphere Client plugin</summary>
   </description>
   <key>com.mycompany.myPlugin.MyPlugin</key>
   <company>MyCompany</company>
   <version>1.0.0</version>
   <client>
      <version>1.0.0</version>
      <description>
         <label>My plugin</label>
```

```
         <summary>My first vSphere Client plugin</summary>
      </description>
      <company>MyCompany</company>
      <type>vsphere-client-serenity</type>
      <url>http://a-web-server-path/mypluginPackage.zip</url>
   </client>
</extension>
```

Using a Secure URL for the Plug-In Location

A best practice is to use a secure URL (HTTPS) for your plug-in package ZIP file location. If you use an HTTPS URL, you must include a `<server>` property in your `vim.Extension` data object. The `<server>` property contains the SHA1 thumbprint for the server that corresponds to the URL.

The following example shows an example `<server>` property.

```
<extension>
...
    <server>
        <url>https://myhost/helloworld-plugin.zip</url>
        <description>
            <label>Helloworld</label>
            <summary>Helloworld sample plugin</summary>
        </description>
        <company>MyCompany</company>
        <!-- SHA1 Thumbprint of the server hosting the .zip file -->
        <serverThumbprint>
            3D:E7:9A:85:01:A9:76:DD:AC:5D:83:1C:0E:E0:3C:F6:E6:2F:A9:97
        </serverThumbprint>
        <type>HTTPS</type>
        <adminEmail>your-email</adminEmail>
    </server>
</extension>
```

Verifying Your Plug-In Package Deployment

Once you register your plug-in package extension, the plug-in is downloaded and deployed on vSphere Client. You can verify that the deployment procedure is successful by using the log files of the Tomcat server, which are available at `/var/log/vmware/vsphere-ui`.

You can verify that your plug-in package is deployed correctly by searching the log file of the vSphere Client application server for your plug-in package ID. If the package is deployed correctly, the plug-in package ID is included in a message about a successful package deployment.

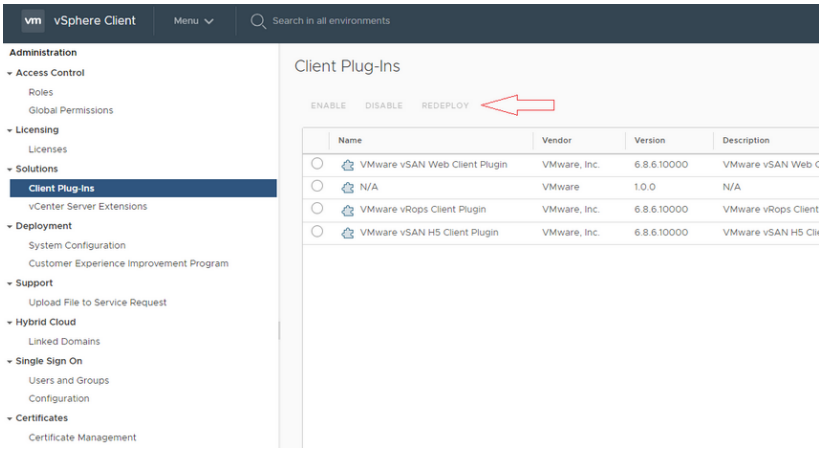
On startup, the vSphere Client caches the downloaded plug-in package in `/etc/vmware/vsphere-ui/vc-packages/vsphere-client-serenity/*` .

> **NOTE**
> If you want to update the content of a plug-in package, you must register the plug-in with a new version or remove the plug-in package from the cache.

Redeploying Plug-ins During Development

When you are developing and testing a plug-in, you can trigger redeployment frequently by using the `Redeploy` button, without the need to make changes to the extension registration record. After the `Redeploy` button is enabled, it is visible in the **Client Plug-Ins** window, where it applies to any selected plug-in. The button causes vCenter Server to undeploy the selected plug-in, and then deploy it again based on the same registration record.

Figure 162: Redeploy feature in Plugin Developer Mode



To enable the `Redeploy` button, set `pluginDeveloperMode` by appending the query parameter `pluginDeveloperMode=true` to any URL loaded into the browser window. For example, the URL could be the following:

`https://`*`vcenter-server-fqdn`*`/ui/?pluginDeveloperMode=true`

When you set `pluginDeveloperMode`, it remains in effect until you browse to a URL that contains the query parameter `pluginDeveloperMode=false`, or until you refresh the browser window while the browser addres field does not include `pluginDeveloperMode=true`.

Unregister a Plug-In Package

You can unregister a plug-in package that you previously registered with a vCenter Server instance.

You can unregister the extension in the following ways:
- Use the vSphere API and invoke the `unregisterExtension()` method of the `ExtensionManager` managed object to unregister your extension programmatically.
- Use the vCenter Managed Object Browser (MOB) interface in your Web browser to delete the extension manually. For more information about how to use the MOB to unregister your extension, see the procedure below.

1. In a Web browser, navigate to the Managed Object Browser of your vCenter Server.

   `https://<vcenter_server_ip_address_or_fqdn>/mob/?moid=ExtensionManager`

2. Log in with your vCenter Server credentials.
3. On the `ManagedObjectReference:ExtensionManager` page, under Methods, click **UnregisterExtension**
4. On the `void UnregisterExtension` page, in the text box inside the Value column, enter the value for the `key` property of the `Extension` data object of your vSphere Client extension.
5. Click **Invoke Method** to unregister the extension.

Unregistering a plug-in package on vCenter Server causes the vSphere Client to delete the plug-in from the environment at the next restart of the vSphere Client.

> **NOTE**
> In the current release of vSphere, any Java services you added are still active after you unregister a plug-in package, and the plug-in might still appear in the vSphere Client Plug-In Management view. This behavior is a known issue, and a workaround is to restart the Tomcat server.

Best Practices for Developing Extensions for the vSphere Client

You can improve your extension solutions by understanding the process of extending the user interface layer and service layer of the vSphere Client, and packaging and deploying your extension solutions. Follow best practices to ensure optimal performance and scalability, and to improve the security of your vSphere Client extensions.

Best Practices for Creating Plug-In Packages

To meet the requirements of your virtual environment, you must extend the capabilities of the vSphere Client by creating plug-in modules. Depending on your extension solution, you can extend the user interface layer and the service layer of the vSphere Client.

Incorrect structure of the plug-in package leads to deployment errors. To avoid these errors, consider the following best practices when creating your plug-in packages.

- Use the generation tools provided with the vSphere Client SDK to develop your vSphere Client extensions and create plug-in packages.
- Verify that the structure of the plug-in package is as follows:
  - `plugin-package.xml` - The file describes general information about the plug-in package, the deployment order of the plug-in modules, and any dependencies upon other plug-in packages.
  - `plugins` folder - The folder contains one or more JAR and WAR files that represent the user interface and Java services components. Limit the number of third-party libraries that you add to this folder.
- To avoid installation errors, make sure that all third-party libraries that you use are added inside the JAR and WAR files of the plug-in package and not inside the `plugins` folder. If you add third-party libraries to the `plugins` folder, the bundles must be OSGi-compliant. Because the vSphere Client resides on the Tomcat Web Server, which is based on the SpringSource dm Server and is built on top of the Equinox OSGi framework, third-party libraries must

be packaged as OSGi bundles. OSGi bundles must include an OSGi manifest file that contains correct and thorough OSGi metadata.

- To avoid deployment errors generated by the Tomcat server, make sure that you do not include third-party libraries that are already available on the server. You can navigate to the `html-client-sdk/server/webapps/h5-bridge-webapp.war/WEB-INF/eclipse/plugins` directory to view the available libraries.
- If your plug-in package contains both user interface and Java service components, place the Java service components before the user interface components in the plug-in package manifest file. Use the `<bundlesOrder>` element to specify the order in which the bundles are deployed to the vSphere Client.
- For best performance, when designing your vSphere Client extension, limit the number of files included in the `plugins` folder of your plug-in package. Ideally, your plug-in package must contain only one WAR file, which contains the user interface plug-in modules, and one JAR file, which contains the Java service plug-in modules. Fragmenting your code into many bundles might significantly increase the deployment time and memory consumption.
- To avoid compatibility issues in case your plug-in package depends on other plug-in packages with specific versions, make sure that you define correctly the plug-in dependencies by using the `match` parameter of the `dependencies` element in your `plugin-package.xml` manifest file. Otherwise, after the vSphere Client deploys your plug-in package, the plug-in will not work because the plug-in dependencies cannot be resolved and may cause errors in the vSphere Client.
  For example, you can use the following lines in the manifest file of your plug-in package to define the minimum supported version of the vSphere Client:

```
...
  <dependencies>
     <pluginPackage id="com.vmware.vsphere.client"
                    version="5.5.0" match="greaterOrEqual" />
  </dependencies>
...
```

  NOTE
  If your plug-in package is only compatible with a specific version of the vSphere Client, you must use the `equal` value of the `match` attribute to specify the version. In this way, you ensure that when you upgrade the vSphere Client, your plug-in package is not deployed, and does not cause any errors.

  NOTE
  If the `match` attribute is not provided, the default value is `greaterOrEqual`.

- To avoid deployment failures, you must create a ZIP archive file for your vSphere Client extension. Moreover, if you want to complete successfully the certification for your vSphere Client plug-in, know that the plug-in signing tool signs only plug-ins that have the ZIP file format.

  NOTE
  For the vSphere 8.0 release, local plug-ins must be upgraded as follows:

  1. Local plug-ins must be FIPS-compliant. See https://developer.vmware.com/docs/13385/preparing-local-plug-ins-for-fips-compliance
  2. Local plug-ins must bundle third-party libraries and installthem in a separate directory. See https://vdc-download.vmware.com/vmwb-repository/dcr-public/77dd6491-bb96-47f3-8c2e-a5a4655f078b/b916bc3c-0168-4487-a9e5-c89b50373c18/Local-Plugins-Library-Isolation.pdf

Best Practices for Plug-In Modules Implementation

Following general design and development recommendations is the first step in creating high-performance and secure vSphere Web Client extensions. You can then move on to special areas, such as developing HTML-based extension solutions.

- Your plug-in package must be OS agnostic. You must avoid reading and writing on the file system from the vSphere Client service layer. In case you need to temporarily store files, you must use the browser cache or your own back-end server.
- To provide a consistent end-user experience in case your vSphere Client extension migrates server workloads, make sure that your extension migrates only to vSphere environments that are hosted by a VMware vCloud Air Network Service Provider. For more information about the available service providers, see http://vcloudproviders.vmware.com/find-a-provider.
- Avoid using deprecated or private APIs and extension points. Using deprecated APIs in your vSphere Client extensions will prevent them from working with future versions of the vSphere Client.
- To prevent performance problems in the vSphere Client and vCenter Server instances, use your Java services only for communication between the vCenter Server instances, or other remote data sources, and the user interface layer. You must not create thread pools in your Java services. Consider implementing any complex business logic in your own backend servers.
- Avoid caching data in the Java service layer. Make sure that the vSphere Client remains stateless. To ensure the scalability of the vSphere Client, you must use your backend server to cache data.
- To increase the security of your extensions, you must limit the access to your plug-ins to specific users. Use the `plugin.xml` extension definition to control the user access to your extensions based on their privileges. For example, you can make your extensions available only to users who have privileges to create or delete Datastore objects.
- To achieve optimal scalability and performance for your vSphere Client plug-ins, your Java services must not require any significant heap allocation.

Best Practices for Developing HTML-Based Extensions

You can use the vSphere Web Client SDK and the vSphere Client development kit to create HTML-based extensions.

Starting with vSphere 5.5 Update 1, an HTML Bridge infrastructure is added to the vSphere Web Client that provides support for HTML-based extensions. The vSphere Web Client SDK provides APIs, tools, and samples that can help you extend the vSphere Web Client.

Starting with vSphere 6.5, you can use the vSphere Client to connect to vCenter Server systems and manage vSphere inventory objects. The vSphere Client development kit is provided to developers that want to create HTML5-based extensions for both Web browser applications. For backward compatibility, the vSphere Client development kit contains the same APIs as the HTML Bridge.

Follow these best practices when you create your HTML-based solutions.

- Make sure that your HTML and JavaScript code is fully functional on different Web browsers and provides the same user experience.
- You must not send calls to the topmost browser window `window.top` or to the parent object of your current window `window.parent`.
- You must include in your HTML-based extensions the latest version of the `web-platform.js` JavaScript file provided with the vSphere Web Client SDK and added to each extension during generation. If you use an older version of this file, your HTML-based extensions might not work in the vSphere Web Client and might cause other HTML-based extensions to stop working.
- To minimize future maintenance work and prevent incompatibility problems, do not change the `web-platform.js` JavaScript file on your own initiative. The file depends on the vSphere Web Client version and is updated with each major release of the SDK. If the file changes between major releases, you must see whether the release notes contain any instructions for manual changes that you must apply to the file before generating your plug-in packages.

- To ensure the integrity of future versions of the vSphere Web Client running HTML-based extensions, do not modify the `WEB_PLATFORM` object. All HTML-based extensions use this global variable to access the vSphere Web Client platform APIs. For example, if you change this variable, other HTML-based extensions that use the `WEB_PLATFORM = self.parent.document.getElementById("container_app")` variable initialization might stop working.

Best Practices for Extending the User Interface Layer

When developing extensions for the user interface layer of the vSphere Client, follow these best practices.

- Create pointer node extensions on the Object Navigator home page only for major applications and solutions. This approach provides consistent and meaningful user experience for the customized vSphere Client.
- When you create action set extensions for a particular type of vSphere object, you must use the extensions filtering mechanism. The defined action sets must be visible only when the user selects the relevant vSphere object type.
- Use the REST API for retrieving data from the service layer. Use proxies only for adding, editing, and deleting issued data requests.
- For better performance, avoid making proxy calls that require more than several seconds to return a response. A best practice is to design your extensions to submit a task that returns immediately, and to track the task progress.
- If you use proxies for data requests, verify that you receive the request response before sending another request through the proxy.
- If you use localization data for your plug-in package, follow these recommendations:
  - Set the `locale` attribute in the `<resource>` element of the `plugin.xml` manifest file to the value `{locale}`. Using the `{locale}` value instructs the vSphere Client to display the plug-in by using the current vSphere Client locale.

    The following XML fragment shows how the `<resource>` element can be used in the plug-in module manifest file.

```
<plugin id="com.vmware.samples.htmlsample"

   defaultBundle="com_vmware_samples_htmlsample">


   <resources baseUrl="locales/">

      <resource>com_vmware_samples_htmlsample</resource>

   </resources>


....


</plugin>
```

  - To avoid collisions with other localized plug-in packages, set a unique resource bundle name to the `defaultBundle` attribute of the `<plugin>` element in the plug-in manifest file. Use your company name and product name as part of the resource bundle name to make it unique.
  - Make sure that the filenames of your resource files end with *`_en_US`* instead of *`-en_US`*

Best Practices for Extending the Service Layer

Following these recommendations and best practices for creating extensions of the vSphere Web Client service layer, can help you improve the security, scalability, and performance of your extension solutions.

- To avoid deployment errors, add your services to the Spring configuration by using the `bundle-context.xml` Spring configuration file. Do not create alternative Spring contexts.

- To increase the deployment speed of your extensions, make sure that you optimize your Spring context initialization. You must use as little source code as possible in the constructor and the initialization method of your Spring beans.
- Avoid using timers for pooling data from the vSphere environment. In case there is no other way to retrieve the required data, you must make sure that data queries are not overlapping.
- If you use a tool to automatically generate the manifest file of your service layer extension, make sure that no third-party packages are added to the `Package-Export` manifest header.

OSGi-Specific Recommendations

Following these OSGi-specific recommendations, helps you improve the performance and scalability of your Java service layer extensions.

- To avoid deployment errors in case your plug-in depends on a third-party library with a different version than the ones available on the application server, you can embed the library inside your bundle. You must also specify the library in your bundle manifest file by using the `Bundle-Classpath` manifest header. In this way, the bundle class loader looks for required classes among the classes from your plug-in and also from the embedded third-party library.
  For example, if your bundle uses classes from the `thirdPartyLibrary.jar`, add the JAR to the root of the bundle and add the following line to the bundle manifest file:
  ```
  Bundle-Classpath: .,thirdPartyLibrary.jar
  ```
  As a result, when you deploy your plug-in on the application server, your bundle dependencies are resolved using the embedded third-party library and not the one that is already on the server.
- To avoid future compatibility issues, make sure that you follow the recommendations of the OSGi Alliance for wiring bundles. Use the `Import-Package` manifest header to declare your package dependencies and not the `Require-Bundle` header.
- To avoid deployment failures in case your bundle imports packages that are exported from `vim25.jar`, remove any packages exported by the `vim25.jar` bundle from the package imports of your `MANIFEST.MF` file. You must add the following line to your `MANIFEST.MF` file:
  ```
  Require-Bundle: com.vmware.vim25;bundle-version=1.0.0
  ```
  You might have deployment issues, if your environment has a plug-in package that contains the `vijava-osgi.jar` bundle.
- To improve the future maintenance of your bundles, you must export as few packages as possible. Remember that every exported package is considered a public API that must be versioned and maintained. If you export packages that contain implementation classes, your specific implementation becomes harder to evolve and to be maintained in the future. Ideally, you must export APIs by using a dedicated API bundles. Other bundles must import the APIs and provide implementation classes that use and publish services. The implementation classes must not export packages.
- To avoid deployment errors, you must not export packages that do not belong to your own code. If you include a third-party bundle in your bundle, do not export any classes from the third-party bundle.
- To avoid future compatibility issues in case you import a package from the vSphere Client bundles, set the package version to `0` in the `MANIFEST.MF` file. When you update the vSphere Client platform, your bundle might stop working if you specified a concrete package version that is not available after the update. If you do not specify a version, the OSGi validation utility logs a warning message in the `plugin-medic.log` file.
  For example, if you import the `com.vmware.vise.data` and `com.vmware.vise.data.query` packages, you must add the following line to your `MANIFEST.MF` file:
  ```
  Import-Package: com.vmware.vise.data;version="0",
  com.vmware.vise.data.query;version="0"
  ```
- To improve the performance of your plug-in package, avoid using the `DynamicImport-Package` manifest header unless necessary. If you use the `DynamicImport-Package` header in your bundle and the packages you want to import are not known in advance, the application framework switches to searching mode for a publicly available package that satisfies the requirement. The use of wildcards is discouraged.
- To improve the deployment time of your plug-in packages, you must add as few bundles as possible to the `<bundlesOrder>` element of your `plugin-package.xml` manifest file. All bundles that are not included in the ordered bundles list are deployed in parallel.
  For example, you can deploy the OSGi bundles from your plug-in package in a parallel manner. This deployment is achieved, if you move all APIs exported by bundle A and imported by bundle B to a separate `my_api.jar` bundle. Include the `my_api.jar` bundle to the ordered bundles list of your plug-in package. In this way, the dependencies of bundle A and B are satisfied in advance and these bundles can start in parallel.
- To improve the deployment time of your plug-in package, do not perform Spring bean initialization in the bundles from the ordered bundles list. The deployment of bundles is blocked until the Spring bean initialization is completed for each bundle that is part of the ordered bundles list. This behavior slows down the startup of the application server. You must use the bundles from the ordered bundles list only to export APIs and data transfer objects, if possible. For more information, see the previous recommendation.
- To speed up the deployment of your plug-in package, you must use as few Web application ARchive (WAR) files as possible, ideally only one WAR file per plug-in package. WAR files are deployed slower that the other bundles,

especially when the Web application has OSGi dependencies. For example, the deployment process can be slowed down when the Web application registers a message broker.
- To avoid runtime errors, you can specify the versions of the packages that you import and export for your OSGi bundle.
- Starting with vSphere 6.7U2, the Tomcat application server logs OSGi, dependency, and deployment errors to `equinox.log`. The `equinox.log` file is a recommended starting point for troubleshooting deployment failures.
- Starting with vSphere 6.5, an OSGi validation utility is added to the vSphere Client which ensures that the deployed plug-ins follow the OSGi-specific best practices. The results from the validation checks are logged to the `plugin-medic.log` file which is located in the same folder as the Tomcat server log file, `vsphere_client_virgo.log`. For more information about the location of the Tomcat server log files, see the Log Files Location table.
  Once the deployment of all plug-ins completes, the validation for the whole set of OSGi bad practices begins. Any issues detected are logged as `INFO` and `WARN` messages to the `plugin-medic.log` file. For example, following are some of the warning messages that can be seen in the log file after you deploy your plug-ins:
  - `DynamicImport-Package should be avoided` - To prevent performance issues during plug-in deployment, you must avoid using the `DynamicImport-Package` manifest header to declare packages that must be looked up at runtime. Using dynamic imports might cause instability issues with the vSphere Client. To complete successfully the certification of your plug-in, use wildcards with caution and avoid using declarations such as the following: `DynamicImport-Package: com.vmware.*`.
  - `Don't use 'com.vmware' prefix for bundle symbolic names and packages` - The warning message is logged when a third-party bundle exports packages with the `com.vmware` prefix and the bundle's symbolic name starts with a different prefix.
  - `Conflicting package exports` - The warning message is logged when two or more plug-ins contain bundles that export the same package. This violation of the recommendations of the OSGi Alliance leads to `ClassNotFoundException`s at runtime that are difficult to troubleshoot. For example, in production environments, this warning message is logged in case two plug-ins contain bundles that export Hibernate or another third-party library with the same version number.

`DataService`-Specific Best Practices

Following these recommendations and best practices for writing Data Service queries, helps you improve the performance and scalability of your extensions.

- To increase the performance of your extension, you must avoid creating constraints, such as `ObjectIdentityConstraint`s, `PropertyConstraint`s, and `RelationalConstraint`s, and defining `OrderingPropertySpec` objects that have multi-valued properties such as collections and arrays.
  For example, when you create a `PropertyConstraint` object that filters all `VirtualMachine` objects based on their `network` property, the filtering process is slowed down. This situation occurs because the back end Data Provider does not support such requests. In such cases, the Data Service fetches the entire data set and then filters the received data.
- To improve the performance of your extension, you must avoid creating constraints and defining `OrderingPropertySpec` objects by using the length of multi-valued properties such as collections and arrays.
  For example, when you create a `PropertyConstraint` object that filters query results by using the property `network._length` for all `VirtualMachine` objects, the filtering process is slowed down. This situation occurs because the back end Data Provider does not support such requests or does not maintain a separate index for property length. In such cases, the Data Service fetches the entire data set and then proceeds with filtering the received data.
- To improve the performance of your extensions, you can use `QuerySpec.resultSpec.maxResultCount` field to limit the returned result set.
- To improve the performance of your extensions in case you use `PropertyConstraint`s, you must use the `com.vmware.vise.data.query.Comparator.EQUALS` comparator instead of a text-matching comparator such as `com.vmware.vise.data.query.Comparator.CONTAINS` and `com.vmware.vise.data.query.Comparator.TEXTUALLY_MATCHES` for the `PropertyConstraint` queries.

Text-matching operations require a specific database indexing which only a few properties, such as `name`, have. If you need to use a text-matching comparator, you can use `CONTAINS` instead of `TEXTUALLY_MATCHES`, because `TEXTUALLY_MATCHES` requires more complex processing.

- To improve the performance of your extensions, you can set a value to the `targetType` field of each `com.vmware.vise.data.PropertySpec` and `com.vmware.vise.data.query.OrderingPropertySpec` object. The Data Service uses the `targetType` field to optimize the performance of the queries.
- To avoid future compatibility issues with your extension, you must avoid using multi-valued properties, such as collections and arrays, as the middle nodes in the property paths.
  For example, you must not use the property path `configurationEx.drsVmConfig.key` for `ClusterComputeResource` objects because the `drsVmConfig` property of the `vim.cluster.ConfigInfoEx` data object is a collection. In this case, you must request the whole `vim.cluster.ConfigInfoEx` data object.
- To avoid future compatibility issues with your extension, you must not use any custom properties defined by the vSphere Client modules. These properties are prone to change in the future. You must use only the properties defined in the vSphere Web Services API for the managed objects and data objects.
- To avoid future compatibility issues with your extension, you must avoid using the `com.vmware.vise.data.query.Conjoiner.EXCEPT` operator in your `CompositeConstraint`s. Instead you must use negation and De Morgan's laws.
- To avoid future incompatibility, avoid using the `relation` field of the `com.vmware.vise.data.PropertySpec` objects.
- To avoid future incompatibility, avoid using the `facets` field of the `com.vmware.vise.data.query.ResultSpec` objects.
- The Data Service uses the value of the `targetType` field to optimize query run time. To improve the performance of your extensions, set the `targetType` field on every constraint except for the following cases:
  - `com.vmware.vise.data.query.ObjectIdentityConstraint` - You must not specify the `targetType` field because the type is already present in the object reference. You can set the type by using the `target` field of the `ObjectIdentityConstraint` class.
  - `com.vmware.vise.data.query.RelationalConstraint` with `hasInverseRelation` field set to true - The `targetType` field is ignored for such constraints.
- To avoid performance issues with your extension in case you use constraints, you must use a specific managed object type as a value for the `targetType` field. For example, if you use an abstract base type such as the `ManagedEntity` managed object type, the speed of the query is slowed down.
- To ease the future optimization of your extensions, you must limit the size of each `CompositeConstraint` by limiting the number of child constraints in the `nestedConstraints` field of the `CompositeConstraint` class, and you must avoid also nesting multiple `CompositeConstraint`.
- Make sure that your Data Provider Adapter takes less than 3 seconds to process a query. If your adapter takes too long to process a request, the Data Service cuts the adapter from the result.

Best Practices for SSL Connections

The following best practices apply to local plug-ins when setting up SSL connections in Java.

- Use TLS 1.2.
  For example:
  ```
  SSLContext sslContext.getInstance("TLSv1.2");
  ```
- Create a dedicated SSL socket factory for each plug-in. Avoid the method `HttpsURLConnection.setDefaultSSLSocketFactory()`, which uses a static factory object that can lead to conflicts between plug-ins.
  For example, use the following method to create an SSL socket factory:
  ```
  SSLSocketFactor socketFactory = sslContext.getSocketFactory();
  ```

  For a more complete example, see the `TrustedService` class in the `html-sample` of the SDK.

Best Practices for Using VimPortType with JAX-WS

The JAX-WS package can leak memory due to repeated creation of `VimPort` connections with the vSphere Web Services SDK. The use of a static `VimPort` object can alleviate the memory leaks, but can lead to other problems, especially with thread safety. JAX-WS does not provide a thread-safe `Port` implementation for the `VimPort` object, which accesses the Web Services API.

Use one or more of the following best practices, depending on the nature of your plug-in:

- For a single-threaded plug-in that does synchronous requests to the vSphere Web Services API, create a static instance of the `VimPort` object and reuse it for subsequent requests.
- For a plug-in that does not need to handle a large number of asynchronous requests, create a static instance of the `VimPort` object and serialize access to it.
- For higher performance with a large number of asynchronous requests, create a pool of `VimPort` objects.
- Use static initializer code to issue a dummy call to the `getPort()` method before your plug-in needs to handle actual requests. The first call to `getPort()` runs slowly because it must load a large amount of WSDL code for the Web Services API.

Best Practices for Deploying and Testing Your vSphere Client Extensions

After you develop your vSphere Client extension, you can follow these recommendations to ensure that your extension is successfully deployed to the vSphere Client.

- To improve the performance of your plug-in package, the initial download and deployment time after the first time the user logs into the vSphere Client, must be less than a minute.
- To ease the testing and debugging of your plug-in package, you must include the build number in the dot-separated version number of the plug-in package when you register the plug-in as a vSphere Client extension.
- To prevent deployment issues when you try to deploy a new version of a registered plug-in package, make sure that you modify the version property of your plug-in package in the `plugin-package.xml` manifest file.
- To prevent deployment issues when you try to deploy a plug-in package with the same version, make sure that you unregister the plug-in package by removing the plug-in as a vCenter Server extension. You must also manually delete the cached files of the plug-in package that are stored on the Virgo server in one of the following locations:

Virgo Server Environment	Location of Cached Packages
vCenter Server Appliance	/etc/vmware/vsphere-ui/vc-packages/vsphere-client-serenity/
Windows OS local development environment	%PROGRAMDATA%\VMware\vCenterServer\cfg\vsphere-client\vc-packages\vsphere-client-serenity\
Mac OS local development environment	/var/lib/vmware/vsphere-client/vsphere-client/vc-packages/vsphere-client-serenity/

- To avoid performance issues, make sure that your plug-in package has only one version registered with the vCenter Server. You must not change the value of the `key` property of the vCenter Server`Extension` data object between releases.
- To verify easily the deployment of your plug-in package and monitor for any issues related to your plug-in, you must know how to work with the Virgo server log files. You can find these log files in one of the following locations:

Table 93: Log Files Location

Environment	Virgo Log Files Location
vSphere Client development environment (Windows or Mac OS)	`html-client-sdk/vsphere-ui/server/logs/vsphere_client_virgo.log`
vCenter Server Appliance 6.5 installation vSphere Client	`/var/log/vmware/vsphere-ui/logs/`

The `vsphere_client_virgo.log` file contains the log information that the Virgo server generates. Problems usually start with the `[ERROR]` tag. Use your plug-in package name or the bundle symbolic name to detect errors caused by your plug-in.

- To log information about your plug-in package, you must use the default logging mechanisms of the vSphere Client. Use the Apache Log4j logging framework to provide debugging information for your plug-in package. The Virgo server uses the Simple Logging Facade for Java (SLF4J) logging API.

VMware Guest SDK Programming Guide

The *Guest SDK Programming Guide* provides information about developing applications using various libraries and interfaces for vSphere guest operating systems.

- The virtual machine guest library VMGuestLib
- Toolbox for extended guest statistics
- DataSets for guest management
- High availability (HA) application monitoring

Revision History

This manual is revised with each product release or when necessary to reflect significant changes.

Table 94: Revision History

Revision Date	Description
5 July 2023	Changed chapter and section titles for search engine optimization.
26 Aug 2022	For ESXi 8.0. New chapter about in-guest DataSets and vSphere API.
2 April 2020	For ESXi 7.0. Added memShares64 for large memory virtual machines.
17 Apr 2018	For ESXi 6.7. No significant changes.
15 Nov 2016	For ESXi 6.5. Minor modifications to “Note on vm.cpu.contention.cpu” section.
16 May 2016	Corrected formulas for extended guest statistics; added **esxtop** comparison.
25 Mar 2016	Added section about **checksystem** utility to verify **glib** version.
22 Jun 2015	Documented the **vmware-appmonitor postAppState** option and arguments.
17 Apr 2015	Added new chapter on fetching extended guest statistics.
29 Sept 2014	Update for ESXi 6.0, with new section about security of Remote RPC.
19 Sept 2013	Update for ESXi 5.5, with new VMGuestAppMonitor_PostAppState function.
17 May 2012	Added vSphere HA Application Monitoring, changed version number for VMware Tools 9.0.
24 Aug 2011	Added information about compatibility with vSphere 5.0.
13 Jul 2010	No new information, but revised to note support for VMware ESX 4.1.
7 May 2009	Revised manual for VMware ESX version 4.0.
29 Nov 2007	No new information, but revised to note support for VMware ESX 3.5 and ESX 3i version 3.5.
18 Jul 2005	Initial release of the VMware Guest SDK providing support for VMware ESX 3.0.

Intended Audience

This book is intended for developers of software to deal with guest operating systems of virtual machines running in the vSphere environment.

Installing the Development Kit

Welcome to the VMware Guest and High Availability (HA) Application Monitoring software development kit.

What the Guest SDK Contains

The Guest and HA Application Monitoring SDK is available as a tarball for Linux or a ZIP file for Windows. Both have a similar directory structure, shown in Table : Components of the SDK, with minor differences for compilation.

Table 95: Components of the SDK

Directory or Folder	Explanation of Contents
`bin/bin32` or `bin/win32` `bin/bin64` or `bin/win64`	Contains the **`vmware-appmonitor`** program, which controls the HA application monitoring heartbeat from the command line.
`docs/`	Contains the Guest SDK terms and conditions, and text of related Open Source licenses. Also contains sample code for HA application monitoring.

`docs/`
`VMGuestAppMonitor/`
`samples/C` or
`samples/visualstudio`
`samples/java`
	The `samples/C` subdirectory (or `samples/visualstudio` subfolder) contains the **`sample.c`** (or `appmon.cpp`) program to demonstrate HA application monitoring API. Follow instructions in the README file to compile with **`make`** or with Visual Studio. The `samples/java` directory contains a Java native interface (JNI) implementation that builds on the C implementation. Again, see the README file.

`include/`	Header files for basic types, GuestAppMonitor and Guest libraries, and session ID.
`include/` `vmGuestLibTest.c`	Sample C program to run all the Guest library functions and return statistics. On Linux, use **`gcc`** to compile this program, and run it on an ESXi hosted virtual machine.
`lib/lib32` or `lib/win32` `lib/lib64` or `lib/win64`	Shared objects or DLL files and libraries for the Guest library, the Guest library for Java, and the HA application monitoring library.
`vmGuestLibJava` `vmGuestLibJava/doc`	JAR file and standard Javadoc for a prepackaged Java implementation of the Guest API. For a list of methods, browse `index.html` and see the VMGuestLibInterface page.

Displaying vSphere Guest Library Statistics

On a Linux virtual machine hosted by ESXi, go to the **`include`** directory and compile the **`vmGuestLibTest.c`** program. Run the output program **`vmguestlibtest`**.

```
gcc -g -o vmguestlibtest -ldl vmGuestLibTest.c
```

```
./vmguestlibtest
```

Guest statistics appear repeatedly until you interrupt the program.

Security of Remote RPC

Guest RPC is a communication channel between the guest operating system and its VMX, or virtual machine executable, the user space component of virtual infrastructure. The VMM, or virtual machine monitor, is the kernel space component.

In the ESXi 6.0 release, Guest RPC was reimplemented on top of VMCI Sockets.

To enforce security for the Guest SDK and HA Application Monitoring SDK, allowing only root and Administrator access to the functions provided by the SDK, on ESXi hosts you can edit the `.vmx` file for the respective virtual machine and set the secure authentication parameter as follows:

```
guest_rpc.rpci.auth.app.APP_MONITOR = TRUE
```

If you do not need to enforce security and want to allow non-root and non-Administrator users to access functions in the Guest and HA Application Monitoring SDK, the secure authentication parameter must not appear in the `.vmx` file, or it must be set **`FALSE`**.

Security of the DataSets API

The DataSets facility can be controlled from vCenter Server, so any security concerns are the same as for vSphere generally.

The DataSets facility can also be controlled with VM Tools from inside a guest OS, so security concerns are the same as for **vmtoolsd**.

Accessing HA Application Monitoring

This section provides an introduction to HA Application Monitoring in the Guest SDK.

To proceed further you'll also need information in the vSphere HA Application Monitoring chapter.

SDK function definitions and simple documentation are in the `vmGuestAppMonitorLib.h` include file.

Controlling the Application Monitoring Heartbeat

To run HA application monitoring programs, the virtual machine must be running on an ESXi host, and application monitoring must have been enabled when configuring HA.

You can enable heartbeats with the compiled **vmware-appmonitor** program. Usage is as follows:

```
vmware-appmonitor { enable | disable | markActive | isEnabled | getAppStatus |
postAppState }
```

- **enable** – Enable application heartbeat so vSphere HA starts listening and monitoring the heartbeat count from this guest virtual machine. The heartbeats should be sent at least once every 30 seconds.
- **disable** – Disable the application heartbeat so vSphere HA stops listening to heartbeats from this guest.
- **markActive** – This starts sending the actual heartbeat every 30 seconds or less.
- **isEnabled** – Indicates whether the heartbeat monitoring was enabled.
- **getAppStatus** – Gets the status of the application, either Green, Red, or Gray.
- **postAppState** – Posts the state of the application. Arguments can be:
  - **appStateOk** – Sends an “Application State is OK” signal to the HA agent running on the host.
  - **appStateNeedReset** – Sends an “Immediate Reset” signal to the HA agent running on the host.

Library Path or Path Environment

On Linux, set your **LD_LIBRARY_PATH** environment to the install location of `GuestSDK/lib/lib32` or **lib64**. On Windows, you can set your **PATH** environment, but it is probably easier to copy **vmware-appmonitor** to the same folder as the DLL files.

Compiling the Sample Program on Linux

You need a C compiler and the **make** program.

1. Go to the `docs/VMGuestAppMonitor/samples/C` directory.
2. Run the **make** command.

   On a 64-bit machine you might want to change **lib32** to **lib64** in the **makefile**.

3. Set **LD_LIBRARY_PATH** as described above.
4. Run the sample program. See below for program usage.

   ```
   ./sample
   ```

Compiling Sample Programs on Windows

You need Visual Studio 2008 or later.

1. Go to the `docs/VMGuestAppMonitor/samples/visualstudio` folder.
2. Open the **`appmon.vcproj`** file and build the solution.
3. Click **Debug › Start Debugging** to run `appmon.exe`. See below for program usage.

Demonstrating the HA Application Monitoring API

The sample program enables HA application monitoring and sends a heartbeat every 15 seconds. After the program starts running, typing Ctrl+C displays three choices:

- s – stop sending heartbeats and exit the program. The virtual machine will reset.
- **`d`** – disable application monitoring and exit the program. This does not cause a reset.
- **`c`** – continue sending heartbeats.

The VM Guest Library

The VM Guest library provides data collection functions that you can call from a program that runs in the guest operating system of a virtual machine on an ESXi host.

About the VM Guest Library

The VM Guest API provides functions that management agents and other software can use to collect data about the state and performance of a VMware virtual machine. The VM Guest API provides fast access to resource management information, without the need for authentication.

The VM Guest API provides read-only access. You can read data using the API, but you cannot send control commands. To send control commands, use the vSphere Web Services SDK. For more information, see the *`VMware vSphere Web Services SDK Programming Guide`* and the *`VMware vSphere API Reference`*, both of which are available on the VMware developer support Web site.

The VM Guest API version number matches the installed VMware Tools version number.

Supported Guest Operating Systems

The VM Guest API software runs on most Windows or Linux guest operating systems supported by the various versions of ESXi.

See the *`VMware Compatibility Guide`* for a list of supported guest operating systems and versions. This guide is now located at https://www.vmware.com/resources/compatibility/search.phpas a Web database. Look for Guest OS.

The Guest API does not support the following guest operating system environments:

- Windows 95 and Windows 98.
- Windows NT 4.0 and earlier.

Prerequisite LibC Version

To help you verify the minimum C library version, the **`checksystem`** utility is bundled in the Guest SDK package for Linux systems. After you extract the Guest SDK, the utility is available in the `GuestSDK/bin` subdirectory.

If the Linux guest OS has a compatible **`glibc`**, the **`checksystem`** utility prints this message:

```
This version of GuestSDK is compatible with the version of glibc in this system.
```

If the Linux guest OS does not have a compatible **`glibc`**, the utility prints this error message:

```
This version of GuestSDK requires version 2.5 or later of glibc.
```

For such a system, you might be able to use an older version of the Guest SDK.

Virtual Machine Statistics

With the VM Guest API, you can monitor various statistics about a virtual machine. You can use this information to retrieve scheduling and resource usage information about its environment. With the help of these statistics, a virtual machine can immediately react to changes in its virtual environment at the application layer.

The information you can retrieve by using the VM Guest API includes:

- Amount of memory reserved for the virtual machine.
- Amount of memory being used by the virtual machine.
- Upper limit of memory available to the virtual machine.
- Number of memory shares assigned to the virtual machine.
- Maximum speed of the virtual machine's CPU.
- Reserved rate at which the virtual machine is allowed to execute. An idling virtual machine might consume CPU cycles at a much lower rate.
- Number of CPU shares assigned to the virtual machine.
- Elapsed time since the virtual machine was last powered on or reset.
- CPU time consumed by a particular virtual machine. When combined with other measurements, you can estimate how fast the virtual machine's CPUs are running compared to the host CPUs.

  IMPORTANT
  The API uses a handle that provides access to the statistics. The handle also is a mechanism to determine whether the API can provide accurate information. (Certain events, such as migrating a virtual machine with VMotion™, temporarily make it impossible to provide accurate information.)

How to Use the VM Guest API

The VM Guest API defines functions and data types that you use to extract virtual machine data. This section covers the following topics:

- VM Guest API Runtime Components
- VM Guest API Data Types
- VM Guest Library Functions
- VM Guest API Error Codes

VM Guest API Runtime Components

To use the vSphere Guest API, the runtime components must be installed in the guest operating system. The runtime components are dynamically loaded binary modules for 32-bit and 64-bit guests. When you install VMware Tools, the vSphere Guest API runtime components are installed as well. You can also download them from http://www.vmware.com/download/sdk/guest_sdk.html.

To make the vSphere Guest API functions available to your program, use your program's standard methods to load the library.

- In a Windows guest operating system, the library file is `vmGuestLib.dll`. The import library file is `vmGuestLib.lib`.
- In a Linux guest operating system, the library file is **`libvmGuestLib.so`**.

If you are using a Security-Enhanced Linux (SELinux) guest OS, its security policies might interfere with dynamic loading of **libvmGuestLib.so**. Refer to documentation about your SELinux policy configuration

The vSphere Guest SDK includes the test program **vmGuestlibTest.c**. If you are using a Windows environment, you must rebuild the test program. The vmGuestLib.dll library file is a non-Unicode DLL. In Microsoft Visual Studio, build the test program **vmGuestlibTest.c** as a non-Unicode executable file so that the program can access the DLL at runtime.

Enabling and Disabling the Runtime Components

The vSphere Guest API runtime components are enabled by default (**disable = "FALSE"**). To disable the runtime components, use the configuration editor in the vSphere Client to edit the configuration file for the virtual machine. The virtual machine must be powered off before you can use the configuration editor.

1. In the vSphere Client window, right-click the virtual machine in the machine list.
2. In the drop-down menu, select Edit Settings.
3. In the Virtual Machine Properties window, click the Options tab.
4. In the list of "Advanced" settings, select General.
5. Click Configuration Parameters.
6. In the Configuration Parameters window, add the following line or, if the file already contains the disable configuration setting, set the value to TRUE:
   ```
   isolation.tools.guestlibGetInfo.disable = "TRUE"
   ```
   The default value for the disable setting is FALSE. The default setting enables the runtime components. Reinstalling VMware Tools does not affect the disable setting. If you disable the vSphere Guest API and reinstall VMware Tools, the vSphere Guest API remains unavailable until you change the configuration setting **guestLibGetInfo.disable** to FALSE.

VM Guest API Data Types

The VM Guest API uses the data types listed in Table : Data Types to support access to virtual machine data.

Table 96: Data Types

Data Type	Description
VMGuestLibHandle	Reference to virtual machine data. **VMGuestLibHandle** is defined in **vmGuestLib.h**.
VMSessionID	Unique identifier for a session. The session ID changes after a virtual machine is migrated using VMotion, suspended and resumed, or reverted to a snapshot. Any of these events is likely to render any information retrieved with this API invalid. You can use the session ID to detect those events and react accordingly. For example, you can refresh and reset any state that relies on the validity of previously retrieved information. Use **VMGuestLib_GetSessionId** to obtain a valid session ID. A session ID is opaque. You cannot compare a virtual machine session ID with the session IDs from any other virtual machines.You must always call **VMGuestLib_GetSessionId** after calling **VMGuestLib_UpdateInfo**. **VMSessionID** is defined in **vmSessionId.h**.
VMGuestLibError	Status code that indicates success or failure. Each function returns a **VMGuestLibError** code. For information about specific error codes, see VM Guest

Table continued on next page

Continued from previous page

	API Error Codes. **`VMGuestLibError`** is an enumerated type defined in **`vmGuestLib.h`**.

VM Guest Library Functions

The vSphere Guest SDK contains the header file **`vmGuestLib.h`**. This file declares the functions and data types that you use to call the VM Guest API. The following sections describe the VM Guest API functions:

- Calling Context Functions
- Accessor Functions (Virtual Machine)

Calling Context Functions

The VM Guest API provides a set of functions that initialize and manipulate the context where the Guest API operates. Before your application can use the accessor functions to retrieve information about a virtual machine, use the following functions to initialize the VM Guest API environment.

1. Call the **`VMGuestLib_OpenHandle`** function to obtain a handle for accessing information about the virtual machine. The guest library handle is a parameter to every Guest API function.
2. Call the **`VMGuestLib_UpdateInfo`** function to update the information available through the handle.
3. Call the **`VMGuestLib_GetSessionId`** function to retrieve a session ID.

About Context Functions

The code block below shows a C language fragment that illustrates the function calls for initialization. (Code fragments in this section do not perform error handling. For information about error handling, see VM Guest API Error Codes.)

Initializing the VM Guest API Environment

```
VMGuestLibHandle glHandle;

VMGuestLibError glError;

VMSessionId sid = 0;

glError = VMGuestLib_OpenHandle(&glHandle);

glError = VMGuestLib_UpdateInfo(glHandle);

glError = VMGuestLib_GetSessionId(glHandle, &sid);
```

You can use the session ID to detect changes that invalidate previously retrieved data. The code block below shows a fragment that illustrates how to use the session ID to detect stale data. The **`ResetStats`** function in the following fragment represents application code to handle the session change.

Detecting Stale Data

```
VMGuestLibHandle glHandle;

VMGuestLibError glError;

VMSessionId oldSid;

VMSessionId newSid;


/* [...code here would access data based on an existing, valid session ID (oldSid)...] */
```

```
/* Update the library, get the session ID, and compare it to the previous session ID */
glError = VMGuestLib_UpdateInfo(glHandle);
glError = GuestLib_GetSessionId(glHandle, &newSid);
if (oldSid != newSid) {
    ResetStats();
    oldSid = newSid;
}
```

Table : Open, Close, and Update Functions lists the context functions for creating and releasing handles, updating information, and obtaining session IDs.

Table 97: Open, Close, and Update Functions

Function	Description
VMGuestLib_OpenHandle	Gets a handle for use with other VM Guest API functions. The guest library handle provides a context for accessing information about the virtual machine. Virtual machine statistics and state data are associated with a particular guest library handle, so using one handle does not affect the data associated with another handle.
VMGuestLib_CloseHandle	Releases a handle acquired with **VMGuestLib_OpenHandle**.
VMGuestLib_UpdateInfo	Updates information about the virtual machine. This information is associated with the **VMGuestLibHandle**. **VMGuestLib_UpdateInfo** requires similar CPU resources to a system call and therefore can affect performance. If you are concerned about performance, minimize the number of calls to **VMGuestLib_UpdateInfo**. If your program uses multiple threads, each thread must use a different handle. Otherwise, you must implement a locking scheme around update calls. The VM Guest API does not implement internal locking around access with a handle.
VMGuestLib_GetSessionId	Retrieves the **VMSessionID** for the current session. Call this function after calling **VMGuestLib_UpdateInfo**. If **VMGuestLib_UpdateInfo** has never been called, **VMGuestLib_GetSessionId** returns **VMGUESTLIB_ERROR_NO_INFO**.

Accessor Functions (Virtual Machine)

Accessor functions retrieve information about a virtual machine. When you call an accessor function, you pass a guest library handle (type **VMGuestLibHandle**) to the function. If the function is successful, it returns the requested data as an output parameter. The function return value is an error code (type **VMGuestLibError**) that indicates success or failure. The code block below shows a C language fragment that illustrates calling **VMGuestLib_GetCpuLimitMHz** to retrieve the processor limit available to the virtual machine.

Using an Accessor Function

```
uint32 cpuLimitMHz = 0;
glError = VMGuestLib_GetCpuLimitMHz(glHandle, &cpuLimitMHz);
```

When a call completes successfully, the function returns the value **VMGUESTLIB_ERROR_SUCCESS**. If the call is unsuccessful, the error code name contains an appropriate description. For details, see VM Guest API Error Codes.

Call **VMGuestLib_UpdateInfo** once to refresh all statistics before calling an accessor function or a series of accessor functions.

Table 98: Accessor Functions for Virtual Machine Data

Function	Description
VMGuestLib_GetCpuLimitMHz	Retrieves the upper limit of processor use in MHz available to the virtual machine. For information about setting the CPU limit, see Limits and Reservations.
VMGuestLib_GetCpuReservationMHz	Retrieves the minimum processing power in MHz reserved for the virtual machine. For information about setting a CPU reservation, see Limits and Reservations.
VMGuestLib_GetCpuShares	Retrieves the number of CPU shares allocated to the virtual machine. For information about how an ESXi host uses CPU shares to manage virtual machine priority, see the *vSphere Resource Management Guide*.
VMGuestLib_GetCpuStolenMs	Retrieves the number of milliseconds that the virtual machine was in a ready state (able to transition to a run state), but was not scheduled to run.
VMGuestLib_GetCpuUsedMs	Retrieves the number of milliseconds during which the virtual machine has used the CPU. This value includes the time used by the guest operating system and the time used by virtualization code for tasks for this virtual machine. You can combine this value with the elapsed time (**VMGuestLib_GetElapsedMs**) to estimate the effective virtual machine CPU speed. This value is a subset of elapsedMs.
VMGuestLib_GetElapsedMs	Retrieves the number of milliseconds that have passed in the virtual machine since it last started running on the server. The count of elapsed time restarts each time the virtual machine is powered on, resumed, or migrated using VMotion. This value counts milliseconds, regardless of whether the virtual machine is using processing power during that time. You can combine this value with the CPU time used by the virtual machine (**VMGuestLib_GetCpuUsedMs**) to estimate the effective virtual machine CPU speed. cpuUsedMS is a subset of this value.
VMGuestLib_GetHostProcessorSpeed	Retrieves the speed of the ESXi host's physical CPU in MHz.
VMGuestLib_GetMemActiveMB	Retrieves the amount of memory the virtual machine is actively using—its estimated working set size.
VMGuestLib_GetMemBalloonedMB	Retrieves the amount of memory that has been reclaimed from this virtual machine by the vSphere memory balloon driver, which is also called the "vmmemctl" driver.
VMGuestLib_GetMemLimitMB	Retrieves the upper limit of memory that is available to the virtual machine. For information about setting a memory limit, see Limits and Reservations.
VMGuestLib_GetMemMappedMB	Retrieves the amount of memory that is allocated to the virtual machine. Memory that is ballooned, swapped, or has never been accessed is excluded.
VMGuestLib_GetMemOverheadMB	Retrieves the amount of "overhead" memory associated with this virtual machine that is currently consumed on the host system. Overhead memory is additional memory that is reserved for data structures required by the virtualization layer.
VMGuestLib_GetMemReservationMB	Retrieves the minimum amount of memory that is reserved for the virtual machine. For information about setting a memory reservation, see Limits and Reservations.

Table continued on next page

Continued from previous page

VMGuestLib_GetMemSharedMB	Retrieves the amount of physical memory associated with this virtual machine that is copy-on-write (COW) shared on the host.
VMGuestLib_GetMemSharedSavedMB	Retrieves the estimated amount of physical memory on the host saved from copy-on-write (COW) shared guest physical memory.
VMGuestLib_GetMemShares	Retrieves the number of memory shares allocated to the virtual machine, as an unsigned 32-bit value. If the value cannot fit in 32-bits, this call returns an error. In that case, use `VMGuestLib_GetMemShares64` instead. For information about how ESXi hosts use memory shares to manage virtual machine priority, see the *vSphere Resource Management Guide*.
VMGuestLib_GetMemShares64	This call was added in vSphere 7.0 to get the 64-bit value of memory shares allocated to the virtual machine.
VMGuestLib_GetMemSwappedMB	Retrieves the amount of memory that has been reclaimed from this virtual machine by transparently swapping guest memory to disk.
VMGuestLib_GetMemTargetSizeMB	Retrieves the size of the target memory allocation for this virtual machine.
VMGuestLib_GetMemUsedMB	Retrieves the estimated amount of physical host memory currently consumed for this virtual machine's physical memory.
VMGuestLib_GetResourcePoolPath	Retrieves the path name of the resource pool affiliated with the virtual machine on the ESXi host where it is running. VMGuestLib_GetResourcePoolPath uses an additional parameter to determine the size of the path name output string buffer. **VMGuestLibError VmGuestLib_GetResourcePoolPath(VMGuestLibHandle handle, size_t *bufferSize, char *pathBuffer);** **handle** is an input parameter specifying the guest library handle. **bufferSize** is an input/output parameter. It is a pointer to the size of the pathBuffer in bytes. If bufferSize is not large enough to accommodate the path (including a NULL terminator), the function returns VMGUESTLIB_ERROR_BUFFER_TOO_SMALL. In this case, the function uses the bufferSize parameter to return the number of bytes required for the string. **pathBuffer** is an output parameter. It is a pointer to a buffer that receives the resource pool path string. The path string is NULL-terminated. For information about using resource pools, see the *vSphere Resource Management Guide*.

For more information about ESXi resource management, see the *vSphere Resource Management Guide*, available on the VMware Web site.

Limits and Reservations

You use the Guest API to retrieve information about limits and reservations for CPU and memory. To set limits and reservations, you can use the vSphere (Web) Client or the vSphere API. Setting limits and reservations on a virtual machine ensures that resources are available to that machine and to other virtual machines that draw resources from the same resource pool.

To use the vSphere Client to set limits or reservations:

1. In the vSphere Client window, click the Resource Allocation tab.

2. In either the CPU or Memory section, click Edit.
3. In the Virtual Machine Properties Window, click the Resources tab.
4. Select either the CPU or Memory setting.
5. Use the slider controls to set limits or reservations.

For more information, see online help for the vSphere Client.

To use the vSphere API to set limits or reservations, call the ReconfigVM_Task method. In the method call, use the VirtualMachineConfigSpec data object to set the cpuAllocation or memoryAllocation property. These properties are of type ResourceAllocationInfo, which has limit and reservation properties. For more information, see the VMware vSphere API Reference Documentation.

VM Guest API Error Codes

Each vSphere Guest API function returns an error code. If successful, the returned error code is **`VMGUESTLIB_ERROR_SUCCESS`**. If the function is unable to complete its task, the error code provides information for diagnosing the problem.

Use the **`VMGuestLib_GetErrorText`** function to retrieve the error text associated with a particular error code. When you call the function, pass the error code to the function; **`VMGuestLib_GetErrorText`** returns a pointer to a string containing the error text.

The code block below shows error handling. The C code fragment declares a guest library handle and calls the function **`VMGuestLib_OpenHandle`**. If the call is not successful, the code calls **`VMGuestLib_GetErrorText`** and displays the error text.

Error Handling

```
VMGuestLibHandle glHandle;

glError = VMGuestLib_OpenHandle(&glHandle);

if (glError != VMGUESTLIB_ERROR_SUCCESS) {

   printf("OpenHandle failed: %s\n", VMGuestLib_GetErrorText(glError));

}
```

Table : Error Codes lists all error codes defined for the vSphere Guest API.

Table 99: Error Codes

Error Code	Description
`VMGUESTLIB_ERROR_SUCCESS`	The function has completed successfully.
`VMGUESTLIB_ERROR_OTHER`	An error has occurred. No additional information about the type of error is available.
`VMGUESTLIB_ERROR_NOT_RUNNING_IN_VM`	The program making this call is not running on a VMware virtual machine.
`VMGUESTLIB_ERROR_NOT_ENABLED`	The vSphere Guest API is not enabled on this host, so these functions cannot be used. For information about how to enable the library, see Calling Context Functions.
`VMGUESTLIB_ERROR_NOT_AVAILABLE`	The information requested is not available on this host.
`VMGUESTLIB_ERROR_NO_INFO`	The handle data structure does not contain any information. You must call **`VMGuestLib_UpdateInfo`** to update the data structure.

Table continued on next page

Continued from previous page

VMGUESTLIB_ERROR_MEMORY	There is not enough memory available to complete the call.
VMGUESTLIB_ERROR_BUFFER_TOO_SMALL	The buffer is too small to accommodate the function call. For example, when you call **VMGuestLib_GetResourcePoolPath**, if the path buffer is too small for the resulting resource pool path, the function returns this error. To resolve this error, allocate a larger buffer.
VMGUESTLIB_ERROR_INVALID_HANDLE	The handle that you used is invalid. Make sure that you have the correct handle and that it is open. It might be necessary to create a new handle using **VMGuestLib_OpenHandle**.
VMGUESTLIB_ERROR_INVALID_ARG	One or more of the arguments passed to the function were invalid.
VMGUESTLIB_ERROR_UNSUPPORTED_VERSION	The host does not support the requested statistic.

Tools for Extended Guest Statistics

As of the vSphere 6.0 release, VMware Tools have expanded support for guest operating system statistics. Rather than offering a fixed set of statistics after calling **VMGuestLib_UpdateInfo()**, a larger and extensible set of statistics are available by calling **VMGuestLib_StatGet()**, or by using the CLI.

Guest OS and Statistics Fetch

The fetch-statistics facility returns semi-structured data in a variety of formats for use within a the guest OS. Four encodings are supported:

- text – simple key-value pairs
- XML – eXtensible Markup Language fragment
- JSON – JavaScript Object Notation
- YAML – Yet Another Markup Language, a human-readable JSON superset intended for data transmission

Statistics are accurate within the current session only. A session represents a powered-on virtual machine running on a single ESXi host. Sessions may be discontinuous across session changes, so monotonically increasing metrics can suddenly decrease. New sessions result from suspending a virtual machine, reverting to a snapshot, vMotion, or certain types of hot-plug operations. It is up to the consumer of statistics to notice that the session has changed and deal with numerical discontinuities.

Guest OS statistics are supported for troubleshooting and support only. Specifically:

- Individual metrics are not guaranteed to be forward or backward compatible. Programs using these metrics are expected to tolerate incompatibility, and are cautioned against programming practices that could break when a metric behaves differently.
- Metrics might be added, removed, or modified to have a different meaning. For example: **co-stop** time changed meaning between ESXi 4.1 and ESXi 5.0.
- VMware reserves the right to make changes, but will try to avoid gratuitous changes. Most changes will still provide equivalent information, but possibly in a different form that better reflects the ESXi host implementation providing the metrics.

This support declaration is similar to the compatibility guarantee for the **esxtop** command, whose statistics the Guest SDK reflects. See https://communities.vmware.com/docs/DOC-9279 for details about **esxtop**.

Prerequisites

Before you start, install the latest version of VMware Tools in all relevant virtual machines.

For extended guest statistics, you must have VMware Tools version 9.10 or later installed in a virtual machine running on an ESXi 6.0 or later host. There is no minimum virtual hardware version. Version 9.10 was released with vSphere 6.0.

Earlier ESXi hosts return no data for extended statistics requests, and earlier VMware Tools lack extended statistics. If statistics are unavailable, a code indicates that the requested item is unsupported.

Guest Statistics Interfaces

You have a choice of three interfaces to fetch the new statistics: the guest SDK library, CLI, or raw RPC.

Guest SDK Library

The Guest SDK library now offers two new functions, get and free.

VMGuestLib_StatGet

```
/* Semi-structured hypervisor statistics collection, for troubleshooting.

 */

VMGuestLibError

VMGuestLib_StatGet(const char *encoding, // IN

const char *stat,    // IN

char **reply,        // OUT

size_t *replySize); // OUT
```

- **`encoding`** – "text" or "xml" or "json" or "yaml" – if not specified, "text" is the default.
- **`stat`** – the statistic to print. See examples below.
- **`reply`** – a pointer to be set with a buffer containing the formatted reply. All current formats return null-terminated C strings, but future formats may not; the caller should treat the buffer as binary unless the format is known. The buffer must later be freed by a call to **`VMGuestLib_StatFree()`**.
- **`replySize`** – a pointer to receive the size of data in the buffer.

VMGuestLib_StatFree

To free the memory returned by **`VMGuestLib_StatGet`**, call **`VMGuestLib_StatFree()`**.

```
void

VMGuestLib_StatFree(char *reply, size_t replySize);
```

- **`reply`** – the pointer that was supplied by the **`reply`** parameter of **`VMGuestLib_StatGet()`**.
- **`replySize`** – the size that was supplied by the **`replySize`** parameter of **`VMGuestLib_StatGet()`**.

C code with StatGet and StatFree functions shows these two function calls used in a sample routine:

C code with StatGet and StatFree functions

```
/*

 * Retrieves semi-structured statistics on ESXi host.

 */

static int

StatGetRaw(const char *encoding, // IN

const char *stat,   // IN
```

```
const char *param)  // IN
{
   int exitStatus = EXIT_SUCCESS;
   VMGuestLibError glError;
   char *result = NULL;
   size_t resultSize = 0;
   char *arg = g_strdup_printf("%s %s", stat, param);

   glError = VMGuestLib_StatGet(encoding, arg, &result, &resultSize);
   if (glError != VMGUESTLIB_ERROR_SUCCESS) {
      exitStatus = EX_TEMPFAIL;
   } else {
      g_print("%*s", (int)resultSize, result);
   }
   VMGuestLib_StatFree(result, resultSize);
   g_free(arg);
   return exitStatus;
}
```

Command Line Interface

The command name varies with each of the three OS types available, but all three have a **`-v`** option to show the VMware Tools version number.

```
Linux:   vmware-toolbox-cmd stat raw <encoding> <stat>
Mac:     vmware-tools-cli stat raw <encoding> <stat>
Windows: VMwareToolboxCmd.exe stat raw <encoding> <stat>
```

These commands are provided for debugging and scripting; the implementation is a wrapper on top of the **`VMGuestLib_StatGet()`** call described above.

Raw RPC interface

The raw RPC interface varies for one of the three OS types available.

```
Linux:       vmtoolsd --cmd="guestlib.stat.get <encoding> <stat>"
Mac:         vmtoolsd --cmd="guestlib.stat.get <encoding> <stat>"
Windows:     vmtoolsd.exe --cmd="guestlib.stat.get <encoding> <stat>"
```

This is a raw form of the statistics API. Function calls or CLI are preferred.

Fetching a List of Statistics

To fetch a list of statistics, make a query with no statistics name supplied (in other words, no argument, **`NULL`**, or **`""`**). The returned buffer contains a list of available metric categories. The list can be emitted in any of four supported formats. The **`text`** format is recommended for user interaction; machine-parsable formats **`xml`**, **`json`**, and **`yaml`** are recommended for program interaction.

The following examples use the CLI to demonstrate. The C language API works similarly. In the first example, the virtual machine has three disks and two network interfaces.

Return list of statistics

```
$ vmware-toolbox-cmd stat raw

session

host

resources

vscsi ide0:0

vscsi scsi0:0

vscsi scsi0:2

vnet 00:0c:29:1e:23:f3

vnet 00:0c:29:1e:23:f4
```

The second example fetches a specific statistic (vscsi scsi0:0) taken from the first example.

Get I/O statistics for a device

```
$ vmware-toolbox-cmd stat raw text vscsi scsi0:0

num.reads = 12605

num.writes = 1039

size.reads = 533612032

size.writes = 14279680

latency.reads = 1944173239

latency.writes = 102025122
```

Metadata Fields

When appearing in machine-parsable formats, numeric statistics have these self-describing metadata fields:

`units` – What the numbers mean. Example values:

`packets`, **`bytes`**, **`KB`** for kilobytes, **`us`** for microseconds, and others as appropriate.

`type` – Indicates how to interpret the number.

`static` – The value is for configuration and is not expected to change frequently. For instance: a reservation, or the maximum CPU speed.

`instant` – Instantaneous measurement of the current value. Expected to go up or down over time. For instance: current memory usage.

`accumulate` – Continuous sum of all data over time. Non-decreasing. The statistics consumer must compute the difference between two values at different times and convert it into a rate. For instance: CPU time used, bytes sent, memory swapped in.

Four examples below show the same statistic, encoded in different formats:

Text

```
$ vmware-toolbox-cmd stat raw text vnet 00:0c:29:1e:23:f3
size.tx = 38137
num.rx = 10920
size.rx = 1312789
reservation = 0
limit = -1
```

YAML

```
$ vmware-toolbox-cmd stat raw yaml vnet 00:0c:29:1e:23:f3
num.tx:
    type: accumulate
    units: packets
    value: 209
size.tx:
    type: accumulate
    units: bytes
    value: 38137
num.rx:
    type: accumulate
    units: packets
    value: 10920
size.rx:
    type: accumulate
    units: bytes
    value: 1312789
```

```
reservation:
    type: static
    units: MBps
    value: 0
limit:
    type: static
    units: MBps
    value: -1
```

XML (formatted here for presentation)

```
$ vmware-toolbox-cmd stat raw xml vnet 00:0c:29:1e:23:f3
<metrics session="4004861987670969122">
<metric name="num.tx" type="accumulate" units="packets">209</metric>
<metric name="size.tx" type="accumulate" units="bytes">38137</metric>
<metric name="num.rx" type="accumulate" units="packets">10992</metric>
<metric name="size.rx" type="accumulate" units="bytes">1322161</metric>
<metric name="reservation" type="static" units="MBps">0</metric>
<metric name="limit" type="static" units="MBps">-1</metric>
</metrics>
```

JSON (formatted here for presentation)

```
$ vmware-toolbox-cmd stat raw json vnet 00:0c:29:1e:23:f3
{"num.tx":{"type":"accumulate","units":"packets","value":209},
 "size.tx":{"type":"accumulate","units":"bytes","value":38137},
 "num.rx":{"type":"accumulate","units":"packets","value":11068},
 "size.rx":{"type":"accumulate","units":"bytes","value":1331791},
 "reservation":{"type":"static","units":"MBps","value":0},
 "limit":{"type":"static","units":"MBps","value":-1}}
```

Metrics Examples

The examples below use the default “text” format for readability. For actual programming, you will probably use one of the machine-parsable formats (XML, JSON, or YAML).

Fetch Available Statistics

The statistics command with no parameters fetches a list of available statistics:

```
$ vmware-toolbox-cmd stat raw

session

host

resources

vscsi scsi0:0

vnet 00:0c:29:1e:23:f3
```

session – the current session. This changes infrequently. Session changes can cause discontinuities.

host – information about the current hypervisor and hardware the virtual machine is running on.

resources – the CPU and memory usage of this virtual machine.

vscsi <name> – storage statistics for a specific virtual disk, where <name> is the internal name in the virtual machine's configuration file. The library does not aggregate storage statistics across disks.

vnet <Ethernet Address> – network statistics for a specific virtual NIC, where <Ethernet Address> is the MAC address currently programmed into the virtual NIC. The library does not aggregate network statistics across NICs.

For information about a specific statistic, enter its name.

Get Session Information

The statistics command with parameters fetches specific statistics:

```
$ vmware-toolbox-cmd stat raw text session

session = 4004861987670969122

uptime = 1036293956

version = VMware ESXi 6.0.0 build-12345

provider =

uuid.bios = 56 4d 2c 53 43 56 66 8e-7c 05 fd 7e 51 1e 23 f3
```

Sessions change for a virtual machine with power on, suspend, revert to snapshot, vMotion, reset, or when it experiences some type of hot-plug. Statistics might be discontinuous across a session change. The reason for a session change, or whether session change constitutes a host change, is not exposed to virtual machines.

session – a cryptographically strong random number indicating the current session. Expected to contain at least 63 bits of entropy. Changes with every session.

uptime – microseconds since the last session change, as measured by the host.

version – string representation of the hypervisor version. Not expected to be parsed. A guest OS should never change its behavior based on the hypervisor version or build number.

provider – string representing the provider. Set by the ExtraConfig **tools.guestlib.stat.provider** and intended for use by vSphere providers such as vCloud Air. Opaque contents defined by the provider.

`uuid.bios` – the virtual machine's SMBIOS UUID, cached at boot time. VMware maintains the SMBIOS UUID as a component of guest licensing; migrating a VM retains the same UUID while cloning a VM generates a different UUID. The algorithm to generate a new UUID varies from release to release. The vSphere API permits changing a VM's UUID. Although vSphere prevents unintentional duplication of UUID, it allows deliberate UUID duplication, because legitimate workflows (such as lab environments) often require duplicated UUIDs.

Host Hardware

In the next example, only the first two values are provided by default. A virtual machine with ExtraConfig **`tools.guestlib.enableHostInfo =TRUE`** (a non-default setting) supplies the remaining values.

```
$ vmware-toolbox-cmd stat raw text host

host.cpu.processorMHz = 2399

host.cpu.coresPerPkg = 4

host.cpu.packages = 2

host.cpu.cores = 8

host.cpu.threads = 16

host.dmi.product = ProLiant ML350 G6

host.dmi.vendor = HP
```

`host.cpu.processorMHz` – nominal processor speed. Other metrics, such as `vm.cpu.used` below, are normalized to this speed.

`host.cpu.coresPerPkg` – actual cores per socket, not including hyperthreads. Useful for determining cache effects and other aspects of socket sharing. Information is also available with CPUID instruction. Note that vSphere does not implement virtual hyperthreads.

`host.cpu.packages` – number of CPU sockets on the host (non-default).

`host.cpu.cores` – number of cores on the host across all sockets, not including hyperthreads (non-default).

`host.cpu.threads` – number of logical CPUs on the host across all sockets, including hyperthreads (non-default).

`host.dmi.product` – "product" field in the host SMBIOS data (non-default).

`host.dmi.vendor` – "vendor" field in the host SMBIOS data (non-default).

Host information (**`dmi.product`** and **`dmi.vendor`**) and total capacity are hidden by default, because this information is considered sensitive and not relevant to virtual machine execution. VMware discourages use of such information, but permits it to be made available to help with support.

CPU and Memory Statistics

For implementation reasons, a virtual machine tracks CPU and memory resources slightly differently. CPU resources, including NUMA, indicate virtualization overhead, shown with **`vm.`** prefix. Memory resources are broken out by guest memory, shown with **`guest.`** prefix, and by overhead memory, with **`ovhd.`** prefix. Future implementations may add additional metrics.

This example shows various CPU and memory statistics:

```
$ vmware-toolbox-cmd stat raw text resources
```

```
vm.cpu.reserved = 4798

vm.cpu.limit = 11995

vm.cpu.used = 224057517

vm.cpu.contention.cpu = 65606184

vm.cpu.contention.mem = 1488848

vm.numa.local = 1837248

vm.numa.remote = 0

guest.mem.reserved = 204800

guest.mem.limit = 1536000

guest.mem.mapped = 1810144

guest.mem.consumed = 1521680

guest.mem.swapped = 3236

guest.mem.ballooned = 27104

guest.mem.swapIn = 3416

guest.mem.swapOut = 6588

ovhd.mem.swapped = 0

ovhd.mem.swapIn = 0

ovhd.mem.swapOut = 0
```

`vm.cpu.reserved` – (static) MHz of current CPU type reserved. Covers all virtual CPU plus overheads, so for example a 2 virtual CPU machine would need 2x **`host.cpu.processorMHz`** to be fully reserved. Overheads are insignificant except during transient conditions such as taking a backup snapshot or during a vMotion. Default 0.

`vm.cpu.limit` – (static) MHz that the virtual machine will not exceed. Default –1 means unlimited.

`vm.cpu.used` – (cumulative) microseconds of CPU time used by this virtual machine. Equivalent to **`esxtop %USED`**. See Comparison to esxtop for details.

`vm.cpu.contention.cpu` = (cumulative) CPU time the virtual machine could have run, but did not run due to CPU contention. This metric includes time losses due to hypervisor factors, such as overcommit. Specific sources of contention vary widely from release to release. See Comparison to esxtop for details about calculating CPU contention.

`vm.cpu.contention.mem` – (cumulative) CPU time the virtual machine could have run, but did not run due to memory contention. This metric includes losses due to swapping. Equivalent to **`esxtop %SWPWT`**.

`vm.numa.local` – (instantaneous) KB of memory currently local, sum across the VM's NUMA nodes.

`vm.numa.remote` – (instantaneous) KB of memory currently remote, sum across the VM's NUMA nodes.

`guest.mem.reserved` – (static) KB of memory reserved for the guest OS. This indicates memory that will never be ballooned or swapped. Default is 0.

`guest.mem.limit` – (static) KB of memory the guest must operate within. Default –1 means unlimited.

guest.mem.mapped – (instantaneous) KB of memory currently mapped into the guest; that is, memory the guest can access with zero read latency. This metric represents memory use from a guest perspective.

guest.mem.consumed – (instantaneous) KB of memory used to provide current mapped memory. This might be lower than mapped due to ballooning, memory sharing, or future optimizations. This metric represents memory use from a host perspective. The difference between **guest.mem.mapped** and **guest.mem.consumed** is additional memory made available due to hypervisor optimizations.

guest.mem.swapped – (instantaneous) KB of memory swapped to disk. A fully reserved virtual machine should never see memory swapped out in steady-state usage. Transient conditions, such as resume from memory-included snapshot, might show some swap usage.

guest.mem.ballooned – (instantaneous) KB of memory deliberately copied on write (COWed) to zero in the guest OS, to reduce memory usage.

guest.mem.swapIn – (cumulative) KB of memory swapped in for the current session.

guest.mem.swapOut – (cumulative) KB of memory swapped out for the current session.

ovhd.mem.swapped – (instantaneous) KB of overhead memory currently swapped.

ovhd.mem.swapIn – (cumulative) KB of overhead memory swapped in for the current session.

ovhd.mem.swapOut – (cumulative) KB of overhead memory swapped out for the current session.

Expected values for some of the statistics:

vm.cpu.contention.mem – usually < 1%, anything greater indicates memory overcommit.

vm.cpu.contention.cpu – < 5% of incremental time during undercommit, < 50% of incremental time at normal levels of overcommit (vSphere is tuned to perform best when somewhat overcommitted).

When contention is < 5%, performance will be deterministic but the host is not fully used.

When contention is between 5% and 50%, the host is becoming fully used (maximum CPU throughput) but individual virtual machines might see less deterministic performance.

vm.numa.local – Expected to match **guest.memory.mapped**. Transient conditions such as NUMA rebalance can cause this to temporarily decrease, then return to normal as memory is migrated.

vm.numa.remote – Expected to be approximately zero in non-overcommitted scenarios.

guest.mem.mapped – Expected to equal configured guest memory; might be smaller if virtual machine has yet to access all its memory.

guest.mem.consumed – Expected to be approximately equal to configured guest memory; will be smaller if host memory is overcommitted.

guest.mem.swapped – Expected to be zero. Non-zero indicates non-graceful memory overcommit.

guest.mem.ballooned – Expected to be zero. Non-zero indicates graceful memory overcommit.

ovhd.mem.swapped – Expected to be zero. Non-zero indicates memory overcommit.

Equations for CPU and memory metrics:

session **uptime** = `vm.cpu.used` + `vm.cpu.contention.cpu` + `vm.cpu.contention.mem` + CPU idle time

configured memory size = **guest.mem.mapped** + **guest.mem.swapped** + (memory not yet touched)

configured memory size = **vm.numa.local** + **vm.numa.remote** (another formula for arriving at the same statistic above)

guest.mem.mapped = **guest.mem.consumed** + **guest.mem.ballooned** + (other copy-on-write sources)

Comparison to esxtop

Individual reasons for lack of vCPU progress are available to vSphere administrators (using either **esxtop** or the vSphere API) but are hidden from the guest OS to preserve isolation between the virtual machine and the configuration of the infrastructure it runs upon. The guest OS sees only an aggregate metric.

`vm.cpu.used` is equivalent to the **esxtop** statistic %USED for a virtual machine.

`vm.cpu.contention.cpu` is equivalent to (%RDY – %MLMTD) + %MLMTD + %CSTP + %WAIT + (%RUN – %USED)

(%RDY – %MLMTD) represents time the guest OS could not run due to host CPU overutilization. Note that %RDY includes %MLMTD, which is why it is subtracted before being added.

%MLMTD represents time the guest OS did not run due to administrator-configured resource limits. ESXi 6.0 and earlier did not add %MLMTD to this computation, but this is fixed in ESXi 6.5.

%CSTP represents time the guest OS could not run due to uneven vCPU progress.

%WAIT represents time the guest OS could not run due to hypervisor overheads.

(%RUN – %USED) corrects for any frequency scaling of the host CPU.

`vm.cpu.contention.mem` is equivalent to %SWPWT.

See https://communities.vmware.com/docs/DOC-9279 for details about **esxtop**.

Note on nominal CPU speed and CPU metrics

The **host.cpu.processorMHz** metric (in the host section) reports a nominal speed, and the virtual machine CPU metrics are normalized to the **processorMHz** metric. Actual processor speed might be higher or lower depending on host power management.

A virtual machine can see `vm.cpu.used` exceed wall clock time due to Turbo Boost, or can see `vm.cpu.used` lag wall clock time due to power saving modes used in conjunction with idle guests. Actual processor speed is not available to the guest OS, but is expected to be close to nominal clock speed when the guest OS is active. See http://www.vmware.com/files/pdf/techpaper/hpm-perf-vsphere55.pdf for more information about vSphere host power management.

Normalizing CPU metrics to nominal CPU speed allows the guest OS to avoid dependence on host power management settings.

Note on vm.cpu.contention.cpu

Using the Extended Guest Statistics discussed in this section, you can obtain a contention ratio by comparing contention time to actual time for a particular time interval. As contention time is reported as a sum across VCPUs, and wall time is reported for the entire virtual machine, the wall time must be scaled up by the number of VCPUs to normalize contention to a 0-100% range.

```
Contention% = 100 * (contention_T2 - contention_T1) / (VCPUs * (time_T2 - time_T1))
```

The `vm.cpu.contention.cpu` metric is similar to “stolen time” returned by **`VMGuestLib_GetCpuStolenMs`** (see Table : Accessor Functions for Virtual Machine Data), except “stolen time” excludes time the virtual machine did not run due to configured resource limits. Comparing this value to **`esxtop`** requires denormalizing the contention ratio, because **`esxtop`** reports a sum of percentages across VCPUs. So:

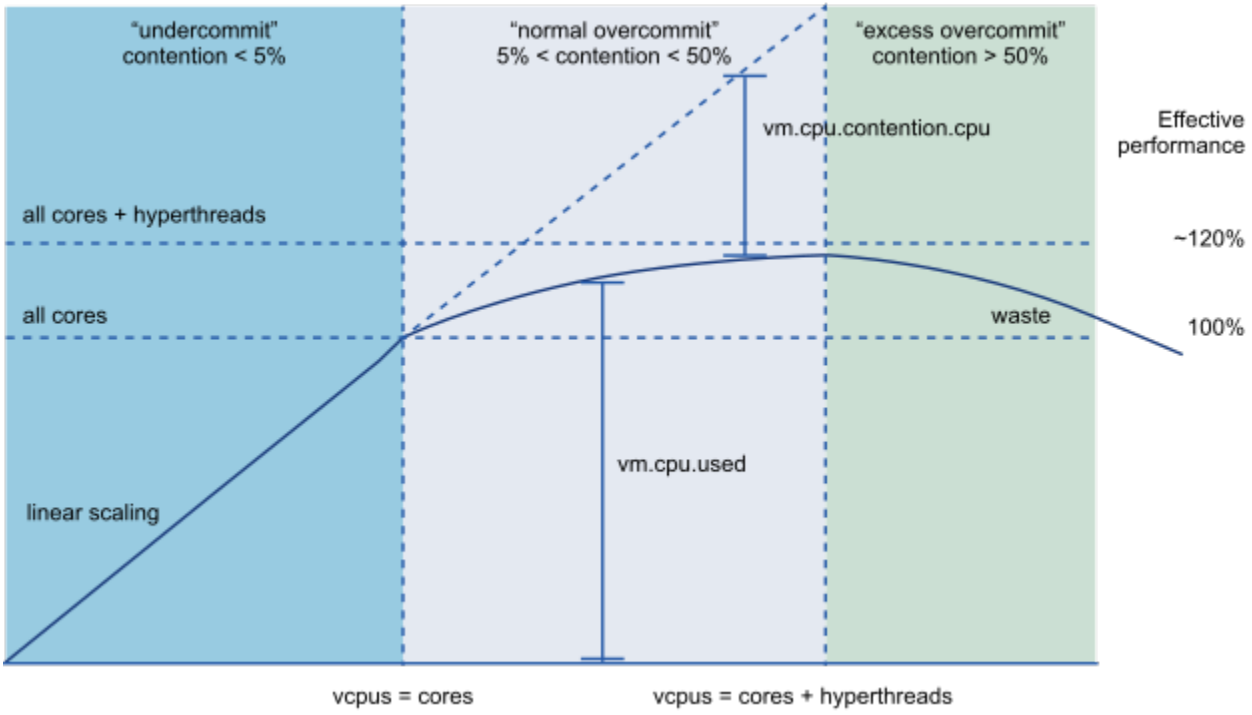
((%RDY – %MLMTD) + %MLMTD + %CSTP + %WAIT + (%RUN – %USED)) ~= *`Contention% * VCPUs`*

Due to sample aliasing where in-guest time samples and **`esxtop`** time samples do not occur simultaneously, instantaneous **`esxtop`** values will not match instantaneous in-guest statistics. Longer time samples or averaging values collected over time will produce more comparable results.

A contention value of < 5% is normal “undercommit” operating behavior, representing minor hypervisor overheads. A contention value > 50% is “excess overcommit” and indicates CPU resource starvation – the workload would benefit from additional CPUs or migrating virtual machines to different hosts. A contention value between 5% and 50% is “normal overcommit” and is more complicated. The goal of this metric is to allow direct measurement of the performance improvement that can be obtained by adding CPU resources.

The following figure illustrates these concepts.

Figure 163: CPU use across virtual machines



VMware best practices describe the available CPU capacity of an ESXi host as equal to the number of cores (not including hyperthreads). A 16 core host with 2.0GHz processors has 16 cores * 2000 MHz/core = 32000 MHz available compute capacity. When actual usage is below that calculated capacity, the hypervisor is considered “under committed” – the hypervisor is scaling linearly with load applied, and is wasting capacity.

As actual usage exceeds available compute capacity, the hypervisor begins utilizing hyperthreads for running virtual machines to keep performance degradation graceful. Maximum aggregate utilization occurs during this “normal overcommit” (between 5% and 50% contention) where each virtual machine sees somewhat degraded performance but overall system throughput still increases. In this “normal overcommit” region, adding load still improves overall efficiency, though at a declining rate. Eventually, all hyperthreads are fully used. Efficiency peaks and starts to degrade; this “excess

overcommit" (>50% contention) indicates the workload would be more efficient if spread across more hosts for better throughput.

One specific scenario deserves special mention: the "monster VM" that attempts to give a single VM all available compute capacity. A VM configured to match the number of host cores (not including hyperthreads) will peak at the capacity of those cores (with < 5% contention) but at a performance about 20% lower than an equivalent physical machine utilizing all cores and hyperthreads. A VM configured to match the number of host threads (2x host cores) will peak at a performance level more analogous to a physical machine, but will show about 40% contention (the upper end of "normal overcommit") running half the cores on hyperthreads. This contention metric indicates the load would run better on a larger host with additional cores, so it is technically "overcommitted" even though performance is better than a hypervisor running at full commit. This behavior is expected when attempting to run maximally sized virtual machines.

Storage Statistics

The following example shows some I/O statistics:

```
$ vmware-toolbox-cmd stat raw text vscsi scsi0:0

num.reads = 12605

num.writes = 1039

size.reads = 533612032

size.writes = 14279680

latency.reads = 1944173239

latency.writes = 102025122
```

`num.reads` – read commands.

`num.writes` – write commands.

`size.reads` – bytes read.

`size.writes` – bytes written.

`latency.reads` – microseconds of all read commands.

`latency.writes` – microseconds of all write commands.

The read IOPs over the last 10 seconds can be calculated as:

$$\frac{(\textbf{num.reads} @ \mathit{TimeNow}) - (\textbf{num.reads} @ \mathit{Time10sec})}{\mathit{TimeNow} - \mathit{Time10sec}}$$

The average latency of reads over the last ten seconds can be calculated as:

$$\frac{(\textbf{latency.reads} @ \mathit{TimeNow}) - (\textbf{latency.reads} @ \mathit{Time10sec})}{(\textbf{num.reads} @ \mathit{TimeNow}) - (\textbf{num.reads} @ \mathit{Time10sec})}$$

Network Statistics

Reservation and limit are supported on DVS (Distributed Virtual Switch) or "opaque" (NSX) switch types only; they are not supported on the default VSS switch type. Between **`reservation`** and **`limit`**, bandwidth is allocated on a share-based system, which is not meaningful to expose to a guest OS.

```
$ vmware-toolbox-cmd stat raw text vnet 00:0c:29:1e:23:f3

num.tx = 209

size.tx = 38137

num.rx = 10920

size.rx = 1312789

reservation = 0

limit = -1
```

num.tx – number of packets transmitted.

size.tx – bytes transmitted.

num.rx – number of packets received.

size.rx – bytes received.

reservation – guaranteed minimum bandwidth for this vNIC.

limit – maximum bandwidth allowed for this vNIC.

DataSets for Guest Management

The DataSets facility provides methods to manipulate VM-related key-value entries using vSphere Automation from the vCenter Server, or from inside a guest OS with VM Tools.

About the DataSets Service

The DataSets facility can enable a variety of applications in which vSphere and a guest OS share data.

The DataSets facility is a way for vCenter Server to store and update persistent information in a virtual machine, where the guest operating system can access it. As part of vSphere Automation APIs, the service provides operations to manipulate key-value entries in a VM, which can then be used by applications and agents inside the guest OS. The guest OS can modify some but not all of this information.

> **NOTE**
> DataSets functionality is available in vSphere 8.0 and later, and requires the VM to be running virtual hardware version 20.

Whether or not the guest OS is running, the vSphere management layer can modify DataSets. Persistent information in DataSets may be any key-value data. Typical uses might be such things as guest deployment status, guest agent configuration, guest related metadata, or other items. The service is intended for information that is not expected to change frequently, and never grows beyond a small number of megabytes (MB).

You can list all the DataSets in a VM. Specifying a DataSet name and other information, you can create, read, update, or delete a DataSet.

You can create key-value pairs within a DataSet, and thereafter list the keys, update their values, or delete key-value pairs. Keys must be no larger than 4KB, and values no larger than 1MB.

Interoperation with VM actions is supported, including migration, cloning, and snapshots.

DataSets should be modified only by the application that creates them. Each application using the service should store its data in at least one unique DataSet, to avoid conflict with other applications. DataSets are visible to other applications, both on the management network and within the guest OS. Therefore sensitive data such as passwords and private keys should not be stored as plain-text.

All input and output data are represented in JSON. When you write DataSet programs in Python, Java, or another such language, a library layer takes care of JSON data communication.

Use Cases for DataSets

As a method to share data between vSphere and a guest OS, the DataSets facility replaces internal mechanisms that VMware used previously.

Here are some use cases for DataSets.

Guest deployment status. Suppose a customer wants to perform guest provisioning that includes running deployment scripts. DataSets can present configuration data and deployment scripts to the guest. The guest can then report back status including possible errors.

Guest agent configuration. Suppose a management application needs to configure an in-guest agent. The management side creates and populates the DataSet with configuration data, and the guest agent reads and responds to that data.

Control Through vCenter Server

Programs can control DataSets through a vCenter Server using the vSphere Automation API (vAPI).

Methods are provided for managing DataSets, and for managing entries in a DataSet. These methods can be run as REST operations, or from Java or Python programs. See developer.vmware.com for complete information about the various vSphere Automation SDKs. Ruby, Perl, and .NET bindings were deprecated but are still available as open source on Github.

DataSets have certain properties, called attributes. Attributes include a description of the DataSet, access permissions from the guest or host, and usage type. Attributes are assigned to each DataSet at create time, and can be modified by an update.

> **NOTE**
> In the following REST examples, `{server}` represents the vCenter Server that manages ESXi hosts running VMs in a datacenter.

Operations on a DataSet:

- create – create a new DataSet with specified attributes
  `POST https://{server}/api/vcenter/vm/{vm}/data-sets`
- delete - delete a DataSet
  `DELETE https://{server}/api/vcenter/vm/{vm}/data-sets/{DataSet}`
- update – modify the attributes of a DataSet
  `PATCH https://{server}/api/vcenter/vm/{vm}/data-sets/{DataSet}`
- get – query the attributes of a DataSet
  `GET https://{server}/api/vcenter/vm/{vm}/data-sets/{DataSet}`
- list - list all DataSets in a VM
  `GET  https://{server}/api/vcenter/vm/{vm}/data-sets`

Operations on DataSet entries:

- set - create or update an entry in a DataSet
  `PUT https://{server}/api/vcenter/vm/{vm}/data-sets/{DataSet}/entries/{key}`
- get - retrieve the value of an entry in a DataSet
  `GET https://{server}/api/vcenter/vm/{vm}/data-sets/{DataSet}/entries/{key}`
- delete - delete an entry in a DataSet
  `DELETE https://{server}/api/vcenter/vm/{vm}/data-sets/{DataSet}/entries/{key}`
- list - list all entry keys in a DataSet
  `GET https://{server}/api/vcenter/vm/{vm}/data-sets/{DataSet}/entries`

These REST operations communicate back and forth in JSON, similar to the examples shown in the sections below.

For instruction and examples of REST programming, see the *`vSphere Automation API Reference`* on code.vmware.com.

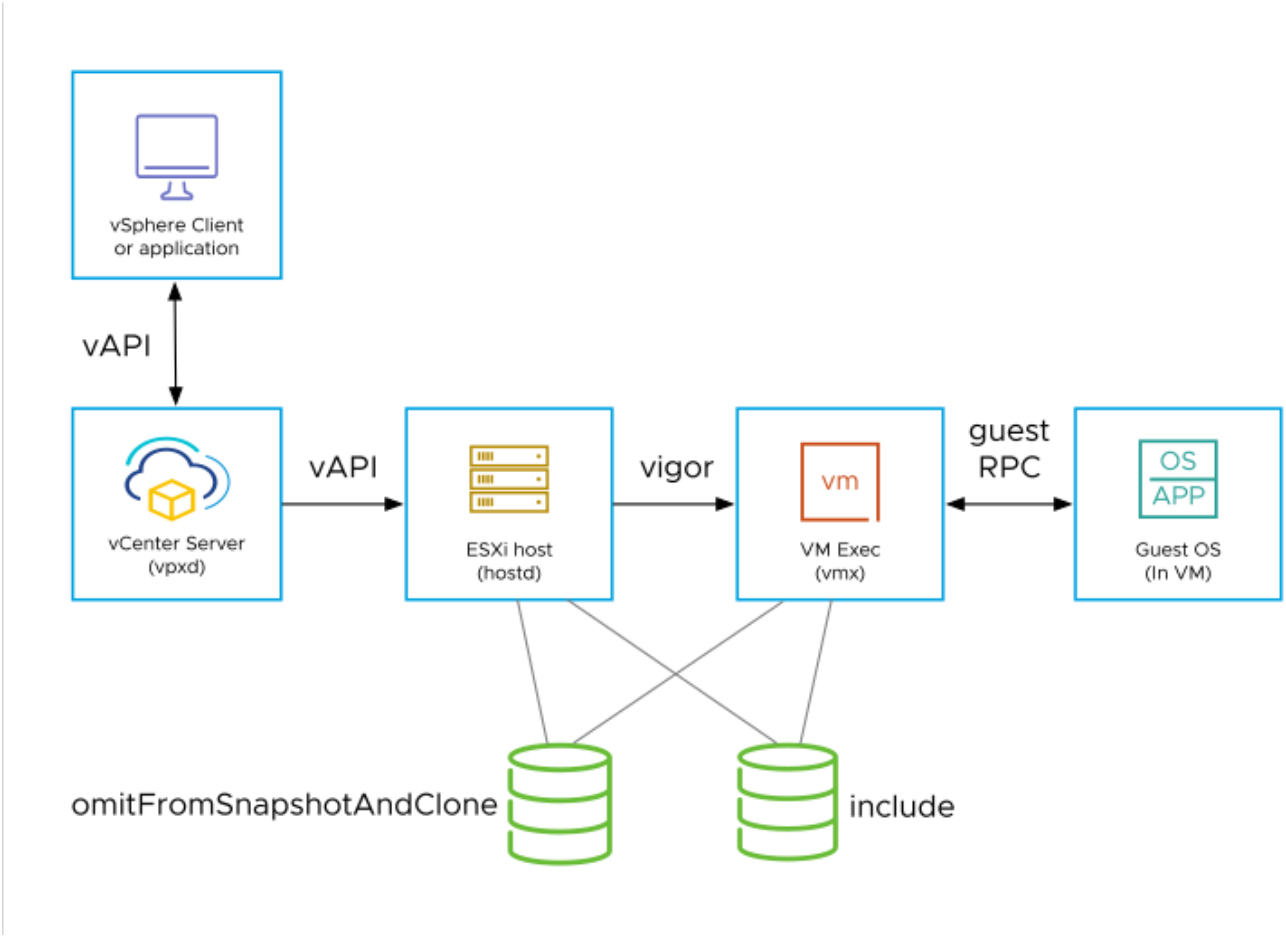
DataSets Transfer to ESXi Host

When vSphere Automation identifies the ESXi host running the requested VM, it transmits control information as directed.

DataSets and DataSet entries are associated with a VM, rather than with a centralized vSphere service. This means DataSets are part of a VM object. Any VM snapshot, clone, vMotion, or similar operation performed on the VM will contain the DataSets and their entries.

By default, the attribute `omitFromSnapshotAndClone` is unset for each Dataset, but if set true for any Dataset, snapshot and clone will not preserve the DataSet or its entries.

Figure 164: DataSets Architecture



When vSphere Automation (vAPI) creates or updates a DataSet, the ESXi host accepts the assigned path to the DataSet store, and writes any data specified in the request. Vigor is an API for VMX interface generated object representation.

During VM power on, the ESXi host locates the DataSet file, opens the base store, initializes the VM, and updates its DataSet list.

During VM power off, the ESXi host detaches the DataSet and frees the base store.

Access from Guest OS

Once a DataSet is present in the VM, the guest OS can access and modify its DataSet entries using a small set of VMware Tools commands.

A guest OS cannot create, delete, or change the attributes of a DataSet. However, once one or more DataSets is present in the VM, the guest OS can obtain a list of DataSets and attributes, and read or write DataSet entries.

The guest OS has read access to DataSet key-value entries. Access from the guest OS does not allow DataSets themselves to be created or deleted, nor their attributes to be modified. The guest OS can modify key values and delete key-value pairs.

Users can access DataSet information using the terminal (CLI) on Windows, Linux, or MacOS using a utility that is part of VMware Tools. In the **`vmtoolsd`** template below, you replace *`directive`* with one of the DataSet commands below. The `--cmd` option is implemented as a message from the guest to the host.

```
vmtoolsd.exe --cmd "directive"

vmtoolsd --cmd "directive"
```

For example, the following command lists all keys from the DataSet named my-dataset.

```
$ vmtoolsd --cmd 'datasets-list-keys {"dataset":"my-dataset"}'
```

Each DataSet command produces JSON output, which on failure will include an error message. JSON schemas are provided in the next section. For easy editing and reuse, JSON can be placed into a file, as described at the end of this section. DataSet commands are presented below, each followed by sample input and results.

datasets-list – Return summary data about any DataSets, such as name and description.

```
datasets-list
```

No input required. Sample JSON results on success:

```
{
  "result" : true,
  "datasets" : [
    { "name":"dataset1", "description":"description of dataset1" },
    { "name":"dataset2", "description":"description of dataset2" },
    { "name":"dataset3", "description":"description of dataset3" }
  ]
}
```

datasets-query – Return a DataSet's attributes (description, usage, access).

```
datasets-query <json-document>
```

Here is an example of `<json-document>` that might be used to pass directives in the above command:

```
{
  "dataset": "dsname1"
}
```

Here is the same JSON used in a **`vmtoolsd`** command line:

```
vmtoolsd --cmd 'datasets-query {"dataset": "dsname1"}'
```

Sample JSON results; error message only on failure:

```
{
  "result": true,
  "info": {
    "name": "dsname1",
    "description" : "dsname1 description",
    "used" : 2345,
```

```
    "hostAccess" : "READ_WRITE",
    "guestAccess" : "READ_ONLY",
    "omitFromSnapshotAndClone" : true },
  "error-message": ""
}
```

Sample JSON results on error:

```
{
  "error-message" : "DataSet dsname2 not found."
}
```

datasets-get-entry – Get the value of one or more keys in a DataSet. If any key is not defined, it's not an error, but no results are returned for undefined keys.

```
datasets-get-entry <json-document>
```

Here is an example of `<json-document>` that might be used to pass directives the above command:

```
{
  "dataset" : "ds1",
  "keys" : [ "key1", "key2", "key3" ]
}
```

Here is an example of results passed back from the command, in JSON format:

```
{
  "result" : true,
  "entries" : [
      { "key1" : "value1"},
      { "key2" : "value2"},
      { "key3" : "value3"}
   ]
}
```

datasets-set-entry – Set the value of one or more keys in a DataSet. An individual key can be specified only once, otherwise the operation fails with an error.

```
datasets-set-entry <json-document>
```

In this example below, `key1` will be set or reset to `value1`, while `key2` will have `appended-value2` added after its previous value, which may be null. If there is no entry with "key2" key, then "append" is essentially ignored – a new entry will be created with "key2" key and "appended-value2" value. Append is useful when you want to set a value that is too large for the underlying message transport to handle, so more than one datasets-set-entry is required. You must append

in chunks that fit within the message limit, around 63K. Here is how <json-document> might be used to pass those directives to datasets-set-entry:

```
{
  "dataset" : "ds1",
  "entries": [
    { "key": "key1", "value": "value1" },
    { "key": "key2", "value": "appended-value2", "append": true }
  ]
}
```

Returns error or success; error message only on failure. Sample JSON results:

```
{
  "result": true
}
```

datasets-list-keys – Enumerate the keys in a DataSet.

```
datasets-list-keys <json-document>
```

Here is an example of <json-document> that might be used to pass directives the above command:

```
{
  "dataset": "ds1"
}
```

Sample JSON results; error message only on failure.

```
{
  "result": true, "error-message": "",
  "keys" : [ "key1", "key2", "key3" ]
}
```

datasets-delete-entry – deletes one or more entries in a DataSet. If any key is not defined, the operation will fail and the DataSet contents will be unchanged.

```
datasets-delete-entry <json-document>
```

Here is an example of <json-document> that might be used to pass directives the above command:

```
{
  "dataset" : "ds1",
  "keys": [ "key1", "key2", "key3" ]
}
```

Returns error or success; error message only on failure. Sample JSON results:

```
{
  "result": true, "error-message": ""
}
```

JSON in a File

VMware Tools 11.2 added the **`--cmdfile`** option to the **`vmtoolsd`** command. This option allows *`directive`* to be specified in a file rather than on the command line, as in previous examples. Due to the 8KB limit for shell commands on Windows, and the vagaries of escaping shell syntax on MacOS and Linux, the command file option is recommended.

```
$ vmtoolsd --cmdfile commandFile.sh
```

Here is what might be in `commandFile.sh` for the **`datasets-get-entry`** command at the beginning of this section.

```
datasets-get-entry {"dataset": "ds1", "keys": ["key1", "key2", "key3"]}
```

Python for DataSets

The Guest SDK contains Python sample code that provides an easy-to-use wrapper for DataSets operations. Functions in the `util.py` file show how to set up Guest RPC and pass JSON objects. The `datasets.py` program implements all of the commands, as shown in its usage message. See sections above for required arguments.

```
Usage: datasets [query|list|listkeys|set|get|delete] arguments
```

DataSets Schema for Guest Access

This section shows JSON schema for the Datasets interfaces. For each DataSets command, input and output schema appear, followed by possible errors.

datasets-list – Return summary data about any DataSets, such as name and description.

```
        no input schema
```

Output schema:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "result": {
      "type": "boolean",
          "description" : "A boolean specifying whether the RPC was successful."
    },
    "error-id": {
      "type": "string",
```

```
          "description": "If set, an error identifier for why the RPC failed."
    },
    "error-message": {
      "type": "string",
          "description": "If set, a usage message and error details for why the RPC
failed."
    },
    "datasets": {
      "type": "array",
          "description": "The list of datasets and descriptions.",
          "examples": [
                    { "name": "dataset1", "description": "Description of dataset1" },
                    { "name": "dataset2", "description": "Description of dataset2" }
                  ],
       "items":
        {
          "type": "object",
          "properties": {
            "name": {
              "type": "string"
            },
            "description": {
              "type": "string"
            }
          },
          "required": [
            "name",
            "description"
          ]
        },
    }
  },
  "required": [
```

```
    "result",
    "datasets"
  ]
}
```

Possible error returned:

- `invalid-argument` if there are any extraneous parameters.

datasets-query – Return a DataSet's attributes (description, usage, access).

Input schema:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "context": {
      "type": "string",
      "description": "The context value from a previous RPC with the same arguments to get the next page of output."
    },
   "dataset": {
      "type": "string",
      "description": "The name of the dataset from which to list the keys."
    }
  },
  "required": [
    "dataset"
  ]
}
```

Output schema:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "result": {
      "type": "boolean",
```

```
          "description" : "A boolean specifying whether the RPC was successful."
    },
    "error-id": {
      "type": "string",
          "description": "If set, an error identifier for why the RPC failed."
    },
    "error-message": {
      "type": "string",
          "description": "If set, a human-readable usage message and error details for why
the RPC failed."
    },
    "info": {
      "type": "object",
      "description": "The attributes of the DataSet.",
      "examples": [
        { "name": "dsname1", "description" : "dsname1 description", "size": 12345,
"used" : 2345, "hostAccess" : "READ_WRITE", "guestAccess" : "READ_ONLY",
"omitFromSnapshotAndClone" : false }
         ],


     "items":   [
       {
       "type": "object",
          "properties": {
            "name": {
              "type": "string",
              "description": "The name of the DataSet."
            },
            "description": {
              "type": "string",
              "description" : "The description of the DataSet."
            },
            "used": {
              "type": "number",
```

```
              "description": "The number of bytes in use by the DataSet."
            },
            "hostAccess" : {
               "type": "string",
               "description" : "The access from the host."
            },
            "guestAccess" : {
               "type": "string",
                "description" : "The access from the guest."
            }
            "omitFromSnapshotAndClone": {
                "type": "boolean",
                "description": "If true, the data set is considered a property of the
virtual machine, and is not included in a snapshot operation or when the virtual machine
is cloned."
            }
          },
          "required": [
            "name",
            "description",
            "used",
            "hostAccess",
            "guestAccess",
            "omitFromSnapshotAndClone"
          ]
        },
      ],
    },
  },
  "required": [
    "result",
    "info"
  ]
```

```
}
```

Possible errors returned:

- `invalid-argument` if the specified JSON is invalid or arguments are missing or malformed.
- `dataset-not-found` if the supplied dataset does not exist.

datasets-get-entry – Get the value of one or more keys in a DataSet.

Input schema:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "dataset": {
      "type": "string",
      "description": "DataSet from which to get entries."
    },
    "context": {
       "type": "string",
       "description": "The context value from a previous RPC with the same arguments to
get the next page of output."
    },
    "keys": {
      "type": "array",
      "description": "Entries to get.",
      "items": [
        {
          "type": "string"
          "description": "The key being requested."
        },
      ]
    }
  },
  "required": [
    "dataset",
    "keys"
```

```
  ]
}
```

Output schema:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "result": {
      "type": "string"
    },
    "error-id": {
      "type": "string",
          "description": "If set, an error identifier for why the RPC failed."
    },
    "error-message": {
      "type": "string",
      "description": "If set, a human-readable usage message and error details for why the
RPC failed."
    },
    "context": {
      "type": "string",
      "description": "If set, a value that can be passed in to the RPC with the same
arguments to get the next page of output."
    },
    "entries": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "key": {
              "type": "string",
              "description": "The queried key and value."
            },
```

```
          },
          "required": [
            "key"
          ]
        },
      ]
    }
  },
  "required": [
    "result",
    "entries"
  ]
}
```

Possible errors returned:

- `invalid-argument` if the specified JSON is invalid or arguments are missing or malformed.
- `dataset-not-found` if the specified dataset does not exist.
- `invalid-context` if the specified context is invalid.

datasets-set-entry – Set the value of one or more keys in a DataSet.

Input schema:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "dataset": {
      "type": "string",
      "description": "DataSet in which to set the entries."
    },
    "entries": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
```

```
            "key": {
              "type": "string",
              "description": "The key being set."
            },
            "value": {
              "type": "string",
              "description": "The new value for the key."
            },
            "append": {
              "type": "boolean",
              "description": "If set, the value is appended to any current value, or set
if this is a new key."
            },
          },
          "required": [
              "key",
              "value",
          ]
        },
      ],
    },
  },
  "required": [
      "dataset",
      "entries"
   ]
}
```

Output schema:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
```

```
    "result": {
      "type": "boolean",
      "description" : "A boolean specifying whether the RPC was successful."
    },
    "error-id": {
      "type": "string",
          "description": "If set, an error identifier for why the RPC failed."
    },
    "error-message": {
      "type": "string",
      "description": "If set, a human-readable usage message and error details for why the
RPC failed."
    },
  },
  "required": [
    "result"
  ]
}
```

Possible errors returned:

- `invalid-argument` if the specified JSON is invalid or arguments are missing or malformed.
- `dataset-not-found` if the specified dataset does not exist.
- `out-of-space` if the total data exceeds the maximum allowed.

datasets-list-keys – Enumerate the keys in a DataSet.

Input schema:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "context": {
      "type": "string",
      "description": "The context value from a previous RPC with the same arguments to get
the next page of output."
    },
   "dataset": {
```

```
      "type": "string",
      "description": "The name of the dataset from which to list the keys."
    }
  },
  "required": [
    "dataset"
  ]
}
```

Output schema.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "result": {
      "type": "boolean",
      "description" : "A boolean specifying whether the RPC was successful."
    },
    "error-id": {
      "type": "string",
          "description": "If set, an error identifier for why the RPC failed."
    },
    "error-message": {
      "type": "string",
      "description": "If set, a human-readable usage message and error details for why the
RPC failed."
    },
    "context": {
      "type": "string",
      "description": "If set, a value that can be passed in to the RPC with the same
arguments to get the next page of output."
    },
    "keys": {
      "type": "array",
```

```
      "description": "The list of keys in the dataset.",
      "examples": [ "key1", "key2", "key3" ]
    }
  },
  "required": [
    "result",
    "keys"
  ]
}
```

Possible errors returned:

- `invalid-argument` if the specified JSON is invalid or arguments are missing or malformed.
- `dataset-not-found` if the specified dataset does not exist.
- `invalid-context` if the specified context is invalid.

datasets-delete-entry – deletes one or more entries in a DataSet.

Input schema:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "dataset": {
      "type": "string",
      "description": "DataSet from which to delete entries."
    },
    "keys": {
      "type": "array",
      "description": "Entries to delete.",
      "items": [
        {
          "type": "string"
        },
      ]
    }
  },
```

```
  "required": [
    "dataset",
    "keys"
  ]
}
```

Output schema:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "result": {
      "type": "boolean",
      "description" : "A boolean specifying whether the RPC was successful."
    },
    "error-id": {
      "type": "string",
          "description": "If set, an error identifier for why the RPC failed."
    },
    "error-message": {
      "type": "string",
      "description": "If set, a human-readable usage message and error details for why the
RPC failed."
    },
  },
  "required": [
    "result"
  ]
}
```

Possible errors returned:

- `invalid-argument` if the specified JSON is invalid or arguments are missing or malformed.
- **`dataset-not-found`** if the specified dataset does not exist.
- `key-not-found` if the specified key does not exist.

Table of Error Codes

Typedef enumeration	Generic error message	Description
`ERROR_SUCCESS`	*`JSON response returned`*.	Not an error.
`ERROR_DATASET_NOT_FOUND`	DataSet *`ds-name`* not found.	DataSet not yet created.
`ERROR_KEY_NOT_FOUND`	Key `k-name` not found.	Attribute key not yet created.
`ERROR_INVALID_ARGUMENT`	Invalid argument: *`arg`*.	Malformed JSON or arguments.
`ERROR_INVALID_CONTEXT`	Invalid context: *`context`*.	Probably key does not exist in DataSet.
`ERROR_ACCESS_DENIED`	Access denied to DataSet *`ds-name`*.	Not authorized for this operation.
`ERROR_OUT_OF_SPACE`	Insufficient resources: *`datum`*.	Message passed was too large.
`ERROR_GENERIC`	Error: *`error-message`*.	Some other error.

In-Guest Communication Limits

Resource constraints impose unavoidable limits on guest to host communications.

Messages from the guest to VMX can be no more than 64KB, including overhead. The VMX sends back at most about 1MB. If the 64KB limit is exceeded, the request fails and an error is returned. This affects several in-guest APIs.

The guest-to-VMX limit affects **`datasets-set-entries`**, which is why the **`append`** flag exists. If you have a value >63K that must be set, then you should break it into chunks. The first chunk should have the first 63K or so bytes of **`value`**, with the **`append`** flag either omitted or false. Subsequent chunks should call **`datasets-set-entries`** again with the same key, the next chunk of the value, and **`append`** flag set true. Iterate until the entire large value has been set.

The VMX-to-guest limit affects **`datasets-list-keys`** and **`datasets-get-entries`**.

If the full reply is >1MB or so, it includes a **`context`** property. If so, then the next chunk of data for that reply will be returned, after another call with the same parameters is sent, plus the **`context`** returned by the previous reply. For both **`datasets-list-keys`** and **`datasets-get-entries`**, any chunking is done at the key level – the key and value are never separated. Confusion might ensue if data changes midway through the process, for example, if vSphere changes the keys or entries involved.

The **`context`** is an opaque string. Programs should never try to use anything but what was returned in the previous reply as its value.

The **`datasets-list-keys`** limit is unlikely to be encountered in most cases, because it only occurs with >1MB of **`keynames`** in a single dataset.

If programs query more than one entry in a single API call, then context can come into play, because the complete reply could be >1MB (any value could be up to 1MB). But this does mean you can get a mix of reply sizes. A **`datasets-get-entries`** of one set of keys that lists a small key, a large key, and another small key might result in three replies. The first will include just value1, the second value2 and the third value3.

If you want to circumvent these limits, or have no idea how big the values might be, you can get a single entry at a time. That is less efficient but may be advisable.

vSphere HA Application Monitoring

This chapter discusses vSphere High Availability (HA) and how to use HA Application Monitoring APIs with the Guest SDK.

About vSphere HA

The vSphere High Availability (HA) feature for ESXi hosts in a cluster provides protection for a guest OS and its applications, by restarting the virtual machine if a guest OS or application failure occurs. The HA feature provides this reset capability using two different mechanisms:

- VM Monitoring – Guest OS heartbeats issued by the VMware Tools process.
- Application Monitoring – Heartbeats issued by a program that uses the HA Application Monitoring SDK to communicate with the VMware Tools process and the vSphere HA agent. This mechanism involves local monitoring by the program to avoid the overhead of sending messages to and from vCenter Server.

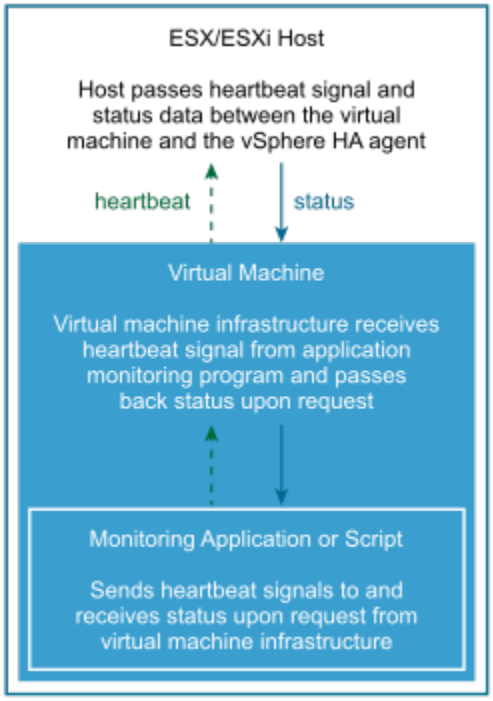
Additionally, the in-guest agent can set state to indicate it needs an immediate reset. This can be done without enabling heartbeats. The HA Application monitoring facility can reset the guest OS when ready to do so, if the in-guest agent has not changed state to say reset is no longer needed.

The application monitoring program sends an enable request to start the monitoring, possibly followed by a heartbeat signal. The vSphere infrastructure passes the signal up from your HA application monitoring program to the virtual machine, and then to the ESXi host. The HA application monitoring facility will reset the virtual machine if the application monitoring program stops sending a heartbeat signal, or requests a reset.

Using the HA Application Monitoring SDK, developers can write HA application monitoring programs in the C or C++ language. The HA Application Monitoring API is available with C/C++ language bindings only.

The following figure depicts the monitoring and reset capability of host and virtual machine.

Figure 165: Heartbeat and status signals



For more information about vSphere HA and application monitoring, see the *vSphere Availability* guide in the vSphere Documentation Center.

Prerequisites for HA Application Monitoring

Before you start working with the HA Application Monitoring SDK, verify that your vSphere application is running within a VMware cluster that has both the **High Availability** and **VM and Application Monitoring** options enabled.

You must install VMware Tools on the virtual machines where your HA monitoring applications are running.

The *vSphere Availability* guide contains information about how to set up a high availability (HA) cluster, and how to configure **VM and Application Monitoring**. With VMware's New Cluster Wizard, you can choose one of three monitoring options:

- **Disabled** – Neither VM Monitoring nor Application Monitoring.
- **VM Monitoring Only** – If you select this option, you will have the Guest OS monitoring discussed previously (the first mechanism).
- **VM and Application Monitoring** – If you select this option, you will also have the ability to employ Application Monitoring and the HA Application Monitoring SDK (the second mechanism).

For information about Web services interfaces for HA, see the *VMware vSphere API Reference Guide*, especially data objects **`VirtualMachineRuntimeInfo`** and **`VirtualMachineRuntimeInfoDasProtectionState`**.

Using the HA Application Monitoring APIs

You can use the HA Application Monitoring SDK to create a stand-alone application monitoring program, or to enhance an existing application or script. The purpose of your application monitoring program determines the API call sequence and the application behavior that you write to handle the response data.

For example, if your application monitoring program is tracking critical applications that are running in a guest OS, your application can intentionally stop sending heartbeat signals if any application-related process fails. The HA monitoring agent interprets the absence of heartbeats as a failure, and resets the virtual machine.

Alternatively, instead of not sending heartbeat signals, your application monitoring program can set the **`needReset`** flag using the **`VMGuestAppMonitor_PostAppState`** call. When the HA monitoring agent notices this flag, it will reset the virtual machine.

Most of the calls you make using the HA Application Monitoring APIs send information to the virtual infrastructure of the ESXi host, and the host relays the information to the HA monitoring agent. However, the **`VMGuestAppMonitor_GetAppStatus`** call is a two-way transaction that lets you request the virtual machine status from the HA monitoring agent.

Most HA Application Monitoring functions lack input parameters, because the calls are local. The vSphere infrastructure passes the heartbeat and status data to and from other levels of the cluster.

Call each function from your application monitoring program. The vSphere infrastructure (in the virtual machine where the application monitoring program is running) passes the function data up to the ESXi host. The local virtual machine sends all status responses to your application monitoring program, even though they are passed down from the HA monitoring agent.

HA Application Monitoring API Functions

The following calls are available to a vSphere HA application monitoring program:

Table 100: HA Application Monitoring API Calls

Call Name	Data Type Returned	Description
VMGuestAppMonitor_Enable	char	Requests the virtual machine infrastructure to monitor the calling application. The virtual machine infrastructure returns a value of VMGUESTAPPMONITORLIB_ERROR_SUCCESS, if monitoring was enabled.

Table continued on next page

Continued from previous page

Call Name	Data Type Returned	Description
		After your application monitoring program makes this call, your program must call VMGuestAppMonitor_MarkActive() at least once every 30 seconds or the virtual machine infrastructure will change the virtual machine's status to Red or Gray.
VMGuestAppMonitor_Disable	int	Requests the virtual machine infrastructure to stop monitoring the calling program. The virtual machine infrastructure returns a value of TRUE if monitoring was disabled.
VMGuestAppMonitor_IsEnabled	int	Returns the current recorded state of application monitoring. The virtual machine infrastructure returns a value of TRUE if monitoring is enabled.
VMGuestAppMonitor_MarkActive	char	Sends a request to mark the program as active. This function is also called the heartbeat because your program must call it at least once every 30 seconds while your application monitoring is enabled, or the virtual machine infrastructure will determine that the monitoring has failed.
VMGuestAppMonitor_PostAppState	int	Publish the application state that the guest OS wants delivered to vSphere HA. The application should monitor its environment and update its state accordingly. Heartbeat counting does not need to be enabled as a pre-condition, so the **`enable()`** call is not necessary. Returns 0 (zero) on success. The single **`state`** parameter passed to this call can be either: • OK – The guest's application agent declared state to be normal and no action is required. • **`needReset`** – The guest's application agent has requested an immediate reset. The guest can request this at any time.
VMGuestAppMonitor_GetAppStatus	char	Returns the current status recorded by the virtual machine infrastructure as 'Green', 'Red', or 'Gray'. • Green. Virtual machine infrastructure acknowledges that the application is being monitored. • Red. Virtual machine infrastructure does not think the application is being monitored. The HA monitoring agent will initialize an asynchronous reset on the virtual machine if the status is Red. • Gray. Application should send VMGuestAppMonitor_Enable again, followed by VMGuestAppMonitor_MarkActive, because either application monitoring failed, or the virtual machine was vMotioned to a different location. Call VMGuestAppMonitor_Free to free the result. If this call returns a nonerror result that was not anticipated, it can mean that another program in the same virtual machine has called VMGuestAppMonitor_Disable or VMGuestAppMonitor_Enable. If your application is still running, call VMGuestAppMonitor_Enable again, followed by calls to VMGuestAppMonitor_MarkActive.
VMGuestAppMonitor_Free	char	Returns a pointer to the memory to be freed.

Code Sample for appmon.cpp

The HA Application Monitoring SDK includes a code sample called **`appmon.cpp.`** The sample is located in the `docs/samples` directory and defines the entry point for the console application. The `appmon.cpp` program includes interface code that your application monitoring program can send after receiving results from calls to VMGuestAppMonitor_Enable, VMGuestAppMonitor_MarkActive, and VMGuestAppMonitor_Disable.

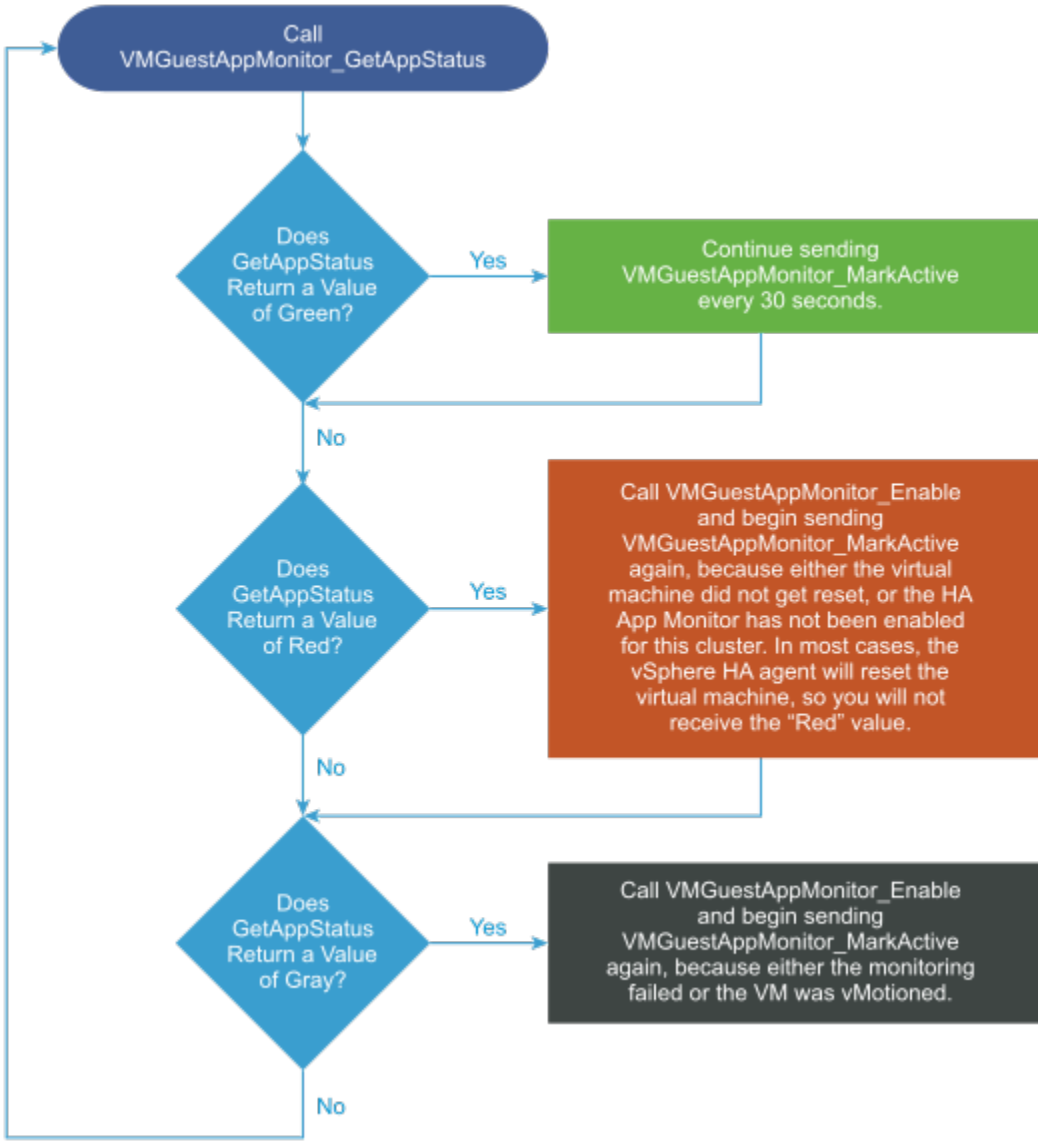
Calling the APIs from Your Application

The following steps provide a possible API sequence of calls:

1. Include **`vmGuestAppMonitorLib.h`** in the declarations for your C program.
2. To start the monitoring, notify the virtual machine that you are going to start sending a heartbeat signal by calling `VMGuestAppMonitor_Enable`.
3. After you enable VM guest app monitoring, call `VMGuestAppMonitor_MarkActive` every 30 seconds to prevent reset of the virtual machine.
4. Send `VMGuestAppMonitor_IsEnabled` to make sure the virtual machine infrastructure received your requests correctly and has begun monitoring.
5. Periodically, call `VMGuestAppMonitor_GetAppStatus` to make sure the vSphere infrastructure is still receiving the heartbeat calls.

   The status will be returned as *`Green`*, *`Red`*, or *`Gray`*. See Table : HA Application Monitoring API Calls for a description of each status value. The figure below shows a possible coding flow for the GetAppStatus call.

Figure 166: Coding flow for VMGuestAppMonitor_GetAppStatus



6. After you call `VMGuestAppMonitor_GetAppStatus`, call the `VMGuestAppMonitor_Free` function to free the memory that was used to store the status.

   If your application does not free the memory, it can use a large amount of memory very quickly because a new status is created every 30 seconds, after every `VMGuestAppMonitor_MarkActive` call.

7. Call `VMGuestAppMonitor_Disable` when you want the agent to stop monitoring.

HA Application Monitoring API Error Messages

Guest infrastructure in vSphere may return the following HA related errors.

Table : HA Application Monitoring Error Codes shows errors resulting from HA Application Monitoring calls.

Table 101: HA Application Monitoring Error Codes

Error Message	Data Type	Code	Description
VMGUESTAPPMONITORLIB_ERROR_SUCCESS	int	0	Call completed successfully.
VMGUESTAPPMONITORLIB_ERROR_OTHER	char		Unknown error.
VMGUESTAPPMONITORLIB_ERROR_NOT_RUNNING_IN_VM	char		Calling application is not running within a virtual machine.
VMGUESTAPPMONITORLIB_ERROR_NOT_ENABLED	char		Monitoring is not enabled.
VMGUESTAPPMONITORLIB_ERROR_NOT_SUPPORTED	char		Monitoring is not supported.

Virtual Disk Development Kit (VDDK) Programming Guide

The VMware® *Virtual Disk Development Kit Programming Guide* introduces the Virtual Disk Development Kit (VDDK) and the vSphere Storage APIs – Data Protection (VADP). For VDDK it describes how to develop software using a virtual disk library that provides a set of system-call style interfaces for managing virtual disks on ESXi hosts. For VADP it describes how to write backup and restore software that can be managed by vCenter Server®.

Revision History

This table summarizes significant changes in past versions of the guide.

Table 102: Revision History

Revision	Description
2024-6-18	VM managed by VM Service for Kubernetes.
2023-11-29	Expand changed block tracking troubleshooting. DNS. Add caveat about vSAN ESA.
2023-09-21	For vSphere 8.0 U2. Added section about container storage snapshots on TKGS.
2023-04-18	For vSphere 8.0 U1. Datastore level VMDK access, SAN performance enhancements.
2022-10-05	For vSphere 8.0. Backup of Datasets and Docker containers; SCSI controller ordering.
2022-05-16	Revised "Configure SCSI Controllers in Linux Proxy VM" topic. Alt tags for accessibility.
2021-08-23	For vSphere 7.0 Update 3. Configurable async I/O buffers for NFC.
2020-09-16	For vSphere 7.0 Update 1. Added best practices for NBD backup. Inclusive terminology (see below). Renamed allowed and disallowed lists for SAN transport.
2020-04-02	For vSphere 7.0. Dedicated network for NBD backups, CBT enhancements.
2019-08-20	For vSphere 6.7 Update 3. Note change of policy for phone home (CEIP) going forward.
2019-04-11	For vSphere 6.7 Update 2, minor corrections, Storage Spaces stop application quiescing.
2018-10-16	For vSphere 6.7 Update 1, improved VixMntapi library on Linux, additional OS support.
2018-04-17	For vSphere 6.7, async NBD, query allocated blocks, First Class Disk, vTPM, phone home.
2017-03-09	Added section to best practices about backing up and restoring encrypted virtual disks.
2017-01-11	Corrections regarding 2TB virtual disk limits, more about ESXi connections.
2016-11-15	Final version for VDDK 6.5. New support for encryption, NBDSSL compression, and VSS extensions.
2016-03-25	Removed statement that disks not associated with a virtual machine can be mounted.
2016-02-02	Fallback to NBDSSL, SCSI order for disk mount, no linked clone SAN restore, EUC non-support.
2015-07-13	Clarified connection port and RAID support, more allowed list details, added Linux freeze and thaw.
2015-05-05	Documented allowed and disallowed lists for device path selection, corrected install procedures.
2014-12-26	Final version for VDDK 6.0. Virtual volumes support. SSL certificate checking now mandatory.
2014-04-08	VDDK 5.5.1 supports GPT. Snapshot quiesce and memory are incompatible. VMDK version 3.
2013-11-08	Fixed several errors involving roles and licensing, physical or virtual proxy, and log level.
2013-10-14	Corrections regarding 32-bit Windows and PackageCode. Removed Reparent and Combine APIs.
2013-09-22	Final version for the vSphere 5.5 release, with new chapter on vApp backup for vCloud Director.
2012-12-21	Bug fix version of the vSphere 5.1 manual: numeric change ID policy, mount restrictions.
2012-10-05	Final version of this manual for the vSphere 5.1 release.
2011-11-18	Bug fix version for 4Q 2011 refresh of the VMware vSphere Documentation Center.
2011-08-22	Final version for the VDDK 5.0 release, subsuming *Designing Backup Solutions* technical note.
2010-10-12	Bug fix revision for the VDDK 1.2.1 release
2010-08-05	Version for vSphere 4.1 and the VDDK 1.2 public release.
2009-05-29	Final version for the VDDK 1.1 public release.

Table continued on next page

Continued from previous page

Revision	Description
2008-04-11	Updated version for release 1.0 of the Virtual Disk Development Kit.
2008-01-31	Initial version of the Virtual Disk Development Kit for partner release.

Intended Audience

This guide is intended for developers who are creating applications that manage virtual storage, especially backup and restore applications. It assumes knowledge of C and C++ programming. For VADP development, this guide assumes knowledge of Java.

Supported Platform Products

You can develop VDDK programs using either Linux or Windows, and test them using VMware Workstation or ESXi and vSphere. To develop and test VADP programs, you need a vCenter Server and ESXi hosts, preferably with shared cluster storage.

VMware Technical Publications

To access VMware SDK and API documentation, visit website https://developer.broadcom.com. To access VMware user and administrator manuals, visit website http://docs.vmware.com.

VMware updated this guide and others to remove instances of non-inclusive terminology.

The Virtual Disk API and vSphere

The virtual disk development kit (VDDK) is an SDK to help developers create applications that access storage on virtual machines. The VDDK package is based on the virtual disk API, introduced in this chapter.

The VMware Storage APIs – Data Protection (VADP) use the virtual disk API and a subset of vSphere APIs to take snapshots of virtual machines running on ESXi, enabling full or incremental backup and restore. VADP replaces VMware Consolidated Backup (VCB).

Virtual Disk Library VixDiskLib

The virtual disk API, or VixDiskLib, is a set of function calls to manipulate virtual disk files in VMDK format (virtual machine disk). Function call semantics are patterned after C system calls for file I/O. Using the virtual disk API, you can write programs to manage VMDK files directly from your software applications.

These library functions can manipulate virtual disks on VMware Workstation or similar products (hosted disk) or virtual disks residing on file system volumes of an ESXi host managed by vCenter Server (managed disk). Hosted is a term meaning that the virtualization platform and disk are hosted by a guest operating system such as Windows or Linux.

The VDDK package installs on either Windows or Linux, so you can write VDDK and VADP applications using either system. Applications can manipulate the virtual disks of any operating system that runs on a supported VMware platform product. You may repackage VDDK binaries into your software application after signing a redistribution agreement. See the *`VDDK Release Notes`* for a list of supported platform products and development systems.

The VDDK and VADP enable you to develop applications that work effectively across multiple virtual disks from a central location.

VDDK Components

The virtual disk development kit includes the following components:

- The virtual disk library, a set of C function calls to manipulate VMDK files
- The disk mount library, a set of C function calls to remote mount VMDK file systems

- C++ code samples that can be compiled with Visual Studio or the GNU C compiler
- PDF manuals and online HTML reference

Virtual Disk Library

VixDiskLib is a standalone wrapper library to help you develop solutions that integrate into VMware platform products. The virtual disk library has the following capabilities:

- It allows programs to create, convert, expand, defragment, shrink, and rename virtual disk files.
- It can create redo logs (parent-child disk chaining, or deltas) and it can delete VMDK files.
- It permits random read/write access to data anywhere in a VMDK file, and reads metadata.
- It can connect to remote vSphere storage using advanced transports, SAN or HotAdd.

For Windows, the virtual disk kernel-mode driver should be 64-bit. User libraries could be 32-bit because Windows On Windows 64 can run 32-bit programs without alteration. VMware provides only 64-bit libraries.

Disk Mount Library

The virtual disk mount library, vixMntapi, allows programmatic access of virtual disks as if they were mounted disk partitions. For more information see Virtual Disk Mount API. The vixMntapi library is packaged in the VDDK with vixDiskLib.

Virtual Disk Utilities

The Virtual Disk Development Kit used to include two command-line utilities for managing virtual disk files: disk mount and virtual disk manager. They were last delivered in the VDDK 5.0 release. For more information see the old *Disk Mount and Virtual Disk Manager User's Guide*, still available on the Web.

Backup and Restore on vSphere

The VMware Storage APIs – Data Protection (VADP) is a collection of APIs that are useful for developing or extending backup software so it can protect virtual machines running on ESXi hosts in VMware based datacenters. For more information see Backing Up Virtual Disks in vSphere.

Backup Design for vCloud Director

With VMware vCloud®, the self-service capabilities of vCloud Director provide three levels of data protection. Backup providers can offer vApp protection at the system level, the tenant level, or the end-user level. For information about vCloud data protection, see the technical note *Backup Design for vCloud Tenant vApps*.

Use Cases for the Virtual Disk Library

The library provides access to virtual disks, enabling a range of use cases for application vendors including:

- Back up a particular volume, or all volumes, associated with a virtual machine.
- Connect a backup proxy to vSphere and back up all virtual machines on a storage cluster.
- Read virtual disk and run off-line anti-virus scanning, or package analysis, of virtual machines.
- Write to virtual disk to perform off-line centralized patching of virtual machines.
- Manipulate virtual disks to defragment, expand, convert, rename, or shrink the file system image.
- Perform data recovery or virus cleaning on corrupt or infected off-line virtual machines.

Developing for VMware Platform Products

In a VMware based data center, commercial backup software is likely to access virtual disks remotely, perhaps from a backup proxy. The proxy can be a virtual machine or a physical machine with backup-restore software installed and access to alternate storage such as a tape autochanger or equivalent.

At a given point in time, during the backup window, backup software:

- Snapshots virtual machines in a cluster, one by one, or in parallel. Virtual machines run off the snapshot.
- Copies the quiesced base disk, or (for incremental backup) only changed blocks, to backup media.
- Records the configuration of virtual machines.
- Reverts and deletes snapshots, so virtual machines retain any changes made during the backup window.

In the above procedure, the virtual disk library is used in the second step only. The other steps use a portion of the vSphere API (called VADP) to snapshot and save configuration of virtual machines. The virtual disk in a cluster is “managed” by vSphere.

Managed Disk and Hosted Disk

Analogous to a hard disk drive, virtual disk files represent the storage volumes of a virtual machine. Each is named with `.vmdk` suffix. On a system running VMware Workstation, file systems of each guest OS are kept in VMDK files hosted on the system’s physical disk. VMDK files can be accessed directly on the host.

With the virtual machine file system (VMFS) on ESXi hosts, VMDK files again represent storage volumes of virtual machines. They are on VMFS, which often resides on shared storage in a cluster. The vCenter Server manages the cluster storage so it can migrate (vMotion) virtual machines from one ESXi host to another without moving VMDK files. VMFS storage is therefore called managed disk.

VMFS disk can reside on a storage area network (SAN) attached to ESXi hosts by Fibre Channel, iSCSI, or SAS connectors. It can also reside on network attached storage (NAS), or on directly attached disk.

Figure: Managed Disk on vSphere depicts the arrangement of managed disk, as VMDK files in a SAN based cluster. Figure: Hosted Disk on Workstation depicts hosted disk on VMware Workstation, as VMDK files on physical disk.

Figure 167: Managed Disk on vSphere

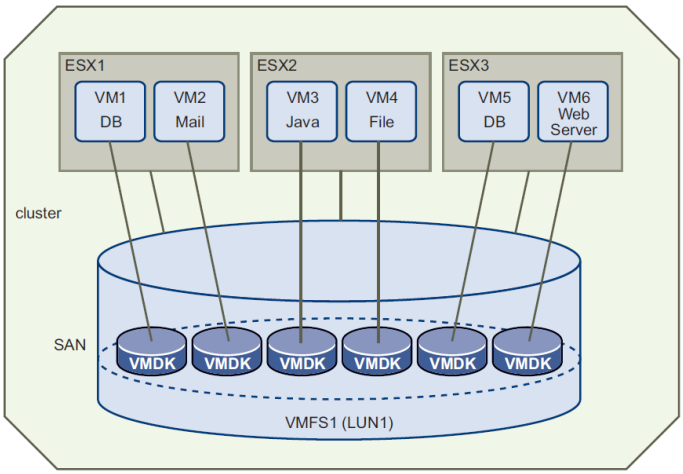
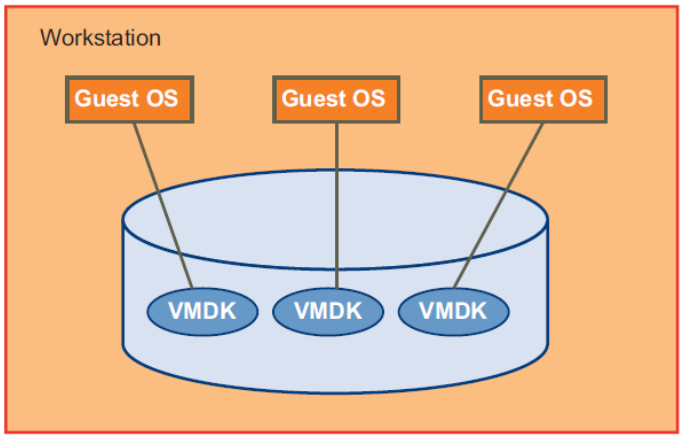


Figure 168: Hosted Disk on Workstation



The VDDK supports both managed disk and hosted disk. Some functions are not supported for managed disk, and others are not supported for hosted disk, as noted in documentation. Managed virtual disk files larger than 2TB are supported by vSphere 5.5 and later.

Advanced Transports

With managed disk, VDDK applications can make use of advanced transports to perform many I/O operations directly on the SAN, rather than over the LAN. This improves performance and saves network bandwidth.

VDDK and VADP Compared

The Virtual Disk Development Kit (VDDK) includes a set of C library routines for manipulating virtual disk (VixDiskLib) and for mounting virtual disk partitions (VixMntapi). The VDDK focuses on efficient access and transfer of data on virtual disk storage.

The vSphere Storage APIs for Data Protection (VADP) is a marketing term for a subset of the vSphere API that enables backup and restore applications. The snapshot-based VADP framework allows efficient, off-host, centralized backup of virtual machine storage. After taking a snapshot to quiesce virtual disk, software can then back up storage using VDDK library routines.

The vSphere API is a network-based Web service that provides the interfaces for vCenter Server management of virtual machines running on ESXi hosts.

Developers need both VDDK and VADP to write data protection software. VADP is presented in Backing Up Virtual Disks in vSphere.

Platform Product Compatibility

To support a new release of vSphere, in most cases you should update and recompile your software with a corresponding new release of VDDK. This is because VDDK is continually updated to support new features in vSphere. As of 5.0, the version number of VDDK matches the version number of vSphere.

Since its inception in 2008, VDDK has been backward compatible with virtual platform products such as VMware Workstation, ESXi, and vCenter Server. VMware Fusion was never supported.

Redistributing VDDK Components

After you use the VDDK to develop software applications that run on VMware platform products, you might need to repackage library components that are compiled into your software.

To qualify for VDDK redistribution, you must be in the VMware TAP program at Standard level or above, and sign a redistribution agreement. Contact your VMware alliance manager to request the VDDK redistribution agreement. VMware would like to know how you use the VDDK, in what products you plan to redistribute it, your company name, and your contact information.

Installing the Development Kit

To develop virtual disk applications, install the VDDK as described in this chapter. For backup applications, VADP development also requires the vSphere Web Services SDK.

Prerequisites for VDDK and VADP

The VDDK has been tested and is supported on the following operating systems.

- Windows 64-bit (x86-64) systems
- Linux 64-bit (x86-64) systems

See the *VDDK Release Notes* for specific versions, which change over time. MacOS is not supported.

VADP development is often done with Java, although vSphere Web Services can use other languages.

Programming Environments

You can compile the sample program and develop vSphere applications in the following environments:

Visual Studio on Windows

On Windows, programmers can use the C++ compiler in Visual Studio 2005, Visual Studio 2008, and later. Along with Visual Studio, you also need to install the 64-bit debugging tools.

C++ and C on Linux

On Linux, programmers can use the GNU C compiler, version 4 and higher. The sample program compiles with the C++ compiler **g++**, but VDDK also works with the C compiler **gcc**.

Java Development for VADP

When developing backup and restore software to run on vSphere, VMware recommends Eclipse with Java, on both Windows and Linux. The vSphere Web Services SDK now includes both Axis and JAX-WS bindings. You can call C or C++ code with wrapper classes, as in Java Native Interface (JNI).

VMware Platform Products

Software applications developed with the VDDK and VADP target the following platform products:

- vCenter Server managing ESXi hosts
- ESXi hosts directly connected

See the *VDDK Release Notes* for specific versions, which change over time.

Hosted products including VMware Workstation are neither tested nor supported.

Storage Device Support

VMware Consolidated Backup (VCB) had a knowledge base article showing the support matrix for storage devices and multipathing. Although VMware provides a hardware compatibility guide, there is no support matrix for VDDK and VADP. Customers must get this information from the vendor of their backup software.

Installing the VDDK Package

The VDDK is packaged as a compressed archive for Windows 64-bit and for Linux 64-bit. The VDDK packages include the following components:

- Header files **vixDiskLib.h** and **vm_basic_types.h** in the **include** directory.
- Function library vixDiskLib.lib (Windows) or **libvixDiskLib.so** (Linux) in the **lib** directory.
- HTML reference documentation in the **doc** directory and sample program in doc/samples.

> **NOTE**
> In the VDDK 5.5 release, VMware discontinued the Windows installer and 32-bit executables for Windows and Linux. The VDDK is available for 64-bit systems only.

To install the package on Windows:

1. On the Download page, choose the .zip file for Windows and download it to your development system.
2. Place the .zip file in a folder under **Program Files** – you can choose the name – and unpack it:
   ```
   cd C:\Program Files\VMware\VDDK670

   unzip VMware-vix-disklib-*.zip
   ```
3. Go to the **bin** subfolder, locate the vstor2install.bat script, and double-click to run it. The batch script should be run in place so that the current directory for execution is the **bin** subfolder. By running it, you implicitly accept the VMware license terms.

   > **NOTE**
   > If **vstor2** is already installed on a backup proxy, you should first uninstall it with **vstor2uninstall.bat** (from its corresponding VDDK version) then run the new install script.

4. Edit the Windows registry with **regedit** and check for the following key. If this key exists from a previous VDDK install, right-click to delete it. Add a registry entry with the following key:
   ```
   HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\VMware, Inc.\VMware Virtual Disk Development Kit
   ```
   For convenience you might want to edit the Windows **Path** environment to include the VDDK installation folder, C:\Program Files\VMware\VDDK550\bin in the example above.

To Install the package on Linux:

1. On the Download page, choose the binary **tar.gz** for 64-bit Linux.
2. Unpack the archive with **tar** to create the **vmware-vix-disklib-distrib** subdirectory.
   ```
   $ tar xvzf VMware-vix-disklib-*.tar.gz
   ```
3. Change to the newly created directory to see its subdirectories:
   ```
   $ cd vmware-vix-disklib-distrib; ls

   bin64  doc  FILES  include  lib32  lib64
   ```
   The **bin64** subdirectory contains a reporting program and virtual disk manager. License terms are in the doc/EULA file. The sample program is under doc/samples. Header files in the **include** subdirectory are for compiling your VDDK applications. Normally the **lib64** components are installed under /usr/lib, /usr/lib/vmware-vix-disklib/lib64 for instance.
4. Install components as needed. You might want to edit your **LD_LIBRARY_PATH** environment to include the library installation path. Alternatively, you can add the library location to the list in /etc/ld.so.conf and run **ldconfig** as the superuser.

Repackaging VDDK Libraries

After you develop an application based on VDDK, you might need the VDDK binaries to run your application.

As described in Redistributing VDDK Components, partners can sign a license agreement to redistribute VDDK binaries that support VADP applications.
To enable VDDK binaries on Windows virtual machines without VDDK installed

1. Install the Microsoft Visual C++ (MSVC) redistributable, possibly as a merge module. The latest MSVC runtime works as side-by-side component, so manually copying it might not work on Vista. See details on the Microsoft Web site for the redistributable package, x86 processors or x64 processors. Side-by-side is also explained on the Microsoft Web site.
2. Install VMware executables and DLLs from the `\bin` and `\lib` folders of the installed VDDK, and the `vstor2-mntapi10.sys` driver into the `Windows\system\drivers` folder or equivalent.
3. Create and install your application, compiled in a manner similar to the `vixDiskLibSample.exe` code, discussed in Virtual Disk API Sample Code.

How to Find VADP Components

ESXi hosts and vCenter Server similarly implement managed objects that support inventory traversal and task requests. Before you write VADP software in Java, you need to download the vSphere Web Services SDK. You can find documentation and ZIP file for download on the VMware Web site.

Virtual Disk Interfaces

VMware offers many options for virtual disk layout, encapsulated in library data structures described here.

VMDK File Location

On ESXi hosts, virtual machine disk (VMDK) files are usually located under one of the `/vmfs/volumes`, perhaps on shared storage. Storage volumes are visible from the vSphere Client, in the inventory for hosts and clusters. Typical names are **`datastore1`** and **`datastore2`**. To see a VMDK file, click **Summary › Resources › Datastore**, right-click **Browse Datastore**, and select a virtual machine.

On Workstation, VMDK files are stored in the same directory with virtual machine configuration (VMX) files, for example `/path/to/disk` on Linux or `C:\My Documents\My`**`Virtual Machines`** on Windows.

VMDK files store data representing a virtual machine's hard disk drive. Almost the entire portion of a VMDK file is the virtual machine's data, with a small portion allotted to overhead.

Virtual Disk Types

The following disk types are defined in the virtual disk library:

- **`VIXDISKLIB_DISK_MONOLITHIC_SPARSE`** – Growable virtual disk contained in a single virtual disk file. This is the default type for hosted disk, and the only setting in the Virtual Disk API Sample Code sample program.
- **`VIXDISKLIB_DISK_MONOLITHIC_FLAT`** – Preallocated virtual disk contained in a single virtual disk file. This takes time to create and occupies a lot of space, but might perform better than sparse.
- **`VIXDISKLIB_DISK_SPLIT_SPARSE`** – Growable virtual disk split into 2GB extents (**`s`** sequence). These files can to 2GB, then continue growing in a new extent. This type works on older file systems.
- **`VIXDISKLIB_DISK_SPLIT_FLAT`** – Preallocated virtual disk split into 2GB extents (**`f`** sequence). These files start at 2GB, so they take a while to create, but available space can grow in 2GB increments.
- **`VIXDISKLIB_DISK_VMFS_FLAT`** – Preallocated virtual disk compatible with ESX 3 and later. Also known as thick disk. This managed disk type is discussed in Managed Disk and Hosted Disk.

- **VIXDISKLIB_DISK_VMFS_SPARSE** – Employs a copy-on-write (COW) mechanism to save storage space.
- **VIXDISKLIB_DISK_ VMFS_THIN** – Growable virtual disk that consumes only as much space as needed, compatible with ESX 3 or later, supported by VDDK 1.1 or later, and highly recommended.
- **VIXDISKLIB_DISK_STREAM_OPTIMIZED** – Monolithic sparse format compressed for streaming. Stream optimized format does not support random reads or writes.

Persistence Disk Modes

In persistent disk mode, changes are immediately and permanently written to the virtual disk, so that they survive even through to the next power on.

In nonpersistent mode, changes to the virtual disk are discarded when the virtual machine powers off. The VMDK files revert to their original state.

The virtual disk library does not encapsulate this distinction, which is a virtual machine setting.

VMDK File Naming

Table : VMDK Virtual Disk Files explains the different types of virtual disk. The first column corresponds to Virtual Disk Types but without the **VIXDISKLIB_DISK** prefix. The third column gives the possible names of VMDK files as implemented on Workstation and ESXi hosts.

> **NOTE**
> When you open a VMDK file with the virtual disk library, always open the one that points to the others, not the split or flat sectors. The file to open is most likely the one with the shortest name.

For information about other virtual machine files, see section “Files that Make Up a Virtual Machine” in the *VMware Workstation User’s Manual*. On ESXi hosts, VMDK files are type **VMFS_FLAT** or **VMFS_THIN**.

Table 103: VMDK Virtual Disk Files

Disk Type in API	Virtual Disk Creation on VMware Host	Filename on Host
MONOLITHIC_SPARSE	In Select A Disk Type, accepting the defaults by not checking any box produces one VMDK file that can grow larger if more space is needed. The **<vmname>** represents the name of a virtual machine. On VMFS partitions, this is name of the disk descriptor file.	<vmname>.vmdk
MONOLITHIC_FLAT or **VMFS_FLAT** or **VMFS_THIN**	If you select only the Allocate all disk space now check box, space is pre-allocated, so the virtual disk cannot grow. The first VMDK file is small and points to a much larger one, whose filename says **flat** without a sequence number. Similarly on VMFS partitions, this is the virtual disk file that points to virtual disk data files, either thick or thin provisioned.	<vnname>-flat.vmdk
SPLIT_SPARSE	If you select only the Split disk into 2GB files check box, virtual disk can grow when more space is needed. The first VMDK file is small and points to a sequence of other VMDK files, all of which have an **s** before a sequence number, meaning sparse. The number of VMDK files depends on the	<vmname>-s<###>.vmdk

Table continued on next page

Continued from previous page

Disk Type in API	Virtual Disk Creation on VMware Host	Filename on Host
	disk size requested. As data grows, more VMDK files are added in sequence.	
SPLIT_FLAT	If you select the Allocate all disk space now and Split disk into 2GB files check boxes, space is pre-allocated, so the virtual disk cannot grow. The first VMDK file is small and points to a sequence of other files, all of which have an **f** before the sequence number, meaning flat. The number of files depends on the requested size.	<vnname>-f<###>.vmdk
MONOLITHIC_SPARSE or **SPLIT_SPARSE** snapshot	A redo log (or child disk or delta link) is created when a snapshot is taken of a virtual machine, or with the virtual disk library. Snapshot file numbers are in sequence, without an **s** or **f** prefix. The numbered VMDK file stores changes made to the virtual disk **<diskname>** since the original parent disk, or previously numbered redo log (in other words the previous snapshot).	<diskname>-<###>.vmdk
SE_SPARSE	Space-efficient sparse (**seSparse**) format. In vSphere 5.1 and later, used by VMware View to optimize linked clone templates. In the vSphere API, see data object **SeSparseVirtualDiskSpec**. Use of **seSparse** as a base disk is neither documented nor supported.	
n/a	Snapshot of a virtual machine, which includes pointers to all its .vmdk virtual disk files.	<vnname>Snapshot.vmsn

For lazy zeroed thick disk, all blocks are allocated, and data written to used blocks, however unused blocks are left as-is, so they may contain data from previous use. Many storage systems will zero-out unused blocks in the background. With eager zeroed thick disk, unused blocks are zeroed-out at allocation time.

Thin Provisioned Disk

With thin provisioned disk, the vSphere Client may report that provisioned size is greater than disk capacity.

Provisioned size for a thin disk is the maximum size the disk will occupy when fully allocated. Actual size is the current size of the thin disk. Overcommit means that if all thin disks were fully provisioned, there would not be enough space to accommodate all of the thin disks.

Internationalization and Localization

VDDK libraries are not localized, but backup partners can support any locale that uses UTF-8 encoding. Other than Unicode, VDDK does not support multibyte character encodings such as Extended Unix Code (EUC) for Chinese, Japanese, and Korean (ISO-2022).

The path name to a virtual machine and its VMDK can be expressed with any character set supported by the host file system. As of vSphere 4 and Workstation 7, VMware supports Unicode UTF-8 path names, although for portability to various locales, ASCII-only path names are recommended.

Windows systems use UTF-16 for localized path names. For example, in locale FR (Français) the VDDK sample code might mount disk at C:\Windows\Temp\vmware-Système, where è is encoded as UTF-16 so the VixMntapi library cannot recognize it. In this case, a workaround is to set the **tmpDirectory** configuration key with an ASCII-only path before program start-up; see Initialize the Library.

For programs opening arbitrary path names, Unicode offers a GNU library with C functions **`iconv_open()`** to initialize codeset conversion, and **`iconv()`** to convert UTF-8 to UTF-16, or UTF-16 to UTF-8.

Grain Directories and Grain Tables

`SPARSE` type virtual disks use a hierarchical representation to organize sectors. See *`Virtual Disk Format 5.0`*. In this context, grain means granular unit of data, larger than a sector. The hierarchy includes:

- Grain directory (and redundant grain directory) whose entries point to grain tables.
  - Grain tables (and redundant grain tables) whose entries point to grains.
    - Each grain is a block of sectors containing virtual disk data. Default size is 128 sectors or 64KB.

Data Structures in Virtual Disk API

Here are important data structure objects with brief descriptions:

- **`VixError`** – Error code of type **`uint64`**.
- **`VixDiskLibConnectParams`** – Public types designate the virtual machine credentials **`vmxSpec`** (possibly through vCenter Server), the name of its host, and the credential type for authentication. For details, see VMX Specification. The **`credType`** can be **`VIXDISKLIB_CRED_UID`** (user name / password, most common), **`VIXDISKLIB_CRED_SESSIONID`** (the HTTP session ID), **`VIXDISKLIB_CRED_TICKETID`** (vSphere ticket ID), or **`VIXDISKLIB_CRED_SSPI`** (Windows only, current thread credentials).

  ```
  typedef char * vmxSpec

  typedef char * serverName

  typedef VixDiskLibCredType credType
  ```

- VDDK 6.7 has a new union in the **`VixDiskLibConnectParams`** structure; see xx.
- **`VixDiskLibConnectParams::VixDiskLibCreds`** – Credentials for either user ID or session ID.
  `VixDiskLibConnectParams::VixDiskLibCreds::VixDiskLibUidPasswdCreds` – String data fields represent user name and password for authentication.

  `VixDiskLibConnectParams::VixDiskLibCreds::VixDiskLibSessionIdCreds` – String data fields represent the session cookie, user name, and encrypted session key.

  `VixDiskLibConnectParams::VixDiskLibCredsb::VixDiskLibSSPICreds` – String data fields for security support provider interface (SSPI) authentication. User name and password are null.
- **`VixDiskLibCreateParams`** – Types represent the virtual disk (see Virtual Disk Types), the disk adapter (see Adapter Types), VMware version, and capacity of the disk sector.
  ```
  typedef VixDiskLibDiskType diskType

  typedef VixDiskLibAdapterType adapterType

  typedef uint hwVersion

  typedef VixDiskLibSectorType capacity
  ```

- **`VixDiskLibDiskInfo`** – Types represent the geometry in the BIOS and physical disk, the capacity of the disk sector, the disk adapter (see Adapter Types), the number of child-disk links (redo logs), and a string to help locate the parent disk (state before redo logs).
  ```
  VixDiskLibGeometry biosGeo

  VixDiskLibGeometry physGeo
  ```

```
VixDiskLibSectorType capacity

VixDiskLibAdapterType adapterType

int numLinks

char * parentFileNameHint
```

- **`VixDiskLibGeometry`** – Types specify virtual disk geometry, not necessarily the same as physical disk geometry.
```
typedef uint32 cylinders

typedef uint32 heads

typedef uint32 sectors
```

As of vSphere 8.0 U2, upon failure `VixDiskLib_Open()` returns the underlying network error, rather than generic error 13 "cannot open file" or similar message. Now applications can better handle conditions such as `VIX_E_HOST_SERVER_SHUTDOWN` for machine powered off or `VIX_E_HOST_NETWORK_CONN_REFUSED` for network dysfunction.

Datastore-Level Access to VMDK

As of vSphere 8.0 U1, VDDK offers direct access to a datastore, in addition to pre-existing folder path access to disk.

For direct datastore access, there is no support for snapshots and CBT. Currently only NBD/NBDSSL and HotAdd transport modes are supported. HotAdd transport does not support RDM disks. For datacenters with encrypted disks, NBDSSL transport mode is required, and clusters must activate vSphere HA to guarantee that all hosts have a crypto key ID. Steps to employ this feature are below.

VDDK header file `vixDiskLib.h` now has a structure definition for `VixDiskLibDatastoreSpec` containing two strings: a managed object reference (MoRef) to the datastore, and a folder path to the disk. Datastore specification is enumerated as type 2 in `VixDiskLibSpecType`. Programs call `VixDiskLib_ConnectEx()` to connect to a datastore instead of a virtual machine or first class disk (FCD). Then programs call `VixDiskLib_Open()` on that datastore with a folder path to open the specified disk.

1. Connect to datastore.
```
VixDiskLibConnection theDSconn;

VixDiskLibConnectParams *cnxParams = VixDiskLib_AllocateConnectParams();

connParams->specType = VIXDISKLIB_SPEC_DATASTORE;

cnxParams->spec.dsSpec.datastoreMoRef = "datastore-19";

cnxParams->spec.dsSpec.diskFolder = "RHELXfsTargetClone_dest";

VixDiskLib_ConnectEx(cnxParams, ..., &theDSconn);
```
2. Open disk on the datastore.
```
VixDiskLibHandle diskHdl;

VixDiskLib_Open(theDSconn, /* diskPath */, /* open flags */, &diskHdl);
```
3. Read from or write to the disk.
4. Close the disk.
5. Disconnect from VixDiskLib.

Credentials and Privileges for VMDK Access

Local operations are supported by local VMDK. Access to ESXi hosts is authenticated by login credentials, so with proper credentials VixDiskLib can reach any VMDK on an ESXi host. VMware vSphere has its own set of privileges, so with the proper privileges (see below) and login credentials, VixDiskLib can reach any VMDK on an ESXi host managed by vCenter Server. VixDiskLib supports the following:

- Both read-only and read/write modes
- Read-only access to disk associated with any snapshot of online virtual machines
- Access to VMDK files of offline virtual machines (vCenter restricted to registered virtual machines)
- Reading of Microsoft Virtual Hard Disk (VHD) format

With vCenter Server, the Role of the backup appliance when saving data must have these privileges for all the virtual machines being backed up:

- Virtual machine > Change Configuration > Toggle disk change tracking
- Virtual machine > Provisioning > Allow read-only disk access *and* Allow VM download
- Virtual machine > Snapshot management > Create snapshot *and* Remove snapshot

On the backup appliance, the user must have the following privileges:

- Datastore > Allocate space
- Virtual machine > Change Configuration > Add new disk *and* Remove disk
- Virtual machine > Change Configuration > Change resource *and* Settings

The user must have this privilege for vCenter Server and all ESXi hosts involved in backup:

- Global > Disable methods *and* Enable methods

If privileges are not applied at the vCenter Server level, the returned error message is somewhat misleading: “The host is not licensed for this feature.”

Adapter Types

The library can select the following adapters:

- **`VIXDISKLIB_ADAPTER_IDE`** – Virtual disk acts like ATA, ATAPI, PATA, SATA, and so on. You might select this adapter type when it is specifically required by legacy software.
- **`VIXDISKLIB_ADAPTER_SCSI_BUSLOGIC`** – Virtual SCSI disk with Buslogic adapter. This is the default on some platforms and is usually recommended over IDE due to higher performance.
- **`VIXDISKLIB_ADAPTER_SCSI_LSILOGIC`** – Virtual SCSI disk with LSI Logic adapter. Windows Server 2003 and most Linux virtual machines use this type by default. Performance is about the same as Buslogic.

Virtual Disk Transport Methods

VMware supports file-based or image-level backups of virtual machines hosted on an ESXi host with SAN or NAS storage. Virtual machines read data directly from a shared VMFS LUN, so backups are efficient and do not put significant load on production ESXi hosts or the virtual network.

VMware offers interfaces for integration of storage-aware applications, including backup, with efficient access to storage clusters. Developers can use VDDK advanced transports, which provide efficient I/O methods to maximize backup performance. VMware supports four access methods: local file, NBD (network block device) over LAN, NBD with SSL encryption (NBDSSL), SAN, and SCSI HotAdd.

Local File Access

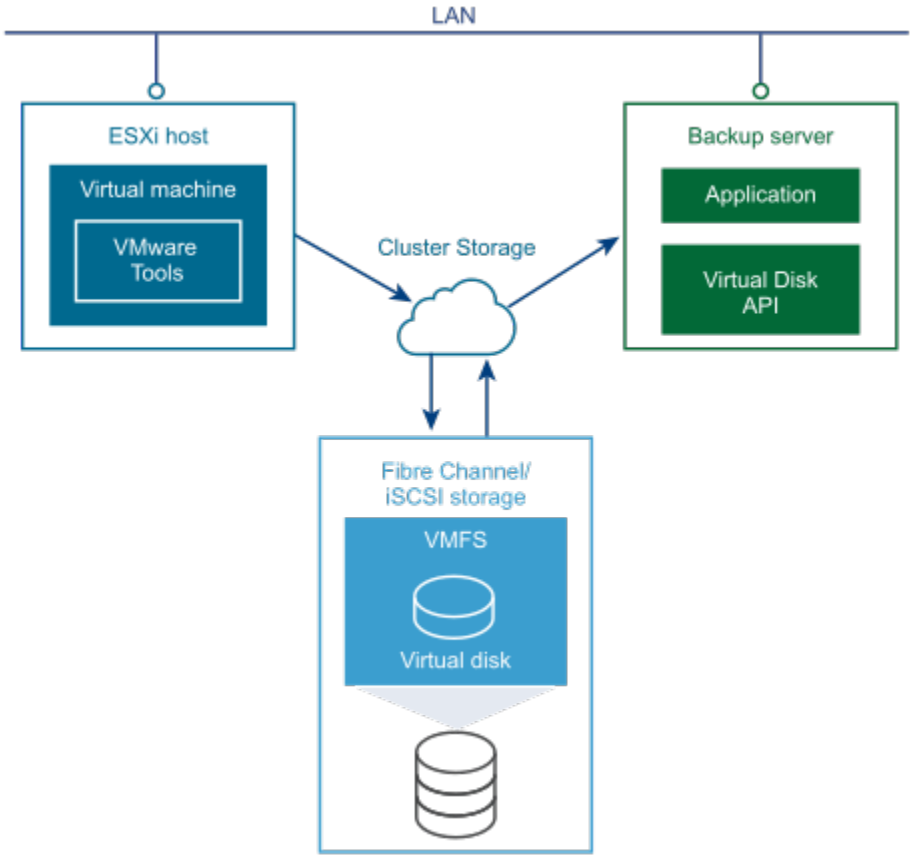
The virtual disk library can read virtual disk data from `/vmfs/volumes` on ESXi hosts, or from the local file system on hosted products. This file access method is built into VixDiskLib, so it is always available on local storage. However it is not a network transport method, and is seldom used for vSphere backup.

SAN Transport

SAN mode requires applications to run on a backup server with access to SAN storage (Fibre Channel, iSCSI, or SAS connected) containing the virtual disks to be accessed. As shown in the figure below, this method is efficient because no data needs to be transferred through the production ESXi host. A SAN backup proxy must be a physical machine. If it has optical media or tape drive connected, backups can be made entirely LAN-free.

In SAN transport mode, the virtual disk library obtains information from an ESXi host about the layout of VMFS LUNs, and using this information, reads data directly from the storage LUN where a virtual disk resides. This is the fastest transport method for software deployed on SAN-connected ESXi hosts.

Figure 169: SAN transport mode for backup



In general, SAN transport works with any storage device that appears at the driver level as a LUN (as opposed to a file system such as NTFS or EXT). SAN mode must be able to access the LUN as a raw device. The real key is whether the device behaves like a direct raw connection to the underlying LUN. SAN transport is supported in Fibre Channel, iSCSI, and SAS based storage arrays (SAS means serial attached SCSI). SAN storage devices can contain SATA drives, but currently there are no SATA connected SAN devices on the VMware hardware compatibility list.

SAN transport is not supported for backup or restore of virtual machines residing on vVol datastores.

VMware vSAN, a network storage solution with host-attached disks, does not support SAN transport. Because vSAN has modes that are incompatible with SAN transport, if the virtual disk library detects vSAN storage, it deactivates SAN transport. Other transport modes do work.

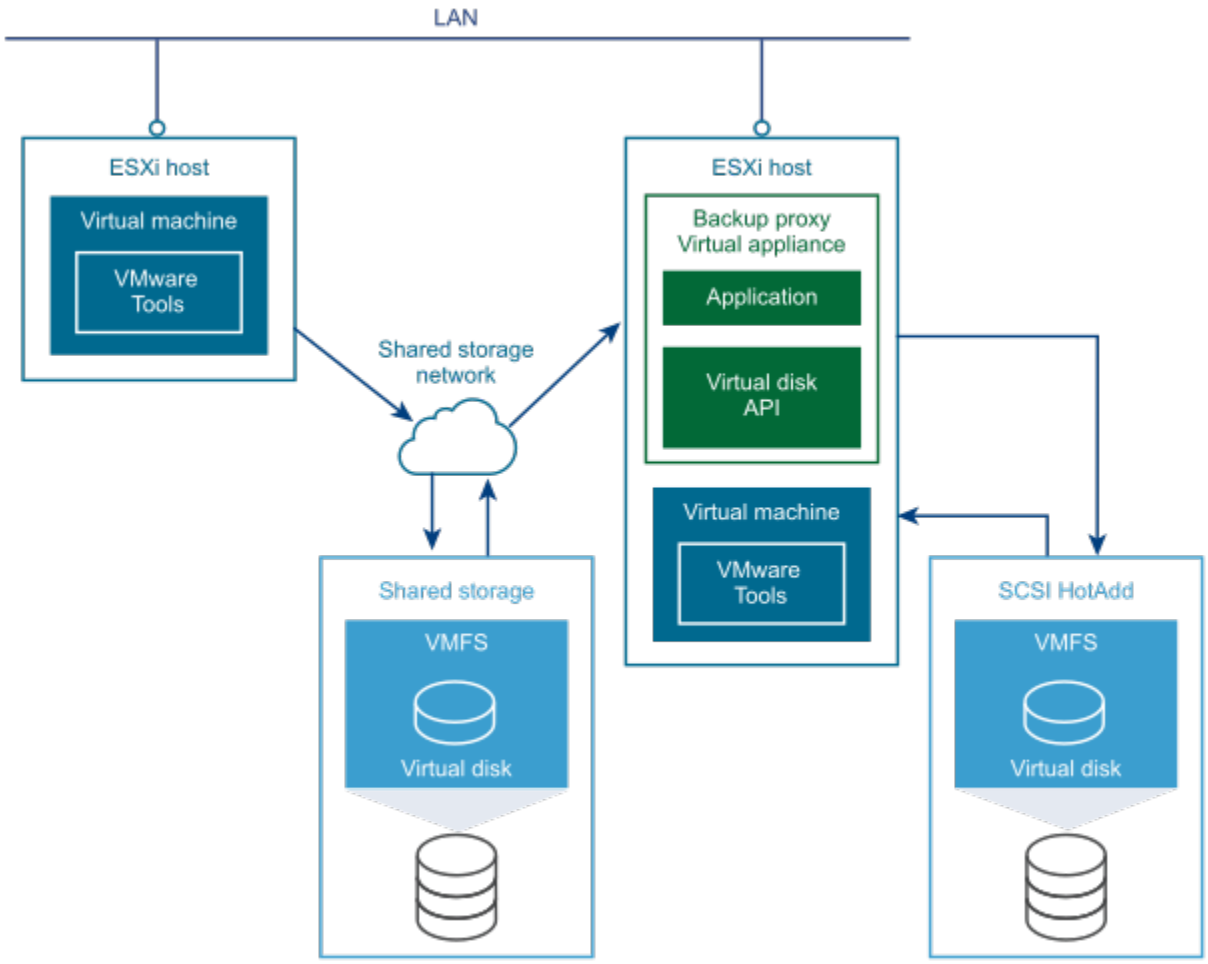
By design, vSAN ESA (express storage architecture) makes a VMDK read-only when it has a snapshot. Do not attempt to restore virtual disks on vSAN ESA if they have snapshot(s).

HotAdd Transport

HotAdd is a VMware feature where devices can be added “hot” while a virtual machine is running. Besides SCSI disk, virtual machines can add additional CPUs and memory capacity.

HotAdd is a good way to get virtual disk data from a virtual machine to a backup appliance (or backup proxy) for sending to the media server. The attached HotAdd disk is shown in the figure below.

Figure 170: HotAdd transport mode for backup



Running the backup proxy as a virtual machine has two advantages: it is easy to move a virtual machine to a new media server, and it can back up local storage without using the LAN, although this incurs more overhead on the physical ESXi host than when using SAN transport mode.

If backup software runs in a virtual appliance, it can take a snapshot and create a linked clone of the target virtual machine, then attach and read the linked clone’s virtual disks for backup. This involves a SCSI HotAdd on the ESXi host where the target VM and backup proxy are running. Virtual disks of the linked clone are HotAdded to the backup proxy. The target virtual machine continues to run during backup.

VixTransport handles the temporary linked clone and hot attachment of virtual disks. VixDiskLib opens and reads the HotAdded disks as a “whole disk” VMDK (virtual disk on the local host). This strategy works only on virtual machines with

SCSI disks and is not supported for backing up virtual IDE disks. HotAdd transport also works with virtual machines stored on NFS partitions.

About the HotAdd Proxy

The HotAdd backup proxy must be a virtual machine. HotAdd involves attaching a virtual disk to the backup proxy, like attaching disk to a virtual machine.

The HotAdd proxy must have access to the same datastore as the target virtual machine, and the VMFS version and data block sizes for the target VM must be the same as the datastore where the HotAdd proxy resides.

In some implementations, a different HotAdd proxy backs up Windows versus Linux virtual machines. In other backup implementations, the same type of HotAdd proxy backs up both.

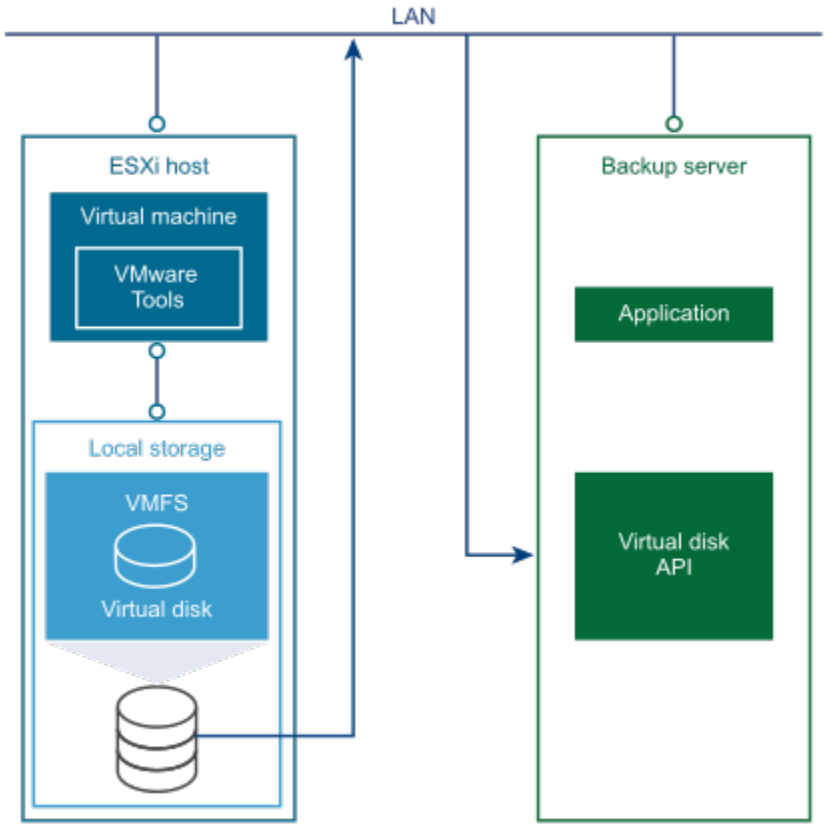
For parallel backup, sites can deploy multiple HotAdd proxies.

NBDSSL Transport

When no other transport is available, networked storage applications can use LAN transport for data access, with NBD (network block device) protocol, optionally with SSL encryption, called NBDSSL. NBD is a Linux-style kernel module that treats storage on a remote host as a block device. NBDSSL is a VMware variant that uses SSL to encrypt all data passed over the TCP connection. The NBDSSL transport method is built into the virtual disk library, so it is always available, and is the fall-back when no other transport method is available.

VMware libraries often fall back to NBDSSL when other transports are not available. Backup applications can select NBD for higher throughput.

Figure 171: NBDSSL transport mode for backup



In this mode, the ESXi host reads data from storage and sends it across a network to the backup server. With LAN transport, large virtual disks can take a long time to transmit. This transport mode adds traffic to the LAN, unlike SAN and HotAdd transport, but NBDSSL transport offers the following advantages:

- The ESXi host can use any storage device, including local storage or remote-mounted NAS.
- The backup proxy can be a virtual machine, so customers can use vSphere resource pools to minimize the performance impact of backup. For example, the backup proxy can be in a lower-priority resource pool than the production ESXi hosts.

When VDDK opens a non-snapshot disk for NBDSSL transfer (read-only or read/write) it selects the ESXi host where the disk's virtual machine currently resides.

However when VDDK opens a snapshot for NBDSSL transfer, the common backup case, VDDK passes the datastore to vCenter Server, which consults its list of ESXi hosts with access to the datastore; vCenter picks the first host with read/write access. The list of hosts is unordered, so the host chosen for NBDSSL transfer of the snapshot is not necessarily the ESXi host where the snapshot's virtual machine resides.

NBDSSL Performance

When reading disk data using NBDSSL transport, VDDK makes synchronous calls. That is, VDDK requests a block of data and waits for a response. The block is read from disk and copied into a buffer on the server side, then sent over the network. Meanwhile, no data gets copied over the network, adding to wait time. To some extent, you can overcome this limitation by using multiple streams to simultaneously read from a single disk or multiple disks, taking advantage of parallelism.

As of vSphere 6.5, NBDSSL performance can be significantly improved using data compression. Three types are available – **`zlib`**, **`fastlz`**, and **`skipz`** – specified as flags when opening virtual disks with the extended **`VixDiskLib_Open()`** call. See Open a Local or Remote Disk.

Asynchronous Mode NBDSSL

As of vSphere 6.7, asynchronous I/O for NBDSSL transport mode is available. Performance of NBDSSL mode data transfer can be improved with this option.

To implement asynchronous NBDSSL, use the new functions `VixDiskLib_ReadAsync` and `VixDiskLib_WriteAsync` in a callback, followed by `VixDiskLib_Wait`, to replace the VixDiskLib read and write functions. In the development kit, see `vixDiskLibSample.cpp` for code examples, following logic for the **`-readasyncbench`** and **`-writeasyncbench`** options.

VDDK has had non-public asynchronous APIs since VDDK 6.0, but behavior is different in earlier releases. File transport mode is synchronous in all releases.

Table 104: Asynchronous read and write

	VDDK 6.0	VDDK 6.5	6.7
`VixDiskLib_ReadAsync`	Always synchronous.	Same as 6.0.	For HotAdd and SAN modes, same as 6.0 and 6.5. NBDSSL mode is asynchronous if the ESXi host supports asynchronous NBD.
`VixDiskLib_WriteAsync`	For HotAdd and NBDSSL modes, always synchronous. For SAN mode, the API is asynchronous. A thread handles writes one by one in the back-end, so outstanding I/O is always 1.	Same as 6.0.	For HotAdd and SAN modes, same as 6.0 and 6.5. NBDSSL mode is asynchronous if the ESXi host supports asynchronous NBD, as does ESXi 6.7.

NFC Session Limits and Timeouts

NBDSSL employs the network file copy (NFC) protocol. The table below shows limits on the number of connections for various host type. These are host limits, not per process limits.

Additionally vCenter Server imposes a limit of 52 connections. **`VixDiskLib_Open()`** uses one connection for every virtual disk that it accesses on an ESXi host. Clone with **`VixDiskLib_Clone()`** also requires a connection. It is not possible to share a connection across physical disks. These NFC session limits do not apply to SAN or HotAdd transport.

Table 105: NFC Session Connection Limits

Host Platform	When Connecting	Limits You To About
vSphere 3.5	to an ESX host	9 connections directly, 27 connections through vCenter Server

Table continued on next page

Continued from previous page

Host Platform	When Connecting	Limits You To About
vSphere 4.x	to an ESXi host	11 connections directly, 23 connections through vCenter Server
vSphere 5.x	to an ESXi host or vCenter Server	Limited by a transfer buffer for all NFC connections, enforced by the host; the sum of all NFC connection buffers to an ESXi host cannot exceed 32MB. Also 52 connections through vCenter Server, including the above per-host limit.
vSphere 6.x vSphere 7.0	to an ESXi host or vCenter Server	The sum of all NFC connection buffers cannot exceed 48MB (by default) for the NFC server in one ESXi host. With 1MB buffers per I/O stream, 48 requests can be concurrent. When customers use vCenter to manage ESXi hosts, the NFC memory limit is not configurable for backup. The NFC memory limit in ESXi hosts can be set with `config.xml` if there is no vCenter.

The `InitEx` configuration file parameter `vixDiskLib.nfc.ReadTimeoutMs` indicates the preferred connection timeout for backup and restore operations, and has the effect of influencing TCP keep-alive time when no reads occur, which they do not during prolonged restores. If the `ReadTimeoutMs` value is set higher than the TCP keep-alive time on the network, then a connection timeout could occur.

For example, if the configuration file sets `ReadTimeoutMs` flag to 3600000 milliseconds (60 minutes) instead of accepting the default (varies from 6 to 45 seconds) then a TCP keep-alive packet transmits only after 60 minutes. If restoring a virtual disk takes longer, leaving network read activity idle for 60 minutes, the connection could break so subsequent restores fail.

SSL Certificates and Security

The VDDK 5.1 release and later were security hardened, with virtual machines set to check SSL certificates.

On Windows VDDK 5.1 and 5.5 required the VerifySSLCertificates and InstallPath registry keys under `HKEY_LOCAL_MACHINE\SOFTWARE` to check SSL certificates. On Linux VDDK 5.1 and 5.5 required adding a line to the **`VixDiskLib_InitEx`** configuration file to set **`linuxSSL.verifyCertificates`** = 1.

As of VDDK 6.0 both SSL certificate verification and SSL thumbprint checking are mandatory and cannot be avoided. The Windows registry and Linux SSL setting are no longer checked, so neither has any effect.

Specifically VDDK 6.0 and later use X.509 certificates with TLS cryptography, replacing SSLv3.

The following library functions enforce SSL certificate checking: **`InitEx`**, **`PrepareForAccess`**, **`EndAccess`**, **`GetNfcTicket`**, and the **`GetRpcConnection`** interface that is used by all advanced transports. SSL verification may use thumbprints to check if two certificates are the same. The vSphere thumbprint is a cryptographic hash of a certificate obtained from a trusted source such as vCenter Server, and passed in the **`SSLVerifyParam`** structure of the NFC ticket.

Virtual Disk API Functions

This chapter provides an overview of functions in the Virtual Disk API and includes the following sections:

After a presentation of Virtual Disk API functions in alphabetic order, sections focus on what the functions do, in the normal order they would appear in a program, except advanced transport functions (SAN and HotAdd) appear after the shutdown functions.

Virtual Disk Library Functions

You can find the *VixDiskLib API Reference* by using a Web browser to open the `doc/index.html` file in the VDDK software distribution. As in most reference manuals, functions are organized alphabetically, whereas in this chapter, functions are ordered by how they might be called.

When the API reference says that a function supports “only hosted disks,” it means virtual disk images hosted by VMware Workstation or similar products. Virtual disk stored on VMFS partitions managed by ESXi or vCenter Server is called “managed disk.”

The functions described in this chapter are based on concepts and employ data structures documented in Virtual Disk Interfaces.

If the library accesses virtual disk on VMFS, I/O by default goes through the ESXi host, which manages physical disk storage. To use function calls that provide direct access to SAN storage, start your program by calling the **`VixDiskLib_ConnectEx()`** function, as described in Advanced Transport APIs.

Alphabetic Table of Functions

Function calls in the Virtual Disk API are listed alphabetically in Table : Virtual Disk API Functions.

Table 106: Virtual Disk API Functions

Function	Description
`VixDiskLib_Attach`	Attach the child disk chain to the parent disk chain.
`VixDiskLib_Cleanup`	Remove leftover transports. See Clean Up After Disconnect.
`VixDiskLib_Clone`	Copy virtual disk to some destination, converting formats as appropriate.
`VixDiskLib_Close`	Close an open virtual disk. See Close a Local or Remote Disk.
`VixDiskLib_Connect`	Connect to the virtual disk library to obtain services. See also **`ConnectEx`**.
`VixDiskLib_ConnectEx`	Connect to optimum transport. See Connect to VMware vSphere
`VixDiskLib_Create`	Create a virtual disk according to specified parameters.
`VixDiskLib_CreateChild`	Create a child disk (redo log or delta link) for a hosted virtual disk.
`VixDiskLib_Defragment`	Defragment the sectors of a virtual disk.
`VixDiskLib_Disconnect`	Disconnect from the library. See Disconnect from Server.
`VixDiskLib_EndAccess`	Notify a host that it may again relocate a virtual machine. See Prepare For Access and End Access.
`VixDiskLib_Exit`	Release all resources held by the library. See Clean Up and Exit.
`VixDiskLib_Flush`	Flush asynchronous write data to disk. Replaced by **`Wait`** function.
`VixDiskLib_FreeErrorText`	Free the message buffer allocated by GetErrorText.
`VixDiskLib_FreeInfo`	Free the memory allocated by GetInfo.
`VixDiskLib_GetErrorText`	Return the text description of a library error code.
`VixDiskLib_GetInfo`	Retrieve information about a virtual disk.

Table continued on next page

Continued from previous page

Function	Description
`VixDiskLib_GetMetadataKeys`	Retrieve all keys in the metadata of a virtual disk.
`VixDiskLib_GetTransportMode`	Get the current transport mode. See Get Selected Transport Method.
`VixDiskLib_Grow`	Increase size of an existing virtual disk.
`VixDiskLib_Init`	Initialize the old virtual disk library. Replaced by `InitEx` function.
`VixDiskLib_InitEx`	Initialize new virtual disk library. See Initialize Virtual Disk API.
`VixDiskLib_ListTransportModes`	Available transport modes. See List Available Transport Methods.
`VixDiskLib_Open`	Open a virtual disk. See Open a Local or Remote Disk.
`VixDiskLib_PrepareForAccess`	Notify a host to refrain from relocating a virtual machine. See Prepare For Access and End Access.
`VixDiskLib_Read`	Read from an open virtual disk. See Read Sectors From a Disk.
`VixDiskLib_ReadAsync`	Asynchronously read a range of sectors.
`VixDiskLib_ReadMetadata`	Retrieve the value of a given key from disk metadata.
`VixDiskLib_Rename`	Change the name of a virtual disk.
`VixDiskLib_Shrink`	Reclaim blocks of zeroes from the virtual disk.
`VixDiskLib_SpaceNeededForClone`	Compute the space required to clone a virtual disk, in bytes.
`VixDiskLib_Unlink`	Delete the specified virtual disk.
`VixDiskLib_Wait`	Wait for all asynchronous operations to complete.
`VixDiskLib_Write`	Write to an open virtual disk. See Write Sectors To a Disk.
`VixDiskLib_WriteAsync`	Asynchronously write a range of sectors.
`VixDiskLib_WriteMetadata`	Update virtual disk metadata with the given key/value pair.

Start Up

The `VixDiskLib_Init()` and `VixDiskLib_Connect()` functions must appear in all virtual disk programs.

`VixDiskLib_Init()` has been superseded by `VixDiskLib_InitEx()`. See Initialize Virtual Disk API.

Initialize the Library

`VixDiskLib_Init()` initializes the old virtual disk library. The arguments `majorVersion` and `minorVersion` represent the VDDK library's release number and dot-release number. The optional third, fourth, and fifth arguments specify log, warning, and panic handlers. DLLs and shared objects may be located in `libDir`.

```
VixError vixError = VixDiskLib_Init(majorVer, minorVer, &logFunc, &warnFunc, &panicFunc,
libDir);
```

You should call **VixDiskLib_Init()** only once per process because of internationalization restrictions, at the beginning of your program. You should call **VixDiskLib_Exit()** at the end of your program for cleanup. For multithreaded programs you should write your own **logFunc** because the default function is not thread safe.
In most cases you should replace **VixDiskLib_Init()** with **VixDiskLib_InitEx()**, which allows you to specify a configuration file. For information about **InitEx**, see Initialize Virtual Disk API.

Connect to a Workstation or Server

VixDiskLib_Connect() connects the library to either a local VMware host or a remote server. For hosted disk on the local system, provide null values for most connection parameters. For managed disk on vSphere, specify virtual machine name, vCenter Server or ESXi host, user name, password, and port number.

```
vixError = VixDiskLib_Connect(&cnxParams, &srcConnection)
```

Always call **VixDiskLib_Disconnect()** before the end of your program.
Calling **vixDiskLib_Disconnect()** invalidates any open file handles, so if you use the VixMntapi library, call **VixMntApi_CloseDiskSet()** before calling disconnect.
You can opt to use the **VixDiskLibSSPICreds** connection parameter to enable Security Support Provider Interface (SSPI) authentication. SSPI has the advantage of not storing plain text passwords in configuration files or in the registry. In order to be able to use SSPI, the following conditions must be met:

- Connections must be made directly to a vCenter Server instance.
- Applications and their connections must employ one of two user account arrangements. The connection must be established either using the same user context with the same user name and password credentials on both the proxy and the vSphere Server, or using a domain user. Attempts by applications to establish connections using the Local System account context will fail.
- User contexts must have administrator privileges on the proxy and have VCB Backup User role assigned in vSphere (by the vCenter Server).

If your setup meets all these conditions, you can enable SSPI authentication by setting **USERNAME** to **__sspi__**. For SSPI, the password must be set, but it is ignored. It can be set to `""` (null string).

VMX Specification

On VMware platform products, `.vmx` is a text file (usually located in the same directory as virtual disk files) specifying virtual machine configuration. The Virtual Machine eXecutable (VMX) process is the user-space component (or “world”) of a virtual machine. The virtual disk library connects to virtual machine storage through the VMX process.

When specifying connection parameters (see Data Structures in Virtual Disk API) the preferred syntax for **vmxSpec** is as follows:

- Managed object reference of the virtual machine, an opaque object that you obtain programmatically using the **PropertyCollector** managed object:
  `moRef=<moref-of-vm>`

  The **moRef** of a virtual machine or disk snapshot on an ESXi host is likely different than the **moRef** of the same virtual machine or disk snapshot as managed by vCenter Server. Here are two example **moRef** specifications, one for ESXi and one for vCenter Server, both referring to the same snapshot:

  ```
  moref=153
  ```

  ```
  moref=271
  ```

Disk Operations

These functions create, open, read, write, query, and close virtual disk.

Create a New Hosted Disk

VixDiskLib_Create() locally creates a new virtual disk, after being connected to the host. In **createParams**, you must specify the disk type, adapter, hardware version, and capacity as a number of sectors. This function supports hosted disk. For managed disk, first create a hosted type virtual disk, then use **VixDiskLib_Clone()** to convert the virtual disk to managed disk.

```
vixError =

VixDiskLib_Create(appGlobals.connection, appGlobals.diskPath, &createParams, NULL, NULL);
```

Currently **VixDiskLib_Create()** enforces a 4GB limit for virtual disks on FAT32 and FAT file systems, a 16TB - 54KB (hex FFFFFFF0000) limit on NTFS file systems, and a 2^64 - 1 limit (more than an exabyte) on exFAT file systems. Hosted virtual disk > 2TB is not supported.
POSIX based file systems including NFS version 3 no longer have a 2GB file size limit. Although various checks are done to avoid creating impossibly large files, it becomes the customer's responsibility to cope with 2GB limits on NFS version 2 or Linux kernel 2.4 (EFS).

Open a Local or Remote Disk

After the library connects to a workstation or server, **VixDiskLib_Open()** opens a virtual disk. With SAN or HotAdd transport, opening a remote disk for writing requires a pre-existing snapshot.

```
vixError =

VixDiskLib_Open(appGlobals.connection, appGlobals.diskPath, appGlobals.openFlags,
&srcHandle);
```

The following flags modify the open instruction:

- **VIXDISKLIB_FLAG_OPEN_UNBUFFERED** – Deactivate host disk caching.
- **VIXDISKLIB_FLAG_OPEN_SINGLE_LINK** – Open the current link, not the entire chain (hosted disk only).
- **VIXDISKLIB_FLAG_OPEN_READ_ONLY** – Open the virtual disk read-only.

As of vSphere 6.5, the following additional flags are available:

- **VIXDISKLIB_FLAG_OPEN_COMPRESSION_ZLIB** – Open for NBDSSL transport, **zlib** compression.
- **VIXDISKLIB_FLAG_OPEN_COMPRESSION_FASTLZ** – Open for NBDSSL transport, **fastlz** compression.
- **VIXDISKLIB_FLAG_OPEN_COMPRESSION_SKIPZ** – Open for NBDSSL transport, **skipz** compression.

Read Sectors From a Disk

VixDiskLib_Read() reads a range of sectors from an open virtual disk. You specify the beginning sector and the number of sectors. Sector size could vary, but is defined in **<vixDiskLib.h>** as 512 bytes because VMDK files have that sector size.

```
vixError = VixDiskLib_Read(srcHandle, i, j, buf);
```

In vSphere 6.7 and later, you can improve performance of NBD transport with asynchronous reads. Asynchronous calls may be used for all transport modes, because they fall back to synchronous as needed.

```
// customized callback for complete notification

void myDiskLibCompletion(void *cbData, VixError result);

// a loop for multiple read requests
```

```
for (...)
   vixError = VixDiskLib_ReadAsync(srcHandle, i, j, buf, myDiskLibCompletion, cbData);
   if (vixError != VIX_ASYNC) {
      // handle error
   }
}
VixDiskLib_Wait(srcHandle); // wait ror all async read to complete
```

Write Sectors To a Disk

VixDiskLib_Write() writes one or more sectors to an open virtual disk. This function expects the fourth parameter **buf** to be **VIXDISKLIB_SECTOR_SIZE** bytes long.

```
vixError = VixDiskLib_Write(newDisk.Handle(), i, j, buf);
```

In vSphere 6.7 and later, you can improve performance of NBD transport with asynchronous writes. Your program **VixDiskLib_WriteAsync** in a loop, then calls **VixDiskLib_Wait** to let all asynchronous writes complete.

```
// customized callback for complete notification
void myDiskLibCompletion(void *cbData, VixError result);
// a loop for multiple write requests
for (...)
   vixError = VixDiskLib_WriteAsync(newDisk.Handle(), i, j, buf, myDiskLibCompletion,
cbData);
   if (vixError != VIX_ASYNC) {
      // handle error
   }
}
VixDiskLib_Wait(newDisk.Handle()); // wait for async write to complete
```

Close a Local or Remote Disk

VixDiskLib_Close() closes an open virtual disk.

```
VixDiskLib_Close(srcHandle);
```

VixDiskLib_Close() returns **VIX_OK** if successful, otherwise a suitable error code. To obtain a list of possible return codes, see Finding Error Code Documentation.
If a program has worker threads called from a master process, errors might occur in the threads after the master process calls **VixDiskLib_Close().** Always wait for worker threads to end before calling close.

Get Information About a Disk

```
vixError = VixDiskLib_GetInfo(srcHandle, diskInfo);
```

VixDiskLib_GetInfo() gets data about an open virtual disk, allocating a filled-in **VixDiskLibDiskInfo** structure. Some of this information overlaps with metadata (see Metadata Handling).

Free Memory from Get Information

This function deallocates memory allocated by **VixDiskLib_GetInfo()**. Call it to avoid a memory leak.

```
vixError = VixDiskLib_FreeInfo(diskInfo);
```

Metadata Handling

VMware provides mechanisms for reading, writing, and repairing virtual disk metadata.

Read Metadata Key from Disk

```
vixError = VixDiskLib_ReadMetadata(disk.Handle(), appGlobals.metaKey, &val[0],
requiredLen, NULL);
```

Retrieves the value of a given key from disk metadata. The metadata for a hosted VMDK is not as extensive as for managed disk on an ESXi host. Held in a mapping file, VMFS metadata might also contain information such as disk label, LUN or partition layout, number of links, file attributes, locks, and so forth. Metadata also describes encapsulation of raw disk mapping (RDM) storage, if applicable.

Get Metadata Table from Disk

VixDiskLib_GetMetadataKeys() retrieves all existing keys from the metadata of a virtual disk, but not the key values. Use this in conjunction with **VixDiskLib_ReadMetadata()**.

```
vixError = VixDiskLib_GetMetadataKeys(disk.Handle(), &buf[0], requiredLen, NULL);
```

Here is an example of a simple metadata table. **Uuid** is the universally unique identifier for the virtual disk.

```
adapterType = buslogic

geometry.sectors = 32

geometry.heads = 64

geometry.cylinders = 100

uuid = 60 00 C2 93 7b a0 3a 03-9f 22 56 c5 29 93 b7 27
```

Write Metadata Table to Disk

VixDiskLib_WriteMetadata() updates virtual disk metadata with the given key-value pair. If the key-value pair is new, it gets added. If the key already exists, its value is updated. A key can be zeroed but not deleted.

```
vixError = VixDiskLib_WriteMetadata(disk.Handle(), appGlobals.metaKey,
appGlobals.metaVal);
```

Check and Repair Sparse Disk Metadata

VixDiskLib_CheckRepair() checks the metadata of a sparse disk, specified as a file on a connection, and optionally repairs the metadata if necessary. Sparse disks occupy space on a datastore only when a portion of the disk is used; the metadata tracks which portions are allocated.

```
vixError = VixDiskLib_CheckRepair(appGlobals.connection, appGlobals.diskPath, TRUE);
```

Disk Chaining and Redo Logs

In VMDK terminology, all the following are synonyms: child disk, redo log, and delta link. From the original parent disk, each child constitutes a redo log pointing back from the present state of the virtual disk, one step at a time, to the original. This pseudo equation represents the relative complexity of backups and snapshots:

backup image < child disk = redo log = delta link < snapshot

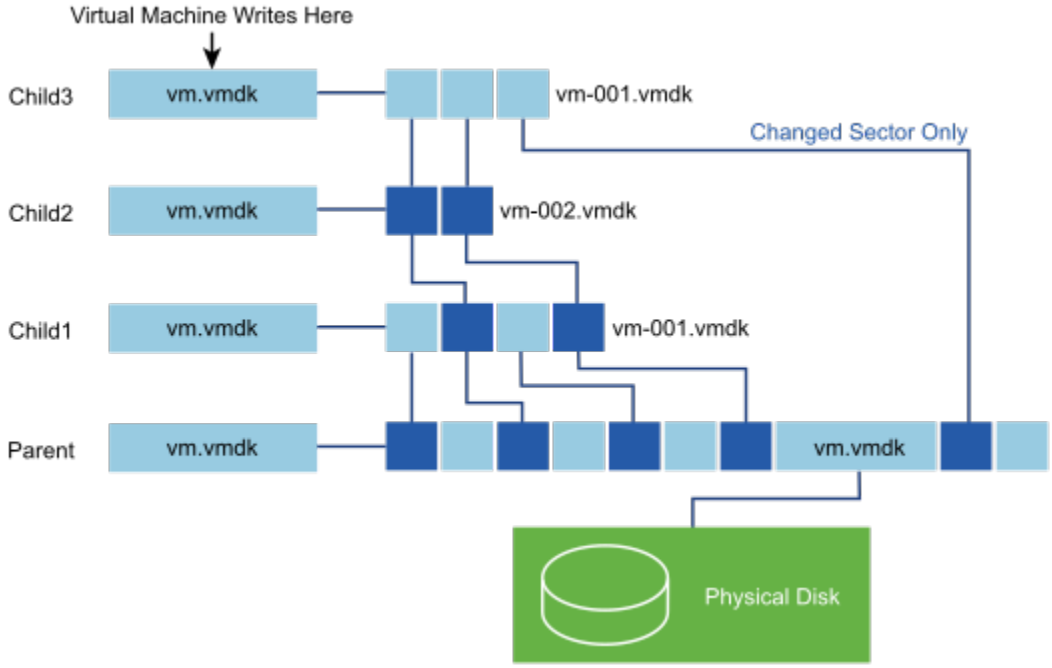
A backup image is less than a child disk because a backup image is merely a data stream. A snapshot is more than a child disk because it contains virtual machine state, with pointers to associated file states on VMDK.

There exist other types of redo logs, such as those that perform progressive protection in vSphere Replication (VR). For disk chaining, the “redo” terminology is especially appropriate for the snapshot revert operation, when changed blocks in the redo log are applied to the base disk, before deleting the redo log. Afterwards the base disk contains a “redo” of all changes that the virtual machine made while the snapshot was active.

Create Child from Parent Disk

Usually you create the first child disk from the parent and create successive children from the latest one in the chain. The disk tracks, in **`SPARSE`** format, any disk sectors changed since inception, as illustrated below.

Figure 172: Child disks created from parent



`VixDiskLib_CreateChild()` creates a child disk (or redo log) for a hosted virtual disk. After you create a child, it is generally not necessary to open the parent, or earlier children in the disk chain. The children’s `vm.vmdk` files point to redo logs, not to the parent disk, `vm-flat.vmdk` in this example. To access the original parent, or earlier children in the chain, you can use **`VixDiskLib_Attach()`** on hosted disk.

```
vixError = VixDiskLib_CreateChild(parent.Handle(), appGlobals.diskPath,

              VIXDISKLIB_DISK_MONOLITHIC_SPARSE, NULL, NULL);
```

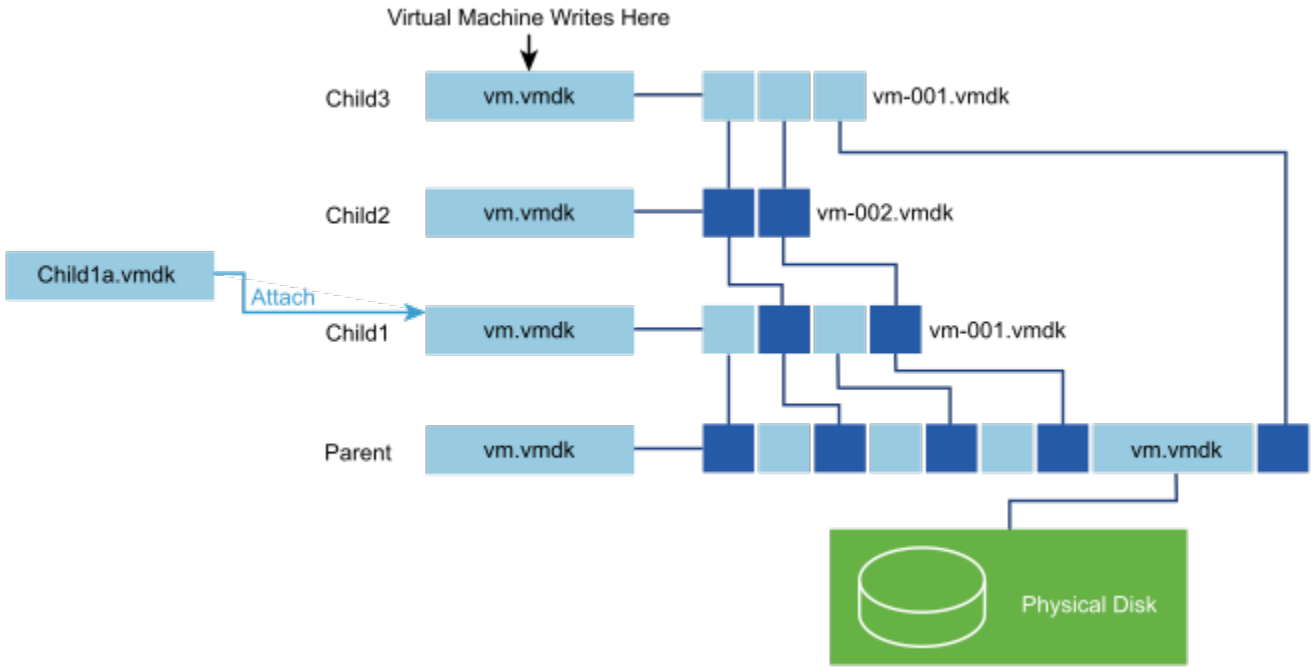
Attach Child to Parent Disk

`VixDiskLib_Attach()` attaches the child disk into its parent disk chain. Afterwards, the parent handle is invalid and the child handle represents the combined disk chain of redo logs. On failure (`vixError!=VIX_OK`) the parent handle is also invalid, so do not close it.

```
vixError = VixDiskLib_Attach(parent.Handle(), child.Handle());
```

For example, suppose you want to access the older disk image recorded by Child1. Attach the handle of new Child1a to Child1, which provides Child1a's parent handle, as shown below. It is now permissible to open, read, and write the Child1a virtual disk.
The parent-child disk chain is efficient in terms of storage space, because the child VMDK records only the sectors that changed since the last **`VixDiskLib_CreateChild()`**. The parent-child disk chain also provides a redo mechanism, permitting programmatic access to any generation with **`VixDiskLib_Attach()`**.

Figure 173: Child disk attached to disk chain



Before VDDK 6.7.1 it was an error to close **`parentHandle`** after **`VixDiskLib_Attach`** succeeds. The VDDK library now marks **`parentHandle`** internally to prevent closure and ensure cleanup. Here is the calling sequence for open and attach:

1. Open the disk for attach.
2. Create a local connection.
3. With backed-up disk (referred to as the parent disk) still open, create child disk with a unique name.
4. Open uniquely named `tmp.vmdk` (referred to as the redo log).
5. Attach the redo log to its parent disk.

```
VixDiskLib_Open(remoteConnection, virtualDiskPath, flags, &parentHandle);

VixDiskLib_Connect(NULL, &localConnection);

VixDiskLib_CreateChild(parentHandle, "C:\tmp.vmdk", VIXDISKLIB_DISK_MONOLITHIC_SPARSE,
NULL, NULL);

VixDiskLib_Open(localConnection, "C:\tmp.vmdk", VIXDISKLIB_FLAG_OPEN_SINGLE_LINK,
&redoHandle);

VixDiskLib_Attach(parentHandle, redoHandle);
```

Here is the calling sequence for close:

1. Close the redo log. Whether to close the parent disk handle is release dependent.
2. Unlink the redo log to detach it from the parent disk.

```
VixDiskLib_Close(redoHandle);

if (VIXDISKLIB_VERSION_MAJOR > 7) {

   VixDiskLib_Close(parentHandle); // to avoid memory leaks

}

VixDiskLib_Unlink(localConnection, "C:\tmp.vmdk");
```

Opening in a Chain

With (parent) base disk B and children C0, C1, and C2, opening C2 gives you the contents of B + C0 + C1 + C2 (not really addition linked data sectors), while opening C1 gives you the contents of B + C0 + C1.

A better solution than recording base disks and which children are descended from which is changed block tracking, **`QueryChangedDiskAreas`** in the vSphere API. See Algorithm for vSphere Backup.

Redo Logs and Linked Clone Backup

For managed virtual disk on vSphere, snapshots are used primarily for saving system state and for backup, while linked clones create duplicate images for provisioning of View desktops. A snapshot is usually a single redo log in a parent-child chain, while linked clones are usually multiple redo logs based on the same parent.

In the vSphere 5.5 release and later, handling of linked clone hierarchies was changed to improve the efficiency of backup and restore. The disk object now contain a “disk backing” that contains one or more parent backing objects until the base disk is reached. This allows access anywhere in the parent-child disk chain. With a clean never-used base virtual machine, the linked clone hierarchy or snapshot chain always has the proper number of parent backing objects for the nodes in the chain.

VDDK does not contain any convenience interfaces for backing up and restoring the linked clone hierarchy (or the snapshot chain). Backup applications are responsible for discovering and saving the hierarchy if they want to support this as a feature. Linked clones cannot be restored using SAN transport.

In VMware View (VDI) environments, linked clone backup might not be necessary or advisable, especially for nonpersistent desktops that revert to default after use.

When the base disk or a child disk has an extra snapshot, when redo logs used to create linked clones were never deleted, or when any parent or child in the chain needs disk consolidation or is in a bad snapshot state, it is possible to have extra (too many) parent backing objects. Consequently, restore applications should never assume the correct number of parent backing objects. They must recursively query until the base parent backing object is reached, and make sure when restoring leaf nodes that the correct parent backing object matches the node being restored.

Cloning a Virtual Disk

On ESXi hosts in vSphere, programs cannot create managed disk, but they can call `vixDiskLib_Clone()` to convert hosted disk to managed disk.

Compute Space Needed for Clone

This function computes the space required (in bytes) to clone a virtual disk, after possible format conversion.

```
vixError = VixDiskLib_SpaceNeededForClone(child.Handle(), VIXDISKLIB_DISK_VMFS_FLAT,
&spaceReq);
```

CAUTION

VixDiskLib_SpaceNeededForClone() might not give accurate results, or could return **VIX_E_INVALID_ARG**, when used with thin provisioned disk type **VIXDISKLIB_DISK_VMFS_THIN**.

Clone a Disk by Copying Data

This function copies data from one virtual disk to another, converting (disk type, size, hardware) as specified.

```
vixError = VixDiskLib_Clone(appGlobals.connection, appGlobals.diskPath, srcConnection,
               appGlobals.srcPath, &createParams, CloneProgressFunc, NULL, TRUE);
```

Due to a vSphere 6.5 security enhancement, `VixDiskLib_Clone()` can no longer clone local disks to remote disks that do not belong to any VM. Previous releases allowed use of a retained username and password, but now enhanced security verification is required. The procedure for `VixDiskLib_Clone()` to make a remote connection is first to get an NFC ticket from the ESXi host based on the `MoRef` of the VM that manages the cloned-to disks. Then build an authenticated NFC connection to the host using that ticket. The `MoRef` of the VM must be specified in `vmxSpec` of the remote connection parameter passed to `VixDiskLib_Clone()`. The VM must be powered off, and the target disks must already exist before cloning, not necessarily with the same names. Afterwards the VM may be removed from the inventory with `UnregisterVM`, but it must remain on the datastore with its cloned virtual disks.

Error Handling

These functions enhance the usefulness of error messages.

Return Error Description Text

VixDiskLib_GetErrorText() returns the textual description of a numeric error code.

```
char* msg = VixDiskLib_GetErrorText(errCode, NULL);
```

Free Error Description Text

VixDiskLib_FreeErrorText() deallocates space associated with the error description text.

```
VixDiskLib_FreeErrorText(msg);
```

Administrative Disk Operations

These functions rename, grow, defragment, shrink, and remove virtual disk.

Rename an Existing Disk

VixDiskLib_Rename() changes the name of a virtual disk. Use this function only when the virtual machine is powered off.

```
vixError = VixDiskLib_Rename(oldGlobals.diskPath, newGlobals.diskPath);
```

Grow an Existing Local Disk

VixDiskLib_Grow() extends an existing virtual disk by adding sectors. This function supports hosted disk, but not managed disk.

```
vixError =

VixDiskLib_Grow(appGlobals.connection, appGlobals.diskPath, size, FALSE, GrowProgressFunc,
NULL);
```

Defragment an Existing Disk

VixDiskLib_Defragment() defragments an existing virtual disk. Defragmentation is effective with **SPARSE** type files, but might not do anything with **FLAT** type. In either case, the function returns **VIX_OK**. This function supports hosted disk, but not managed disk.

```
vixError = VixDiskLib_Defragment(disk.Handle(), DefragProgressFunc, NULL);
```

Defragment consolidates data in the 2GB extents, moving data to lower-numbered extents, and is independent of defragmentation tools in the guest OS, such as Disk > Properties > Tools > Defragmentation in Windows, or the **defrag** command for the Linux Ext2 file system.
VMware recommends defragmentation from the inside out: first within the virtual machine, then using this function or a VMware defragmentation tool, and finally within the host operating system.

Shrink an Existing Local Disk

VixDiskLib_Shrink() reclaims unused space in an existing virtual disk, unused space being recognized as blocks of zeroes. This is more effective (gains more space) with **SPARSE** type files than with pre-allocated **FLAT** type. On success, the function returns **VIX_OK**. This function supports hosted disk, but not managed disk.

```
vixError = VixDiskLib_Shrink(disk.Handle(), ShrinkProgressFunc, NULL);
```

In VMware system utilities, “prepare” zeros out unused blocks in the VMDK so “shrink” can reclaim them. In the API, use **VixDiskLib_Write()** to zero out unused blocks, and **VixDiskLib_Shrink()** to reclaim space. Shrink does not change the virtual disk capacity, but it makes more space available.

Unlink Extents to Remove Disk

VixDiskLib_Unlink() deletes all extents of the specified virtual disk, which unlinks (removes) the disk data. This is similar to the remove or erase command in a command tool.

```
vixError = VixDiskLib_Unlink(appGlobals.connection, appGlobals.diskPath);
```

Shut Down

All Virtual Disk API applications should call these functions at end of program.

Disconnect from Server

VixDiskLib_Disconnect() breaks an existing connection.

```
VixDiskLib_Disconnect(srcConnection);
```

Clean Up and Exit

VixDiskLib_Exit() cleans up the library before exit.

```
VixDiskLib_Exit();
```

Advanced Transport APIs

For managed disk, the first release of VDDK required network access ESXi host (LAN or NBDSSL transport). With VDDK 1.1 programs can access virtual disks directly on a storage device, LAN-free. Direct SAN access increases I/O performance. To select the most efficient transport method, a set of APIs is available, including:

- **VixDiskLib_InitEx()** – Initializes the advanced transport library. You must specify the library location. Replaces **VixDiskLib_Init()** in your application.
- **VixDiskLib_ListTransportModes()** – Lists transport modes that the virtual disk library supports.

- **VixDiskLib_ConnectEx()** – Establishes a connection using the best transport mode available, or one you select, to access a given machine's virtual disk. Currently it does not check validity of transport type. Replaces **VixDiskLib_Connect()** in your application.

Initialize Virtual Disk API

VixDiskLib_InitEx() replaces **VixDiskLib_Init()** to initialize new releases of the library. Parameters are similar, except you should specify an actual **libDir**, and the **configFile** parameter is new. For multithreaded programming, you should write your own **logFunc**, because the default logging function is not thread-safe. VixDiskLib loads many libraries (DLLs or shared objects) at start time using the **Path** or **LD_LIBRARY_PATH** environment. At run time, **libDir** specifies where to load advanced transport modes, allowed list, and denied list. On Windows, ***libDir** could be C:\Program Files\VMware\VMware Virtual**Disk DevelopmentKit**. On Linux, ***libDir** could be /usr/lib/vmware-vix-disklib.

```
VixError vixErr = VixDiskLib_InitEx(majorVersion, minorVersion, &logFunc, &warnFunc,
&panicFunc, *libDir, *configFile);
```

Logged messages appear by default in a temporary folder or log directory, for VDDK and for many other VMware products. See Location of Log Files.
The currently supported entries in the **configFile** are listed below. The correct way to specify a configuration is **name=value**. See below for a sample configuration file.

- **tmpDirectory = "<TempDirectoryForLogging>"** – Programs should set this because VDDK's default temporary folder can change from backup job to job, which could cause cleanup issues.
- **vixDiskLib.transport.LogLevel** – Overrides default logging for **vixDiskLib** advanced transport functions, not including NFC (see **nfc.LogLevel** below). The default value is 3, indicating audit. Each level includes all of the messages generated by lower numbered levels. Here are the levels:
  ```
  0 = Panic (failure only)

  1 = Error

  2 = Warning

  3 = Audit

  4 = Info

  5 = Verbose

  6 = Trivia
  ```
- **vixDiskLib.transport.san.denyList** – Specifies a device node path, or a comma separated list of device node paths, that VDDK uses as a list of LUNs on which not to attempt VMFS file system discovery. This has the effect of making fewer device nodes available to SAN transport, and avoids I/O on specific device node paths. The special string **all** indicates that VDDK should use only the device node paths specified by **vixDiskLib.transport.san.allowList**.
- **vixDiskLib.transport.san.allowList** – Specifies a device node path, or a comma separated list of device node paths, that VDDK uses as a list of LUNs on which to attempt VMFS file system discovery. This has the effect of making more device nodes available to SAN transport, encouraging I/O on specific device node paths. Backup applications may create a special device node and allow this device node for use in addition to those found by VDDK's device node scanner. Backup applications can also disallow specific devices found by VDDK's device node scanner to prevent use by SAN transport. Combining allowed and denied lists, applications can establish a preferred device policy for backup I/O.
- **vixDiskLib.disklib.EnableCache** – Caching by **vixDiskLib** is off (0) by default. Setting 1 turns it on. Caching increases performance when information is read repeatedly, or accessed randomly. In backup applications, information is usually accessed sequentially, and caching can actually reduce performance. Moreover with caching, backup applications risk getting stale information if a disk sector is rewritten (by another application) before the cache is refreshed.

- **vixDiskLib.linuxSSL.verifyCertificates** – Whether to check SSL certificates and thumbprints when connecting to a virtual machine. The only valid value is 1 for On, which is the default.
- **vixDiskLib.ssl.enableSslFIPS** – Set this value to 1 (for On) to enable FIPS validated cryptography with OpenSSL.

The following NFC related options override the default numbers provided to the various NFC functions. The NFC timeouts shown in the example (below **LogLevel**) correspond to default values on ESXi 5.x hosts.

- **vixDiskLib.nfc.AcceptTimeoutMs** – Overrides default value (3 minutes) for NFC accept operations.
- **vixDiskLib.nfc.RequestTimeoutMs** – Overrides default value (3 minutes) for NFC request operations.
- **vixDiskLib.nfc.ReadTimeoutMs** – Overrides default value (one minute) for NFC read operations. Similar to TCP keep-alive interval. See NFC Session Limits and Timeouts.
- **vixDiskLib.nfc.WriteTimeoutMs** – Overrides default value (ten minutes) for NFC write operations.
- **vixDiskLib.nfc.LogLevel** – Overrides the default logging level for NFC operations. The default value is 1, indicating error messages only. The meaning of values is listed here. Each level includes all of the messages generated by lower numbered levels. This is the final NFC setting.

  ```
  0 = Quiet (minimal loggin)

  1 = Errpr

  2 = Warning

  3 = Info

  4 = Debug
  ```

The `nfcAio` configurations set socket options. Each socket has a send buffer area and a receive buffer area in the kernel. The receive buffer is used to cache data from the network until the application process reads it. The send buffer is used to cache data that needs to be sent to the network. Appropriate buffer sizes help maximize network throughput and performance. On Windows, `bufSize` has significant impact on backup and restore performance of NBD(SSL) transport. On Linux, the effect may be trivial.
vixDiskLib.nfcAio.SocketOption.SndBufSize – Set the maximum send buffer size for NFC client.
vixDiskLib.nfcAio.SocketOption.RcvBufSize – Set the maximum receive buffer size for NFC client.
vixDiskLib.nfcAio.SocketOption.ServerSndBufSize – Set the maximum send buffer size for NFC server.
vixDiskLib.nfcAio.SocketOption.ServerRcvBufSize – Set the maximum receive buffer size for NFC server.
Here is a sample InitEx configuration file:

```
# temporary directory for logs etc.

tmpDirectory="/usr/local/vendorapp/var/vmware/temp"

# log level 0 to 6 for quiet ranging to verbose

vixDiskLib.transport.LogLevel=2

# deactivate caching to disk

vixDiskLib.disklib.EnableCache=0

# whether to check SSL thumbprint on Linux - has no effect

vixDiskLib.linuxSSL.verifyCertificates=1

# network file copy options

vixDiskLib.nfc.AcceptTimeoutMs=180000

vixDiskLib.nfc.RequestTimeoutMs=180000
```

```
vixDiskLib.nfc.ReadTimeoutsMs=60000

vixDiskLib.nfc.WriteTimeoutsMs=600000

# nfc.LogLevel (0 = Quiet, 1 = Error, 2 = Warning, 3 = Info, 4 = Debug)

vixDiskLib.nfc.LogLevel=2
```

Timeout values are stored in a 32-bit field, so the maximum timeout you may specify is 2G (2,147,483,648). Timeouts are specified in milliseconds and apply to each disk handle. NFC settings apply to NBD and NBDSSL, but not to SAN or HotAdd.

Wildcards in SAN Allow and Deny Lists

After configuring `allowList` and `denyList` for SAN transport mode, customers requested a wildcard mechanism for ease of use. VDDK 8.0.1 and later support wildcards to simplify list making. You can use meta character expressions * ? [xyz] [a-z] [0-9] with the same meanings as in Bash shell.

In case of duplicates or ambiguity, `vixDiskLib.transport.san.allowlist` takes precedence over `vixDiskLib.transport.san.denylist`. Wildcards match neither directory separator slash (/) nor folder separator backslash (\). On Windows, deny and allow lists must contain "`\\?\`" or "`\\.\`*`PhysicalDrive`*" prefix, and sharp (#) must be inside double-quotes to avoid interpretation as a comment; `allowList="\\?\scsi#disk*"` for example.

Metacharacters	Examples of wildcard use
*	Match the preceding character or subexpression zero or more times, for example /dev/* matches /dev/, /dev/ab, and /dev/cde
?	Match any single character, for example /dev/sd? matches /dev/sda and /dev/sdb
[a-z]	Match a character in the specified range, for example /dev/test[1-2] matches /dev/test1 and /dev/test2
[xyz]	Match any character included in the set, for example /dev/sd[abc] matches /dev/sda, /dev/sdb, and /dev/sdc

Example of allow and deny lists for Linux:

```
vixDiskLib.transport.san.denylist="/dev/sd*"

vixDiskLib.transport.san.allowlist="/dev/sd[a-f]"
```

Example of allow and deny lists for Windows:

```
vixDiskLib.transport.san.allowlist="\\?\scsi#disk&ven_scst_bio&prod_disk[1-3]*"

vixDiskLib.transport.san.denylist="\\?\scsi#disk&ven_scst_bio&prod_disk4*"
```

How to Enable FIPS on VDDK Proxy VM

Open SSL 3.0 contains a module that conforms to Federal Information Processing Standards (FIPS), so the proxy VM must run VDDK 8.0.1 or later for Open SSL 3.0 support.

To prepare a VDDK proxy VM to run FIPS, follow these steps:

1. Location of the FIPS dynamic library differs. To install on Windows or Linux, run one of these commands:
   ```
   openssl.exe fipsinstall -out \path\of\fipsmodule.cnf -module VDDKpackage\bin\fips.dll
   ```
   ```
   openssl fipsinstall -out /path/of/fipsmodule.cnf -module VDDKpackage/lib64/fips.so
   ```
2. In the Open SSL configuration file, dot-include `fipsmodule.cnf` must be updated with an absolute path, and other values should be set as in this example:
   ```
   openssl_conf = openssl_init

   .include /path/of/fipsmodule.cnf

   [openssl_init]

   providers = provider_sect

   alg_section = algorithm_sect

   [provider_sect]

   default = default_sect

   fips = fips_sect

   [default_sect]

   activate = 1

   [algorithm_sect]

   default_properties = "fips=yes"
   ```
3. Set environment variable `OPENSSL_CONF` to the path of the Open SSL configuration file. Set environment variable `OPENSSL_MODULES` to the path of `fips.dll` or `fips.so`, as above.
4. Before VixDiskLib initialization, add `vixDiskLib.ssl.enableSslFIPS=1` to the VDDK configuration file.
5. With FIPS enabled, the VDDK information log will record “SSL is in FIPS mode” when `VixDiskLibInitEx()` is called.

Phone Home Support

The Customer Experience Improvement Program (CEIP) helps VMware understand what products and versions have been installed in a datacenter. This information is collected to assist and streamline problem determination and resolution.

Whether this information gets sent to VMware is a customer choice (on-premise) with CEIP opt-in and opt-out, or as part of their Terms of Service. VMware collects the product name, version, and transport to assist in problem resolution. No other partner-specific information is collected.

VMware asks that vendors add these lines to the VDDK configuration file for identification. Legal characters: 26 letters, digits, underscore (_), minus (-), period (.), and space. Double quotes are needed if settings contain spaces. Optional `DatabaseDir` stores phone home data in a separate folder.

```
vixDiskLib.phoneHome.ProductName = vendorName or ApplicationName
```

```
vixDiskLib.phoneHome.ProductVersion = versionNumber
```

```
vixDiskLib.phoneHome.DatabaseDir = folderName
```

Using the vSphere Client, customers can join or leave CEIP by clicking **Menu > Administration > Customer Experience Improvement Program**. When customers leave the CEIP, they deactivate phone-home for all products, including VDDK.

Location of Log Files

On Linux, log messages appear under `/var/log` by default. On Windows, they appear in a temporary folder, whose location can change from time to time. Early Windows systems used `C:\Windows\Temp`. Windows XP and Server 2003 use `C:\Documents andSettings\<user>\Local Settings\Temp\vmware-<user>`. Vista, Windows 7, and Server 2008 use `C:\Users\<user>AppData\Local\Temp\vmware-<user>`.

On all versions of Windows the user's **`TEMP`** environment setting overrides the default Temp folder location. Temporary is something of a misnomer because files are never deleted from the Temp folder, unless the user or an application deletes them. If the **`TEMP`** or Windows default Temp folder is not found, VDDK (and other VMware software) have a fallback to `<localAppDir>\Temp`.

Alternatively, your software can set a custom temporary directory, as shown in the sample InitEx file.

List Available Transport Methods

The **`VixDiskLib_ListTransportModes()`** function returns the currently supported transport methods as a colon-separated string value, currently "**`file:san:hotadd:nbdssl`**" where **`nbdssl`** indicates LAN transport. The default transport priority over the network is **`san:hotadd:nbdssl`** assuming all are available.

```
printf("Transport methods: %s\n", VixDiskLib_ListTransportModes());
```

Connect to VMware vSphere

`VixDiskLib_ConnectEx()` connects the library to managed disk on a remote ESXi host or through VMware vCenter Server. For hosted disk on the local system, it works the same as **`VixDiskLib_Connect()`**. **`VixDiskLib_ConnectEx()`** takes three additional parameters:

- Boolean indicating **`TRUE`** for read-only access, often faster, or **`FALSE`** for read/write access. If connecting read-only, later calls to **`VixDiskLib_Open()`** are always read-only regardless of the **`openFlags`** setting.
- Managed object reference (**`MoRef`**) of the snapshot to access using this connection. This is required for most transport methods (SAN, HotAdd, NBDSSL) and to access a powered-on virtual machine. You must also specify the associated **`vmxSpec`** property in **`connectParams`**. When connecting to an ESXi host, provide the ESXi **`MoRef`**. When connecting by vCenter Server, pass the vSphere **`MoRef`**, which differs.
- Preferred transport method, or **`NULL`** to accept the defaults. If you specify any advanced transport mode as the only method, and that method is not available, the **`VixDiskLib_ConnectEx()`** call does not fail, but the subsequent **`VixDiskLib_Open()`** call will fall back to NBDSSL mode.

```
VixDiskLibConnectParams cnxParams = {0};

if (appGlobals.isRemote) {

      cnxParams.vmName = vmxSpec;

      cnxParams.serverName = hostName;

      cnxParams.credType = VIXDISKLIB_CRED_UID;

      cnxParams.creds.uid.userName = userName;

      cnxParams.creds.uid.password = password;

      cnxParams.port = port;

}

VixError vixError = VixDiskLib_ConnectEx(&cnxParams, TRUE, "snapshot-47", NULL,
&connection);
```

When a program calls **`VixDiskLib_ConnectEx()`** with NULL parameter to accept default transport mode, SAN is selected as the preferred mode, if SAN storage is available from the ESXi host. Then if the program opens a virtual disk on local storage, subsequent writes will fail. In this case, the program should explicitly pass **`nbdssl`** as the preferred transport mode.

The port is where vCenter Server listens for API queries. Specifying null allows the library to select the port, usually 443 (HTTPS). By default VADP uses the same port for virtual machine operations as other SOAP-based Web Services. By default VDDK uses port 902 (VIX automation) for NBDSSL data transport.

Connect to ESXi hosts

In the connection parameters **`cnxParams`**, the **`vmxSpec`** managed object reference would be different on an ESXi host than on the vCenter Server, as shown below. ESXi hosts offer no prepare-for and end access protection. Otherwise ESXi host connections are similar to vCenter Server connections.

```
vmxSpec = "moid=23498";
```

```
vmxSpec = "moid=898273";
```

Reuse a vCenter Server Session

As of vSphere 6.5, you can recycle a vCenter Server session to avoid session overflow. Set the credential type to **`VIXDISKLIB_CRED_SESSIONID`** and supply the value of **`vmware_soap_session`** from a still-live vCenter Server session. For NBD(SSL) and HotAdd transport, the `sessionId.key` password can be any non-empty string. For SAN transport, you must specify the actual password.

```
if (appGlobals.isRemote) {
     cnxParams.vmName = vmxSpec;
     cnxParams.serverName = hostName;
     cnxParams.credType = VIXDISKLIB_CRED_SESSIONID;
     cnxParams.creds.sessionId.cookie = cookie;
     cnxParams.creds.sessionId.userName = userName;
     cnxParams.creds.sessionId.key = "reuse"; /* password */
     cnxParams.port = port;
}
```

Get Selected Transport Method

The **`VixDiskLib_GetTransportMode()`** function returns the transport method selected for **`diskHandle`**.

```
printf("Selected transport method: %s\n", VixDiskLib_GetTransportMode(diskHandle));
```

Prepare For Access and End Access

The **`VixDiskLib_PrepareForAccess()`** function notifies a vCenter-managed host that a virtual machine's disks are being opened, probably for backup, so the host should postpone virtual machine operations that might interfere with virtual disk access. Call this function before creating a snapshot on a virtual machine. Internally, this function deactivates the vSphere API method **`RelocateVM_Task`**.

```
vixError = VixDiskLib_PrepareForAccess(&cnxParams, "vmName");
```

The connection parameters must indicate one virtual machine only. When opening a managed disk, provide valid credentials for the vCenter Server that manages the ESXi host with the disk. The second parameter is currently just for identity tracking purposes, and is limited to 50 characters. It could be the virtual machine name or the name of your application. If you run **`VixDiskLib_PrepareForAccess()`** directly on an ESXi host, the system throws an error saying "VDDK: HostAgent is not a vCenter, cannot deactivate svMotion."
Every **`VixDiskLib_PrepareForAccess()`** call should have a matching **`VixDiskLib_EndAccess()`** call.
The **`VixDiskLib_EndAccess()`** function notifies the host that a virtual machine's disks have been closed, so operations that rely on the virtual disks to be closed, such as vMotion, can now be allowed. Call this function after closing all the virtual disks, and after deleting the virtual machine snapshot. Normally this function is called after previously calling **`VixDiskLib_PrepareForAccess`**, but you can call it to clean up after a crash. Internally, this function re-enables the vSphere API method **`RelocateVM_Task`**.

```
vixError = VixDiskLib_EndAccess(&cnxParams, "vmName");
```

Here is a code snippet showing use of **`PrepareForAccess`** in a backup program that waits up to 10 minutes for Storage vMotion to finish. Regular vMotion would finish much faster than that.

```
/* New sample code accounts for VMODL_TYPE_VIM_FAULT_METHOD_ALREADY_DISABLED_FAULT */

if (appGlobals.vmxSpec != NULL) {

     for (int i = 0; i < 10; i++) {

          vixError = VixDiskLib_PrepareForAccess(&cnxParams, "Sample");

          if (vixError == VIX_OK) {

              break;

          } else {

              Sleep(60000);

          }

     }

}
```

SAN Mode on Linux Uses Direct Mode

With SAN transport on Linux, read and write operations are performed in "direct" mode (**`O_DIRECT`**), meaning that no read or write buffering is done. Direct mode prevents other processes from accessing the latest data, and avoids loss of information if the process dies before committing its write buffers. In direct mode, the most time efficient performance can be achieved if applications follow these guidelines when performing reads and writes. These guidelines are applicable to Windows also.

- The buffer used for data transfer should be aligned on a page boundary.
- The transfer length should be an even multiple of the page size.

Clean Up After Disconnect

If virtual machine state was not cleaned up correctly after connection shut down, **`VixDiskLib_Cleanup()`** removes extra state for each virtual machine. Its three parameters specify connection, and pass back the number of virtual machines cleaned up, and the number remaining to be cleaned up.

```
int numCleanedUp, numRemaining;

VixError vixError = VixDiskLib_Cleanup(&cnxParams, &numCleanedUp, &numRemaining);
```

Ordering of Function Calls in Sequence

The code block below shows the suggested ordering of VixDiskLib function calls in sequence, including older function calls in combination with newer ones (Ex and Access).

For backup, the open, read, and close calls are made on virtual machine snapshots.

```
VixDiskLib_InitEx()

VixDiskLib_PrepareForAccess()

VixDiskLib_ConnectEx()

VixDiskLib_Open()

VixDiskLib_Read()

VixDiskLib_Close()

VixDiskLib_Disconnect()

VixDiskLib_EndAccess()

VixDiskLib_CleanUp()

VixDiskLib_Exit()
```

Updating Applications for Advanced Transport

To update your applications for advanced transport with managed disk, follow these steps:

1. Find all instances of **`VixDiskLib_Connect()`** and change them to **`VixDiskLib_ConnectEx()`**.

   The vixDiskLib sample program was extended to use **`VixDiskLib_ConnectEx()`** with the **`-mod`** option.

2. Likewise, change **`VixDiskLib_Init()`** to **`VixDiskLib_InitEx()`** and be sure to call it only once.
3. Deactivate virtual machine relocation with the **`VixDiskLib_PrepareForAccess()`** call.
4. Add parameters in the middle:
   - **`TRUE`** for high performance read-only access, **`FALSE`** for read/write access.
   - Snapshot **`moRef`**, if applicable.
   - **`NULL`** to accept transport method defaults (recommended).
5. Re-enable virtual machine relocation with the **`VixDiskLib_EndAccess()`** call.
6. Find **`VixDiskLib_Disconnect()`** near the end of program, and for safety add a **`VixDiskLib_Cleanup()`** call immediately afterwards.
7. Compile with the new flexible-transport-enabled version of VixDiskLib.

   The advanced transport functions are useful for backing up or restoring data on virtual disks managed by VMware vSphere. Backup is based on the snapshot mechanism, which provides a data view at a certain point in time, and allows access to quiescent data on the parent disk while the child disk continues changing.

Algorithm for vSphere Backup

A typical backup application follows this algorithm:

- Preferably through vCenter Server, contact the ESXi host and discover the target virtual machine.
- Ask the ESXi host to take a snapshot of the target virtual machine.

- Using the vSphere API (**`PropertyCollector`**), capture configuration (**`VirtualMachineConfigInfo`**) and changed block information (with **`queryChangedDiskAreas`**). Save these for later.
- Using advanced transport functions and VixDiskLib, access the snapshot and save the data in it. If Changed Block Tracking is enabled, the snapshot contains only incremental backup data.
- Ask the ESXi host to delete the backup snapshot.

A typical back-in-time disaster recovery or file-based restore follows this algorithm:

- Preferably through VMware vCenter, contact the ESXi host containing the target virtual machine.
- Ask the ESXi host to halt and power off the target virtual machine.
- Using advanced transport functions, restore a snapshot from saved backup data.
- For disaster recovery to a previous point in time, have the virtual machine revert to the restored snapshot. For file-based restore, mount the snapshot and restore requested files.

Backing Up Virtual Disks in vSphere presents these algorithms in more detail and includes code samples. For best practices in backup and restore, see Tips and Best Practices.

Backup and Recovery Example

The VMware vSphere API method **`queryChangedDiskArea`** returns a list of disk sectors that changed between an existing snapshot, and some previous time identified by a change ID.

The **`queryChangedDiskAreas`** method takes four arguments, including a snapshot reference and a change ID. It returns a list of disk sectors that changed between the time indicated by the change ID and the time of the snapshot. This is useful for incremental backup. Before a full backup, you can call **`VixDiskLib_QueryAllocatedBlocks`** to get a list of in-use disk sectors so your backup can skip unallocated sectors.
Suppose that you create an initial backup at time *`T1`*. Later at time *`T2`* you take an incremental backup, and another incremental backup at time *`T3`*. (You could use differential backups instead of incremental backups, which would trade off greater backup time and bandwidth for shorter restore time.)
For the full backup at time T1:

1. Keep a record of the virtual machine configuration, **`VirtualMachineConfigInfo`**.
2. Create a snapshot of the virtual machine, naming it **snapshot_T1**.
3. Obtain the change ID for each virtual disk in the snapshot, **changeId_T1** (per VMDK).
4. Back up the sectors returned by **`VixDiskLib_QueryAllocatedBlocks`**, avoiding unallocated disk.
5. Delete **snapshot_T1**, keeping a record of **changeId_T1** along with lots of backed-up data.

For the incremental backup at time T2:

1. Create a snapshot of the virtual machine, naming it **snapshot_T2**.
2. Obtain the change ID for each virtual disk in the snapshot, **changeId_T2** (per VMDK).
3. Back up the sectors returned by **`queryChangedDiskAreas(snapshot_T2`**,... **`changeId_T1)`**.
4. Delete **snapshot_T2**, keeping a record of **changeId_T2** along with backed-up data.

For the incremental backup at time T3:

1. Create a snapshot of the virtual machine, naming it **snapshot_T3**.
   At time T3 you can no longer obtain a list of changes between T1 and T2.
2. Obtain the change ID for each virtual disk in the snapshot, **changeId_T3** (per VMDK).
3. Back up the sectors returned by **`queryChangedDiskAreas(snapshot_T3`**,... **`changeId_T2)`**.
   A differential backup could be done with **`queryChangedDiskAreas(snapshot_T3`**,... **`changeId_T1)`**.
4. Delete **snapshot_T3**, keeping a record of **changeId_T3** along with backed-up data.

For a disaster recovery at time T4:

1. Create a new virtual machine with no guest operating system installed, using configuration parameters you previously saved from **`VirtualMachineConfigInfo`**. You do not need to format the virtual disks, because restored data includes formatting information.

2. Restore data from the backup at time T3. Keep track of which disk sectors you restore.
3. Restore data from the incremental backup at time T2, skipping any sectors already recovered.
   With differential backup, you can skip copying the T2 backup.
4. Restore data from the full backup at time T1, skipping any sectors already recovered. The reason for working backwards is to get the newest data while avoiding unnecessary data copying.
5. Power on the recovered virtual machine.

When programs open remote disk with SAN transport mode, they can write to the base disk, but they cannot write to a snapshot (redo log). Opening and writing snapshots is supported only for hosted disk.

Multithreading Considerations

In multithreaded programs, disk requests should be serialized by the client program. Disk handles are not bound to a thread and may be used across threads. You can open a disk in one thread and use its handle in another thread, provided you serialize disk access. Alternatively you can use a designated open-close thread, as shown in the workaround below.

Multiple Threads and VixDiskLib

VDDK supports concurrent I/O to multiple virtual disks, with certain limitations:

- **`VixDiskLib_InitEx()`** should be called once per process, from the main thread.
- On Windows, backup applications should call **`VixDiskLib_InitEx`** at the very beginning of the program. As of VDDK 7.0, which introduced some thread local storage, all threads that call VixDiskLib functions must be created and initialized after calling **`VixDiskLib_InitEx`**. This is advised but not required for Linux.
- In the **`VixDiskLib_InitEx()`** function call, do not specify logging callbacks as NULL. This causes VixDiskLib to provide its default logging functions, which are not thread safe. If you are using VDDK in a multithreaded environment, you must provide your own thread-safe log functions.
- When you call **`VixDiskLib_Open()`** and **`VixDiskLib_Close()`**, VDDK initializes and uninitializes a number of libraries, some of which do not work if called from multiple threads. For example, this fails:

  ```
  Thread 1: VixDiskLib_Open ...... VixDiskLib_Close

  Thread 2: ................................... VixDiskLib_Open ...... VixDiskLib_Close
  ```

  The workaround is to use one designated thread to do all opens and closes, and to have other worker threads doing reads and writes. This diagram shows concurrent reads on two separate disk handles. Concurrent reads on the same disk handles are not allowed.

  ```
  Open/Close Thread:

  VixDiskLib_Open ...... VixDiskLib_Open ...... VixDiskLib_Close ......
  VixDiskLib_Close ......

  (handle1)              (handle2)              (handle1)               (handle2)

  I/O Thread 1:

  (owns handle1)   VixDiskLib_Read ... VixDiskLib_Read ...

  I/O Thread 2:

  (owns handle2)                          VixDiskLib_Read ... VixDiskLib_Read ...
  ```

Capabilities of Library Calls

This section describes limitations, if any.

Support for Managed Disk

Some operations are not supported:

- For **`VixDiskLib_Connect()`** to open a managed disk connection, you must provide valid vSphere access credentials. On ESXi hosts, **`VixDiskLib_Open()`** cannot open a single link in a disk chain.
- For **`VixDiskLib_Create()`** to create a managed disk on an ESXi host, first create a hosted type disk, then use **`VixDiskLib_Clone()`** to convert the hosted virtual disk to managed virtual disk.
- **`VixDiskLib_Defragment()`** can defragment hosted disks only.
- **`VixDiskLib_Grow()`** can grow hosted disks only.
- **`VixDiskLib_Unlink()`** can delete hosted disks only.
- Until ESXi 5.1, the HotAdd transport was available only with vSphere Enterprise Edition and higher.

Support for Hosted Disk

Most everything (except advanced transport) is supported, except:

- The **`VixDiskLib_ConnectEx()`** extended connect function.
- SAN and HotAdd advanced transports.
- **`VixDiskLib_PrepareForAccess()`** and **`VixDiskLib_EndAccess()`** to delay Storage VMotion.

Virtual Disk API Sample Code

This chapter discusses the VDDK sample program, in the following sections:

Compiling the Sample Program

The sample program is written in C++, although the Virtual Disk API also supports C. For compilation to succeed, the correct DLLs or shared objects must be loaded. You can ensure the success of dynamic loading in a variety of ways.

- Set the path inside the VDDK program.
- Set the path for the shell being used in Linux or in Visual Studio for Windows.
  For a default installation, the Linux path is `/usr/share/doc/vmware-vix-disklib/sample`.
- In Windows, set the Path element in the System Variables.
  To do this in Windows XP, right-click **Computer › Properties › Advanced › Environment Variables**, select **`Path`** in the System Variables lower list, click Edit, and add the path of the VDDK bin directory.

  In Windows 7, right-click **Computer › Properties › Advanced System Settings › Environment Variables**, select **`Path`** in the **System Variables** list, click Edit, and add the path of the VDDK bin directory.

  `C:\Program Files\VMware\VMware`**`Virtual Disk`**`Development Kit\doc\sample\` is the default path.

Note that VDDK loads DLLs by relative path rather than absolute path, so conflicting versions of the DLLs could cause problems.

Visual C++ on Windows

To compile the program, find the sample source `vixDiskLibSample.cpp` at this location:

`C:\Program Files\VMware\VMware Virtual Disk Development Kit\doc\sample\`
For VDDK 5.5 and later, make sure that you have the 64-bit debugging tools installed along with Visual Studio. Double-click the **`vcproj`** file, possibly convert format to a newer version, and choose Build > Build Solution.
To execute the compiled program, choose Debug > Start Without Debugging, or type this in a command prompt after changing to the `doc\sample` location given above:
`Debug\vixdisklibsample.exe`

SLN and VCPROJ Files

The Visual Studio solution file `vixDiskLibSample.sln` and project file **`vixDiskLibSample.vcproj`** are included in the sample directory.

C++ on Linux Systems

Find the sample source in this directory:

```
/usr/share/doc/vmware-vix-disklib/samples/diskLib
```

You can copy `vixDiskLibSample.cpp` and its **`Makefile`** to a directory where you have write permission, or switch user to root. On some Linux systems you need to add **`#include`** statements for **`<stdio.h>`** and **`<string.h>`** after the **`#else`** clause on line 15. Type the **`make`** command to compile. Run the application:

```
make
./vix-disklib-sample
```

> **NOTE**
> If this fails, edit `/etc/ld.so.conf` and run **`ldconfig`** as root or change your **`LD_LIBRARY_PATH`** environment to include the library installation path, `/usr/lib/vmware-vix-disklib/lib64`.

Makefile

The **`Makefile`** fetches any packages that are required for compilation but are not installed.

Library Files Required

The virtual disk library comes with dynamic libraries, or shared objects on Linux, to simplify the delivery of third-party and open source components.

Windows requires the `lib/vixDiskLib.lib` file for linking, and the `bin/*.dll` files at runtime.
Linux uses **`.so`** files for both linking and running.

Usage Message

Running the sample application without arguments produces the following usage message:

```
Usage: vixdisklibsample command [options] diskPath

commands:

 -create : creates a sparse virtual disk with capacity specified by -cap

 -redo parentPath : creates a redo log 'diskPath' for base disk 'parentPath'

 -info : displays information for specified virtual disk

 -dump : dumps the contents of specified range of sectors in hexadecimal

 -fill : fills specified range of sectors with byte value specified by -val

 -wmeta key value : writes (key,value) entry into disk's metadata table

 -rmeta key : displays the value of the specified metada entry

 -meta : dumps all entries of the disk's metadata

 -clone sourcePath : clone source vmdk possibly to a remote site

 -readbench blocksize: do read benchmark on a disk using the specified I/O block size in
sectors
```

```
 -writebench blocksize: do write benchmark on disk using the specified I/O block size in
sectors

options:

 -adapter [ide|scsi] : bus adapter type for 'create' option (default='scsi')

 -start n : start sector for 'dump/fill' options (default=0)

 -count n : number of sectors for 'dump/fill' options (default=1)

 -val byte : byte value to fill with for 'write' option (default=255)

 -cap megabytes : capacity in MB for -create option (default=100)

 -single : open file as single disk link (default=open entire chain)

 -multithread n: start n threads and copy the file to n new files

 -host hostname : hostname / IP addresss (ESXi or vCenter)

 -user userid : user name on host (default = root)

 -password password : password on host

 -port port : port to use to connect to host (default = 443)

 -vm vmPath=/path/to/vm : inventory path to vm that owns the virtual disk

 -libdir dir : Directory containing vixDiskLibPlugin library

 -initex configfile : Use VixDiskLib_InitEx

 -ssmoref moref : Managed object reference of VM snapshot

 -mode mode : Mode string to pass into VixDiskLib_ConnectEx

 -thumb string : Provides a SSL thumbprint string for validation
```

The **`-thumb`** option is a new security-related feature in the VDDK 5.1 release. See SSL Certificate Thumbprint.

The sample program's **`-single`** option, which opens a single link instead of the entire disk chain, is supported for (local) hosted disk, but not for (remote) managed disk.

To connect to an ESXi host with the sample program, you must specify the options **`-host`**, **`-user`**, **`-password`**, and provide a **`diskPath`** on the ESXi host's datastore. For example:

```
vix-diskLib-sample -info -host esx5 -user root -password secret "[datastore1] <VM>/
<VM>.vmdk"
```

To connect to vCenter Server, you must also specify the options **`-libdir`** and **`-vm`**. Programs need **`libdir`** so the DiskLibPlugin can connect with vCenter Server, which must locate the VM. For example:

```
vix-diskLib-sample -info -host vc5 -user Administrator -password secret

                -libdir <pluginDir> -vm vmPath=<path/to/VM> "[<partition>] <VM>/<VM>.vmdk"
```

The **`vmPath`** is formulated in vSphere Client by starting at vCenter and inserting `/vm/` before the VM name. The **`diskPath`** is ascertained by clicking **Edit Settings › Hard Disk** and copying the **Disk File** name.

```
vix-disklib-sample -info -host vc5 -user Administrator -password secret
```

```
                -libdir /usr/lib/vmware-vix-disklib/lib64 -vm vmPath=Datacenter/vm/RHEL5

                "[datastore1] RHEL5/RHEL5.vmdk"
```

To connect using an advanced transport, for example to virtual machine disk on SAN storage, you must also specify the options **-mode** and **-ssmoref**. The transport mode and managed object reference (of a snapshot) are required for **VixDiskLib_ConnectEx()**. To find the **ssmoref**, log in to the managed object browser for the vCenter Server, and click **content › rootFolder › Datacenter › datastore › vm › snapshot**. A snapshot must exist, because it is a bad idea to open the base disk of a powered-on VM.

```
vix-disklib-sample -info -host vc5 -user Administrator -password secret -mode san

                -libdir /usr/lib/vmware-vix-disklib/lib64 -vm vmPath=Datacenter/vm/RHEL5

                -ssmoref snapshot-72 "[datastore1] RHEL5/RHEL5.vmdk"
```

On Windows, the VDDK package installs `diskLibPlugin.dll` in the `\bin` folder, not the `\lib` folder, so change **<pluginDir>** accordingly.

Walk-Through of Sample Program

The sample program is the same for Windows as for Linux, with **#ifdef** blocks for Win32.

Include Files

Windows dynamic link library (DLL) declarations are in **process.h**, while Linux shared object (**.so**) declarations are in **dlfcn.h**. Windows offers the **tchar.h** extension for Unicode generic text mappings, not readily available in Linux.

Definitions and Structures

The sample program uses twelve bitwise shift operations (1 << 11) to track its available commands and the multithread option. The Virtual Disk API has about 30 library functions, some for initialization and cleanup. The following library functions are not demonstrated in the sample program:

- **VixDiskLib_Rename()**
- **VixDiskLib_Defragment()**
- **VixDiskLib_Grow()**
- **VixDiskLib_Shrink()**
- **VixDiskLib_Unlink()**
- **VixDiskLib_Attach()**

The sample program transmits state in the **appGlobals** structure.

Dynamic Loading

The **#ifdef DYNAMIC_LOADING** block is long, starting on line 97 and ending at line 339. This block contains function definitions for dynamic loading. It also contains the **LoadOneFunc()** procedure to obtain any requested function from the dynamic library and the **DynLoadDiskLib()** procedure to bind it. This demonstration feature could also be called “runtime loading” to distinguish it from dynamic linking.

To try the program with runtime loading enabled on Linux, add **-DDYNAMIC_LOADING** after **g++** in the **Makefile** and recompile. On Windows, define **DYNAMIC_LOADING** in the project.

Wrapper Classes

Below the dynamic loading block are two wrapper classes, one for error codes and descriptive text, and the other for the connection handle to disk.

The error wrapper appears in **`catch`** and **`throw`** statements to simplify error handling across functions.

Wrapper class **`VixDisk`** is a clean way to open and close connections to disk. The only time that library functions **`VixDiskLib_Open()`** and **`VixDiskLib_Close()`** appear elsewhere, aside from dynamic loading, is in the **`CopyThread()`** function near the end of the sample program.

Command Functions

The print-usage message appears next, with output partially shown in Usage Message.

Next comes the **`main()`** function, which sets defaults and parses command-line arguments to determine the operation and possibly set options to change defaults. Dynamic loading occurs, if defined. Notice the all-zero initialization of the **`VixDiskLibConnectParams`** declared structure:

```
VixDiskLibConnectParams cnxParams = {0};
```

For connections to an ESXi host, credentials including user name and password must be correctly supplied in the **`-user`** and **`-password`** command-line arguments. Both the **`-host`** name of the ESXi host and its **`-vm`** inventory path (**`vmxSpec`**) must be supplied. When set, these values populate the **`cnxParams`** structure. Initialize all parameters, especially **`vmxSpec`**, or else the connection might behave unexpectedly.

A call to **`VixDiskLib_Init()`** initializes the library. In a production application, you can supply appropriate **`log`**, **`warn`**, and **`panic`** functions as parameters, in place of NULL.

A call to **`VixDiskLib_Connect()`** creates a connection to disk. If host **`cnxParams.serverName`** is null, as it is without the **`-host`** argument, a connection is made to hosted disk on the local host. Otherwise a connection is made to managed disk on the remote host. With **`-ssmoref`** argument, advanced transport is used.

Next, an appropriate function is called for the requested operation, followed by error information if applicable. Finally, the **`main()`** function closes the library connection to disk and exits.

DoInfo()

This procedure calls **`VixDiskLib_GetInfo()`** for information about the virtual disk, displays results, and calls **`VixDiskLib_FreeInfo()`** to reclaim memory. The parameter **`disk.Handle()`** comes from the **`VixDisk`** wrapper class discussed in Wrapper Classes.

In this example, the sample program connects to an ESXi host named **`esx5`** and displays virtual disk information for a Red Hat Enterprise Linux client. For an ESXi host, path to disk is often something like **`[datastore1]`** followed by the virtual machine name and the VMDK filename.

vix-diskLib-sample -info -host esx5 -user root -password secret "[datastore1] RHEL6/RHEL6.vmdk"

```
vix-diskLib-sample -info -host esx5 -user root -password secret "[datastore1] RHEL6/
RHEL6.vmdk"

Disk "[datastore1] RHEL6/RHEL6.vmdk" is open using transport mode "nbdssl".

capacity          = 4194304 sectors
```

```
number of links   = 1

adapter type      = LsiLogic SCSI

BIOS geometry     = 0/0/0

physical geometry = 261/255/63

Transport modes supported by vixDiskLib: file:nbdssl
```

If you multiply physical geometry numbers (261 cylinders * 255 heads per cylinder * 63 sectors per head) the result is a capacity of 4192965 sectors, although the first line says 4194304. A small discrepancy is possible due to rounding. In general, you get at least the capacity that you requested. The number of links specifies the separation of a child from its original parent in the disk chain (redo logs), starting at one. The parent has one link, its child has two links, the grandchild has three links, and so forth.

DoCreate()

This procedure calls **`VixDiskLib_Create()`** to allocate virtual disk. Adapter type is SCSI unless specified as IDE on the command line. Size is 100MB, unless set by **`-cap`** on the command line. Because the sector size is 512 bytes, the code multiplies **`appGlobals.mbsize`** by 2048 instead of 1024. Type is always monolithic sparse and Workstation 5. In a production application, **`progressFunc`** and callback data can be defined rather than NULL. Type these commands to create a sample VMDK file (the first line is for Linux only):

export LD_LIBRARY_PATH=/usr/lib/vmware-vix-disklib/lib64

vix-disklib-sample -create sample.vmdk

As a VMDK file, monolithic sparse (growable in a single file) virtual disk is initially 65536 bytes (2 ^ 16) in size, including overhead. The first time you write to this type of virtual disk, as with **`DoFill()`** below, the VMDK expands to 131075 bytes (2 ^ 17), where it remains until more space is needed. You can verify file contents with the **`-dump`** option.

DoRedo()

This procedure calls **`VixDiskLib_CreateChild()`** to establish a redo log. A child disk records disk sectors that changed since the parent disk or previous child. Children can be chained as a set of redo logs.

The sample program does not demonstrate use of **`VixDiskLib_Attach()`**, which you can use to access a link in the disk chain. **`VixDiskLib_CreateChild()`** establishes a redo log, with the child replacing the parent for read/write access. Given a pre-existing disk chain, **`VixDiskLib_Attach()`** creates a related child, or a cousin you might say, that is linked into some generation of the disk chain.

For a diagram of the attach operation, see Attach Child to Parent Disk.

Write by DoFill()

This procedure calls **`VixDiskLib_Write()`** to fill a disk sector with ones (byte value FF) unless otherwise specified by **`-val`** on the command line. The default is to fill only the first sector, but this can be changed with options **`-start`** and **`-count`** on the command line.

DoReadMetadata()

This procedure calls **`VixDiskLib_ReadMetadata()`** to serve the **`-rmeta`** command-line option. For example, type this command to obtain the universally unique identifier:

vix-disklib-sample -rmeta uuid sample.vmdk

DoWriteMetadata()

This procedure calls **`VixDiskLib_WriteMetadata()`** to serve the **`-wmeta`** command-line option. For example, you can change the tools version from 1 to 2 as follows:

vix-disklib-sample -wmeta toolsVersion 2 sample.vmdk

DoDumpMetadata()

This procedure calls **`VixDiskLib_GetMetadataKeys()`** then **`VixDiskLib_ReadMetadata()`** to serve the **`-meta`** command-line option. Two read-metadata calls are needed for each key: one to determine length of the value string and another to fill in the value. See Get Metadata Table from Disk.

In the following example, the sample program connects to an ESXi host named **`esx3`** and displays the metadata of the Red Hat Enterprise Linux client's virtual disk. For an ESXi host, path to disk might be **`[storage1]`** followed by the virtual machine name and the VMDK filename.

```
vix-diskLib-sample -meta -host esx3 -user admin -password secret "[storage1]RHEL5/
RHEL5.vmdk"

geometry.sectors = 63

geometry.heads = 255

geometry.cylinders = 522

adapterType = buslogic

toolsVersion = 1

virtualHWVersion = 7
```

Tools version and virtual hardware version appear in the metadata, but not in the disk information retrieved by DoInfo(). Geometry information and adapter type are repeated, but in a different format. Other metadata items not listed above might exist.

DoDump()

This procedure calls **`VixDiskLib_Read()`** to retrieve sectors and displays sector contents on the output in hexadecimal. The default is to dump only the first sector numbered zero, but you can change this with the **`-start`** and **`-count`** options. Here is a sequence of commands to demonstrate:

```
vix-disklib-sample -create sample.vmdk

vix-disklib-sample -fill -val 1 sample.vmdk

vix-disklib-sample -fill -val 2 -start 1 -count 1 sample.vmdk

vix-disklib-sample -dump -start 0 -count 2 sample.vmdk

od -c sample.vmdk
```

On Linux (or Cygwin) you can run the **`od`** command to show overhead and metadata at the beginning of file, and the repeated ones and twos in the first two sectors. The **`-dump`** option of the sample program shows only data, not overhead.

DoTestMultiThread()

This procedure employs the Windows thread library to make multiple copies of a virtual disk file. Specify the number of copies with the **-multithread** command-line option. For each copy, the sample program calls the **CopyThread()** procedure, which in turn calls a sequence of six Virtual Disk API routines.

On Linux the multithread option is unimplemented.

DoClone()

This procedure calls **VixDiskLib_Clone()** to make a copy of the data on virtual disk. A callback function, supplied as the sixth parameter, displays the percent of cloning completed. For local hosted disk, the adapter type is SCSI unless specified as IDE on the command line, size is 200MB, unless set by **-cap** option, and type is monolithic sparse, for Workstation. For an ESXi host, adapter type is taken from managed disk itself, using the connection parameters established by **VixDiskLib_Connect()**.

If **createParams.diskType** is **VIXDISKLIB_DISK_VMFS_THIN**, clone converts the destination VMDK to thin. This is an exception to ignoring **createParams** for the remote case.

The final parameter **TRUE** means to overwrite if the destination VMDK exists.

The clone option is a good backup method. Sometimes a cloned virtual disk is smaller, because it can be organized more efficiently. Moreover, a fully allocated flat file can be converted to sparse or thin representation.

SSL Certificate Thumbprint

The sample program in the VDDK 5.1 release added the **-thumb** option to allow an SSL certificate thumbprint to be provided and used. The thumbprint is used for authentication through vCenter Server.

Practical Programming Tasks

This chapter presents some practical programming challenges not covered in the sample program.

Scan VMDK for Virus Signatures

One of the Use Cases for the Virtual Disk Library is to scan a VMDK for virus signatures. Using our sample program framework, this example function implements the **-virus** command-line option, using hypothetical library routine **SecureVirusScan()**, supplied by an antivirus software vendor. The library routine scans a buffer against the vendor's latest pattern library, returning TRUE if it identifies a virus.

```
extern int SecureVirusScan(const uint8 *buf, size_t n);

/*

*   DoVirusScan - Scan the content of a virtual disk for virus signatures.

*/

static void DoVirusScan(void)

{

    VixDisk disk(appGlobals.connection, appGlobals.diskPath, appGlobals.openFlags);

    VixDiskLibDiskInfo info;

    uint8 buf[VIXDISKLIB_SECTOR_SIZE];
```

```
    VixDiskLibSectorType sector;


    VixError vixError = VixDiskLib_GetInfo(disk.Handle(), &info);
    CHECK_AND_THROW(vixError);
    cout << "capacity = " << info.capacity << " sectors" << endl;
    // read all sectors even if not yet populated
    for (sector = 0; sector < info.capacity; sector++) {
       vixError = VixDiskLib_Read(disk.Handle(), sector, 1, buf);
       CHECK_AND_THROW(vixError);
       if (SecureVirusScan(buf, sizeof buf)) {
          printf("Virus detected in sector %d\n", sector);
        }
    }
    cout << info.capacity << " sectors scanned" << endl;
}
```

This function calls **`VixDiskLib_GetInfo()`** to determine the number of sectors allocated in the virtual disk. The number of sectors is available in the **`VixDiskLibDiskInfo`** structure, but normally not in the metadata. With **`SPARSE`** type layout, data can occur in any sector, so this function reads all sectors, whether filled or not. **`VixDiskLib_Read()`** continues without error when it encounters an empty sector full of zeroes.
The following difference list shows the remaining code changes necessary for adding the **`-virus`** option to the `vixDiskLibSample.cpp` sample program:

```
43a44
> #define COMMAND_VIRUS_SCAN      (1 << 10)
72a74
> static void DoVirusScan(void);
425a429
>     printf(" -virus: scan source vmdk for virus signature \n");
519a524,525
>       } else if (appGlobals.command & COMMAND_VIRUS_SCAN) {
>          DoVirusScan();
564a571,572
>       } else if (!strcmp(argv[i], "-virus")) {
>             appGlobals.command |= COMMAND_VIRUS_SCAN;
```

Creating Virtual Disks

This section discusses the types of local VMDK files and how to create virtual disk for a remote ESXi host.

Create Local Disk

The sample program presented in Virtual Disk API Sample Code creates virtual disk of type **MONOLITHIC_SPARSE**, in other words one big file, not pre-allocated. This is the default because modern file systems, in particular NTFS, support files larger than 2GB, and can hold more than 2GB of total data. This is not true of legacy file systems, such as FAT16 on MS-DOS and early Windows, or the ISO9660 file system for writing files on CD, or NFS version 2, or Linux kernel 2.4. All are limited to 2GB per volume. FAT and FAT32 were extended to 4GB in NT 3.51.
However, a **SPLIT** virtual disk might be safer than the **MONOLITHIC** variety, because if something goes wrong with the underlying host file system, some data might be recoverable from uncorrupted 2GB extents. VMware products do their best to repair a damaged VMDK, but having a split VMDK increases the chance of salvaging files during repair. On the downside, **SPLIT** virtual disk involves higher overhead (more file descriptors) and increases administrative complexity.
When required for a FAT16 or early Linux file system, you can create **SPLIT_SPARSE** virtual disk. The change is simple: the line highlighted in boldface. The sample program could be extended to have an option for this.

```
static void DoCreate(void)

{

   VixDiskLibAdapterType adapter = strcmp(appGlobals.adapterType, "scsi") == 0 ?
                       VIXDISKLIB_ADAPTER_SCSI_BUSLOGIC : VIXDISKLIB_ADAPTER_IDE;

   VixDiskLibCreateParams createParams;

   VixError vixError;

   createParams.adapterType = adapter;

   createParams.capacity = appGlobals.mbSize * 2048;

   createParams.diskType = VIXDISKLIB_DISK_SPLIT_SPARSE;

   vixError = VixDiskLib_Create(appGlobals.connection, appGlobals.diskPath, &createParams,
NULL, NULL);

   CHECK_AND_THROW(vixError);

}
```

> **NOTE**
> You can split VMDK files into smaller than 2GB extents, but created filenames still follow the patterns shown in Table : VMDK Virtual Disk Files.

This one-line change to **DoCreate()** causes creation of 200MB split VMDK files (200MB being the capacity set on the previous line) unless the **-cap** command-line argument specifies otherwise.

Create Remote Disk

As stated in Support for Managed Disk, **VixDiskLib_Create()** does not support managed disk. To create a managed disk on the remote ESXi host, first create a hosted disk on the local Workstation, then convert the hosted disk into managed disk with **VixDiskLib_Clone()** over the network.
To create remote managed disk using the sample program, type the following commands:

```
./vix-disklib-sample -create -cap 1000000 virtdisk.vmdk
./vix-disklib-sample -clone virtdisk.vmdk -host esx3i -user root -password secret
vmfsdisk.vmdk
```

You could write a virtual-machine provisioning application to perform the following steps:

1. Create a hosted disk VMDK with 2GB capacity, using **`VixDiskLib_Create()`**.
2. Write image of the guest OS and application software into the VMDK, using **`VixDiskLib_Write()`**.
3. Clone the hosted disk VMDK onto the VMFS file system of the ESXi host.

   ```
   vixError = VixDiskLib_Clone(appGlobals.connection, appGlobals.diskPath,

                  srcConnection, appGlobals.srcPath,                &createParams,
   CloneProgressFunc, NULL, TRUE);
   ```

   In this call, **`appGlobals.connection`** and **`appGolbals.diskPath`** represent the remote VMDK on the ESXi host, while **`srcConnection`** and `appGlobals.src`Path represent the local hosted VMDK.

4. Power on the new guest OS to get a new virtual machine.

   On Workstation, the **`VixVMPowerOn()`** function in the VIX API does this. For ESXi hosts, you must use the **`PowerOnVM_Task`** method. As easy way to use this method is in the VMware vSphere Perl Toolkit, which has the **`PowerOnVM_Task()`** call (non-blocking), and the **`PowerOnVM()`** call (synchronous).

5. Provision and deploy the new virtual machine on the ESXi host.

Special Consideration for ESXi Hosts

No matter what virtual file type you create in Step 1, it becomes type **`VIXDISKLIB_DISK_VMFS_FLAT`** in Step 3.

VMDK File Versions

Virtual disk programs must be able to cope with VMDK files up to version three (3).

Version 1 was the initial version of VMDK. All released builds of **`vixDiskLib`** can read and write this version.

Version 2 added disk encryption for hosted products (Workstation and Fusion), although encrypted disks were never implemented on ESXi. Version 2 VMDK files can be transferred to and appear on ESXi, where they are treated like version 1 VMDK files.

Version 3 added support for persistent changed block tracking (CBT), and is set when CBT is enabled for a virtual disk. CBT is supported on VMFS datastores. This version 3 first appeared in ESXi 4.0 and continues unchanged in recent vSphere releases. When CBT is enabled, the version number is incremented, and decremented when CBT is deactivated.

If you look at the `.vmdk` descriptor file for a version 3 virtual disk, you can see a pointer to its `*-ctk.vmdk` ancillary file. For example:

```
version=3

...

# Change Tracking File

changeTrackPath="Windows-2008R2x64-2-ctk.vmdk"
```

The **`changeTrackPath`** setting references a file that describes changed areas on the virtual disk.

If you want to back up the changed area information, then your software should copy the `*-ctk.vmdk` file and preserve the “Change Tracking File” line in the `.vmdk` descriptor file. If you do not want to back up the changed area information, then you can discard the ancillary file, remove the “Change Tracking File” line, read the VMDK file data as if it were version 1, and roll back the version number on restore.

Working with Virtual Disk Data

The virtual disk library reads and writes sectors of data. It has no interface for character or byte-oriented I/O.

Reading and Writing Local Disk

Demonstrating random I/O, this function reads a sector at a time backwards through a VMDK. If it sees the string “VmWare” it substitutes the string “VMware” in its place and writes the sector back to VMDK.

```
#include <string>

static void DoEdit(void)/

{

    VixDisk disk(appGlobals.connection, appGlobals.diskPath, appGlobals.openFlags);

    uint8 buf[VIXDISKLIB_SECTOR_SIZE];

    VixDiskLibSectorType i;

    string str;

    for (i = appGlobals.numSectors; i >= 0; i--) {

       VixError vixError;

       vixError = VixDiskLib_Read(disk.Handle(), appGlobals.startSector + i, 1, buf);

       CHECK_AND_THROW(vixError);

       str = buf;

       if (pos = str.find("VmWare", 0)) {

          str.replace(pos, 5, "VMware");

          buf = str;

          vixError = VixDiskLib_Write(disk.Handle(), appGlobals.startSector + i, 1, buf);

          CHECK_AND_THROW(vixError);

       }

    }

}
```

Reading and Writing Remote Disk

The **`DoEdit()`** function is similar for remote managed virtual disk on ESXi hosts, but beforehand you must call **`VixDiskLib_Connect()`** with authentication credentials instead of passing NULL parameters.

```
    if (appGlobals.isRemote) {

          cnxParams.vmxSpec = NULL;

          cnxParams.serverName = appGlobals.host;

          cnxParams.credType = VIXDISKLIB_CRED_UID;
```

```
        cnxParams.creds.uid.userName = appGlobals.userName;
        cnxParams.creds.uid.password = appGlobals.password;
        cnxParams.port = appGlobals.port;
    }
    VixError vixError = VixDiskLib_Init(1, 0, NULL, NULL, NULL, NULL);
    CHECK_AND_THROW(vixError);
    vixError = VixDiskLib_Connect(&cnxParams, &appGlobals.connection);
```

Deleting a Disk (Unlink)

The function to delete virtual disk files is **`VixDiskLib_Unlink()`**. It takes two arguments: a connection and a VMDK filename.

```
vixError = VixDiskLib_Unlink(appGlobals.connection, appGlobals.diskPath);
```

Effects of Deleting a Virtual Disk

When you delete a VMDK, you lose all the information it contained. In most cases, the host operating system prevents you from doing this when a virtual machine is running. However, if you delete a VMDK with its virtual machine powered off, that guest OS becomes unbootable.

Renaming a Disk

The function to rename virtual disk files is **`VixDiskLib_Rename()`**. It takes two arguments: the old and the new VMDK filenames.

```
vixError = VixDiskLib_Rename(oldGlobals.diskpath, newGlobals.diskpath);
```

Effects of Renaming a Virtual Disk

The server expects VMDK files of its guest OS virtual machines to be in a predictable location. Any file accesses that occur during renaming might cause I/O failure and possibly cause a guest OS to fail. Renaming virtual disk does not reset changed block tracking (CBT).

Managing Child Disks

In the Virtual Disk API, redo logs are managed as a parent-child disk chain, each child being the redo log of disk changes made since its inception. Trying to write on the parent after creating a child results in an error. The library expects you to write on the child instead. See Attach Child to Parent Disk for a diagram.

Create Redo Logs

A redo log is created by taking a virtual machine snapshot, which contains both disk data and virtual machine state. On hosted disk only, **`VixDiskLib_CreateChild()`** creates a redo log without virtual machine state.

You could write a simple application to create redo logs, or snapshots on managed disk, at 3:00 AM nightly. (although multiple snapshots have a performance impact). When you create a redo log while the virtual machine is running, the

VMware host re-arranges file pointers so the primary VMDK, <vmname>.vmdk for example, keeps track of redo logs in the disk chain. Use the disk chain to re-create data for any given day.
To re-create data for any given day

1. Locate the <vmname>-<NNN>.vmdk redo log for the day in question.

   <NNN> is a sequence number. You can identify this redo log or snapshot by its timestamp.

2. Initialize the virtual disk library and open the redo log to obtain its parent handle.
3. Create a child disk with the **VixDiskLib_Create()** function, and attach it to the parent:

   ```
   vixError = VixDiskLib_Attach(parent.Handle(), child.Handle());
   ```

4. Read and write the virtual disk of the attached child.

   This is just an example. On managed disk, multiple snapshots are not recommended for performance reasons. Backup software on vSphere usually takes a snapshot, saves data to backup media, then deletes the snapshot.

Virtual Disk in Snapshots

The Virtual Disk API provides the following features to deal with the disk component of snapshots:

- Attaching an arbitrary child in a disk chain
- Opening read-only virtual disks
- Ability to open snapshot disk on ESXi hosts through VMware vCenter

RDM Disks and Virtual BIOS

This section outlines low-level procedures for restoring raw device mapping (RDM) disks and NVRAM.

Restore RDM Disks

Backing up and restoring RDM disks presents unusual challenges. The original backed-up RDM configuration might not apply, and is probably not appropriate, if users restore:

- A virtual machine to a different host or datastore.
- A virtual machine that was deleted, when its originally mapped RDM was also deleted, or the containing LUN was repurposed and rewritten.
- The RDM to a different virtual machine, even if that virtual machine is on the same host and datastore. Users might do this to access files on the disk, or to test a restore.

When performing a proxy backup of an RDM disk, you must present the same LUN ID to both the ESXi host and the proxy server. (This restriction does not apply to VMFS disk because the virtual disk library reads the VMFS header and matching UUID. But for RDM the host and proxy require the same LUN ID.)
Restoring RDM disks is appropriate if the original virtual machine's VMX file and disk mapping is no longer available, but the LUN containing the RDM is still available. In this case, the RDM image on the LUN might still be valid, so it does not need to be restored. If this is true, do not make changes to the RDM configuration during your restore operations. Instead, complete the restore process in two phases:

- Restore the virtual machine configuration (VMX) and system disk. This restores the virtual machine, but does not restore the RDM.
- Add the RDM disk to the virtual machine. After doing so, you can complete normal restore operations on the RDM disk.

Alternatively, it is possible to create a virtual machine that can host the RDM disk and access its contents. After you create the virtual machine, restore its virtual machine configuration (VMX) from backup, and then restore any selected disks.

Restore the Virtual BIOS or UEFI

The **.nvram** file stores the BIOS or UEFI customizations of a virtual machine. Usually the only important items in this file are the boot drive setting and the boot order (in the case of multiple virtual disks).

Newer releases of vSphere can change the boot order using extended attribute settings, so boot order no longer must be stored in the **.nvram** file. However some users want to preserve a virtual machine's serial port settings in the **.nvram** file, and possibly other items, so applications should back up and restore this information.
To back up and restore NVRAM:

1. For each virtual machine, make a separate copy of the **.nvram** file.
2. Back up each virtual machine using standard methods.
3. If necessary, restore the virtual machine using standard methods.
4. Overwrite the virtual machine's **.nvram** file with the saved copy of the original **.nvram** file.

> **IMPORTANT**
> VMware now recommends saving the **.nvram** file as part of virtual machine backup, a change in recommendation since vSphere 4.1.

Interfacing With VMware vSphere

This section provides pointers to various vSphere programming interfaces.

The VIX API

The VIX API was a popular developer interface for VMware Workstation and other hosted products. It has been deprecated for vSphere. For similar functionality, see Guest Operations in vSphere Web Services.

Virus Scan all Hosted Disk

Suppose you want to run the antivirus software presented in Scan VMDK for Virus Signatures for all virtual machines hosted on a VMware Workstation. Here is the high-level algorithm for an VIX-based application that would scan hosted disk on all virtual machines.
To virus scan hosted virtual disk:

1. Write an application including both the Virtual Disk API and the VIX API.
2. Initialize the virtual disk library with **VixDiskLib_Init()**.
3. Connect VIX to the Workstation host with **VixHost_Connect()**.
4. Call **VixHost_FindItems()** with item-type (second argument) **VIX_FIND_RUNNING_VMS**.

   This provides to a callback routine (fifth argument) the name of each virtual machine, one at a time. To derive the name of each virtual machine's disk, append "`.vmdk`" to the virtual machine name.

5. Write a callback function to open the virtual machine's VMDK.

   Your callback function must be similar to the **VixDiscoveryProc()** callback function shown as an example on the **VixHost_FindItems()** page in the *VIX API Reference Guide*.

6. Instead of printing "Found virtual machine" in the callback function, call the **DoVirusScan()** function shown in Scan VMDK for Virus Signatures.
7. Decontaminate any infected sectors that the virus scanner located.

The vSphere Web Services API

The VMware vSphere Web Services (WS) API is a developer interface for ESXi hosts and vCenter Server. See vSphere Management on the developer site for more information:

https://developer.broadcom.com/sdks/vsphere-management-sdk/latest/

The *Developer's Setup Guide* for the VMware vSphere WS SDK has a chapter describing how to set up your programming environment for Java.

The *Programming Guide* for the vSphere SDK contains some sample code written in Java, and based on the JAX-WS development framework.

ESXi hosts and the VMware vSphere WS API use a programming model based on Web services, in which clients generate Web services description language (WSDL) requests that pass over the network as XML messages encapsulated in simple object access protocol (SOAP). On ESXi hosts or vCenter Server, the vSphere layer answers client requests, usually passing back SOAP responses. This is a different programming model than the object-oriented function-call interface of C++ and the VIX API.

Virus Scan All Managed Disk

Suppose you want to run the antivirus software presented in Scan VMDK for Virus Signatures for all virtual machines hosted on an ESXi host. Here is the high-level algorithm for a VMware vSphere solution that can scan managed disk on all virtual machines.
To virus scan managed virtual disk:

1. Using the VMware vSphere Perl Toolkit, write a Perl script that connects to a given ESXi host.
2. Call **`Vim::find_entity_views()`** to find the inventory of every **`VirtualMachine`**.
3. Call **`Vim::get_inventory_path()`** to get the virtual disk name in its appropriate resource.

   The VMDK filename is available as **`diskPath`** in the **`GuestDiskInfo`** data object.

4. Using Perl's **`system(@cmd)`** call, run the extended `vixDiskLibSample.exe` program with **`-virus`** option.

   For ESXi hosts you must specify **`-host`**, **`-user`**, and **`-password`** options.

5. Decontaminate any infected sectors that the virus scanner located.

Read and Write VMDK Using vSphere API

The VMware vSphere Web Services API (version 2 and later) offers some methods to manage VMDK files. The **`VirtualDiskManager`** managed object defines more than a dozen methods similar to those in the Virtual Disk API documented here.

For more information, navigate to the latest VMware vSphere documentation center on the Web, search for **`VirtualDiskManager`**, and follow the specific link. At top of page you can click Local Methods to see a list of all methods defined by the **`VirtualDiskManager`** managed object.

`VirtualDiskManager` methods are not heavily used, and were not updated to support object oriented file systems such vSAN and vVols, nor storage profile based management (SPBM).

First Class Disk (FCD) Backup

First Class Disk (FCD), also known as Improved Virtual Disk, provides storage lifecycle management on virtual disks, independent of virtual machines. An FCD may be created natively using the FCD interfaces or an existing virtual disk may be promoted to an FCD.

First Class Disk is a named virtual disk not associated with a VM. The vSphere API for handling FCD is called VSLM, virtual storage lifecycle management.

> **NOTE**
> Developers of backup software should implement back up and restore of FCD.

In vSphere 6.5, VDDK supported the backup of FCD in attached mode, but not in detached mode. To back up FCD, programs had to attach the FCD to a dummy VM (such as one without a guest OS) and then back up the VM.

In vSphere 6.7, VDDK supports the backup of detached FCD, in any transport mode. Detached FCD is neither related to nor attached to a VM. It is identified by a combination of its UUID and the datastore ID where it resides. An FCD must be attached to a VM for regular I/O, but not for snapshot based backup.

First Class Disks are identified by UUID. This UUID is globally unique and the primary identifier for the FCD. The UUID remains valid even if its FCD is relocated or snapshotted. FCD operations are performed using the `VStorageObjectManagerBase` managed object, extended for either ESXi hosts or vCenter Server. See the *`vSphere API Reference`* for details.

FCD Workflows and Operations

Create and Delete: FCDs can be created using the `createDisk_Task` method. An existing VMDK can be promoted to FCD using the `registerDisk` method. Once a disk has been registered as an FCD it cannot be unregistered, however registering a disk as an FCD has no impact other than enabling the VSLM APIs to manage them.

Snapshot: FCD snapshots use the same underlying mechanisms as regular snapshots and have the same restrictions as regular snapshots. FCD snapshots are independent of VM snapshots. Each FCD snapshot receives a separate snapshot ID. Snapshots are managed by the VStorageObjectManager specific `createSnapshotEx_Task` and `deleteSnapshot_Task` methods. You can revert an FCD to a given snapshot using the `RevertVStorageObject_Task` method. Reverting an FCD removes all snapshots after the selected snapshot. To access data at the time of the snapshot without discarding subsequent snapshots, you can use the `VStorageObjectManagerCreateDiskFromSnapshot` method or VADP restore.

Backup and Restore: FCDs that are not attached to VMs can be backed up using the VADP interfaces. For backup applications, the best practice is to back up VMDKs and FCDs attached to the VMs, and also back up FCDs that are not attached to a VM.

Discovering FCDs for Backup

FCDs can be listed on a per-datastore basis. The recommended practice is to iterate across the list of datastores from vCenter, get the FCDs for each datastore, then perform a backup of each FCD. VADP can back up both attached and detached FCDs.

Use the `Datacenter` managed object's `datastore` property to list all datastores in a datacenter. For each datastore, use the `listVStorageObject` method to list the FCDs. For each FCD, use the `VStorageObject.getConfig.getId` API to retrieve the FCD's ID. The FCD ID is used in `connParams->vStorageObjSpec.id` below. The FCD snapshot ID is used in the `connParams->vStorageObjSpec.ssid` to specify the snapshot to back up from.

New Connect Parameters and Functions

VDDK 6.7 has a new union in the `VixDiskLibConnectParams` structure, and new function calls to allocate and free the structure, `VixDiskLib_AllocateConnectParams()` and `VixDiskLib_FreeConnectParams()`. The new `typedef` is a union of:

- `vmxSpec`, the traditional specification string for a VM.
- `VixDiskLibVStorageObjectSpec` specifying the FCD's UUID, datastore `MoRef`, and snapshot UUID.

`VixDiskLib_AllocateConnectParams()` is the recommended way to allocate an instance of `VixDiskLibConnectParams`. Programs can allocate `VixDiskLibConnectParams` as before, although then they cannot use new features in 6.7 such as detached FCD.

Except for connection parameters, VDDK code remains the same. All transport modes may be used. Here is sample C++ code to allocate and initialize the connect parameters structure:

```
auto connParams = VixDiskLib_AllocateConnectParams();

connParams->specType = VIXDISKLIB_SPEC_VSTORAGE_OBJECT;

connParams->vStorageObjSpec.id = "XXXXXX" // FCD UUID

connParams->vStorageObjSpec.datastoreMoRef = "datastore-N" // datastore

connParams->vStorageObjSpec.ssid = "XXXXXX" // FCD snapshot UUID

...

VixDiskLib_FreeConnectParams(connParams); // free connect params near end
```

For a `vmxSpec`, here is a sample of legacy C++ code, and a revision:

```
// old:

VixDiskLibConnectParams connParams;

connParams.vmxSpec = "moref=vm-XX"; // vm moref

// new:

auto connParams = VixDiskLib_AllocateConnectParams;

connParams->specType = VIXDISKLIB_SPEC_VMX;

connParams->vmxSpec = "moref=vm-XX"; // vm moref

...

VixDiskLib_FreeConnectParams(connParams); // free connect params near end
```

Subsequently, `VixDiskLib_Open` ignores its disk path parameter because path location is determined by `VixDiskLibVStorageObjectSpec`. The snapshot UUID is specified in `VixDiskLibVStorageObjectSpec`, so the disk path is invisible to the customer anyway. Consequently programs can pass NULL as the disk path parameter to `VixDiskLib_Open`. Programs can also pass an FCD's UUID as the path parameter:

```
VixError

VixDiskLib_Open(const VixDiskLibConnection connection,

                const char *path,

                uint32 flags,

                VixDiskLibHandle *diskHandle);
```

However for the VixMntapi open disks call, passing a null `diskNames` parameter may cause a crash. Programs should pass the FCD's UUID only, and specify `numberOfDisks` = 1.

```
VixError

VixMntapi_OpenDisks(VixDiskLibConnection connection,
                    const char *diskNames[],
                    size_t numberOfDisks,
                    uint32 openFlags,
                    VixDiskSetHandle *handle);
```

FCD Managed Objects and Tasks

The following managed objects and tasks are available in vSphere for manipulating FCD storage. For details, see the *vSphere Web Services API Reference* on https://developer.broadcom.com/xapis under Hyperconverged Infrastructure.

- `HostVStorageObjectManager` manages FCD when connected to an ESXi host.
- `VcenterVStorageObjectManager` manages FCD when connected to a vCenter Server.
- `VirtualMachine.AttachDisk_Task()` attaches FCD to a VM.
- `VirtualMachine.DetachDisk_Task()` detaches FCD from the VM.

> **NOTE**
> The `CreateSnapshotEx_Task` and `RegisterDisk` methods are not supported when an FCD is attached to a template VM. Both throw the `NotSupported` fault in this case.

Behavior of FCD With Changed Block Tracking

The behavior of FCD with changed block tracking (CBT) differs from that of a regular VMDK disk.

1. Attaching an FCD with CBT deactivated (the default) to a VM with CBT enabled causes CBT on the FCD to become enabled. However, detaching that FCD from the VM does not deactivate CBT.
2. Attaching an FCD with CBT enabled to a VM with CBT deactivated throws an error, unless the FCD is attached as "independent nonpersistent" disk.
   Use `ReconfigVM_Task` and connect the disk with the `diskMode` set to `independent_nonpersistent` in the backing info. (`VirtualDiskSeSparseBackingInfo`, `VirtualDiskFlatVer2BackingInfo`, `VirtualDiskRawDiskMappingVer1BackingInfo` and `VirtualDiskSparseVer2BackingInfo` support this mode.)

Behavior of FCD with vMotion

An FCD can be attached to more than one VM.

When your program calls `VixDiskLib_PrepareForAccess`, VDDK deactivates the vSphere API method `RelocateVM_Task` for all attached VMs, one after another. If an error occurs while disabling, VMs whose relocate method was deactivated remain deactivated, whereas subsequent VMs can still be relocated. Programs should call `VixDiskLib_EndAccess` to re-enable the relocate method, regardless of `VixDiskLib_PrepareForAccess` return status. Otherwise vMotion or Storage vMotion could fail for the remaining deactivated VMs.

Code Sample for Connect Parameters

In the development kit, see lines in the `vixDiskLibSample.cpp` program where connection parameters are allocated with the `VixDiskLib_AllocateConnectParams()` call.

In vSphere 6.7 and later, the Web Services SDK contains FCD code samples in Java for creating, attaching, detaching, and deleting FCD.

In the vSphere 6.7 U2 release, Container Native Storage (CNS) supports FCD, and support for more solutions will be added in the future.

Container Storage Snapshot

Cloud Native Storage (CNS) is a vSphere feature that allows Kubernetes to auto-provision scalable storage on demand. CNS also provides vSphere administrator visibility into container volumes through vCenter Server. Container Storage Interface (CSI) is part of CNS, based on First Class Disk (FCD).

This section describes how to create volume snapshots in a Kubernetes cluster and retrieve snapshot handles using CSI.

On Vanilla Kubernetes Cluster

First create a VolumeSnapshotClass as in this YAML example:

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: example-vanilla-rwo-filesystem-snapshotclass
driver: csi.vsphere.vmware.com
deletionPolicy: Delete
```

Run the **`kubectl`** command on this file, `sample_snapshotclass.yaml` in this example:

```
$ kubectl apply -f sample_snapshotclass.yaml
```

Next create a VolumeSnapshot as in this YAML example:

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: example-vanilla-rwo-filesystem-snapshot
  namespace: test-ns
spec:
  volumeSnapshotClassName: example-vanilla-rwo-filesystem-snapshotclass
  source:
    persistentVolumeClaimName: example-vanilla-rwo-pvc
```

Run the **`kubectl`** command on this file, `sample_snapshot01.yaml` in this example:

```
$ kubectl apply -f sample_snapshot01.yaml
```

This will result in VolumeSnapshot and VolumeSnapshotContent API objects. Inspect the Status field of the VolumeSnapshotContent API object to find the SnapshotHandle field.

For example, if the FCD Volume ID and FCD Snapshot ID for a FCD snapshot are the first two, the snapshot handle is a combination of the two separated by plus mark.

```
4ef058e4-d941-447d-a427-438440b7d306

766f7158-b394-4cc1-891b-4667df0822fa

4ef058e4-d941-447d-a427-438440b7d306+766f7158-b394-4cc1-891b-4667df0822fa
```

Similarly you can create a second VolumeSnapshot from the same volume and retrieve the SnapshotHandle from the VolumeSnapshotContent API object.

With the SnapshotHandle of the first and second snapshots, you can retrieve Changed Blocks using the vSphere API, as in standard CBT backup.

TKGS Cluster

If you are on a Tanzu Kubernetes Grid Service (TKGS) cluster, a VolumeSnapshotClass named `volumesnapshotclass-delete` is already created for you.

Create a VolumeSnapshot as in this YAML example:

```
apiVersion: snapshot.storage.k8s.io/v1

kind: VolumeSnapshot

metadata:

  name: example-tkgs-rwo-filesystem-snapshot

  namespace: test-ns

spec:

  volumeSnapshotClassName: volumesnapshotclass-delete

  source:

    persistentVolumeClaimName: example-tkgs-rwo-pvc
```

Run the **`kubectl`** command on this file, `sample_snapshot01.yaml` in this example:

```
$ kubectl apply -f sample_snapshot01.yaml
```

This will result in VolumeSnapshot and VolumeSnapshotContent API objects. Check the following annotation of the VolumeSnapshot API object and retrieve the SnapshotHandle from <snap-id>:

```
"csi.vsphere.volume/snapshot": "<fcd-id>+<snap-id>"
```

Similarly you can create a second VolumeSnapshot from the same volume and retrieve SnapshotHandle from VolumeSnapshot annotation. With SnapshotHandle of the first and second snapshots, you can retrieve Changed Blocks using the vSphere API, as in standard CBT backup.

For more information about the CSI Snapshot feature, see topic "Volume Snapshot and Restore" in the manual *`Getting Started with VMware vSphere Container Storage Plug-in`* on docs.vmware.com.

VADP and VM Service Managed VMs

As of vSphere 8.0 U3, VADP partners can repurpose existing applications to support backup and restore of virtual machines managed by IaaS Platform, formerly Tanzu. IaaS Platform includes two related components: VM Service for the vSphere side and VM Operator for the Kubernetes side.

Prerequisites: VADP partner applications must support first class disk (FCD), often used for Kubernetes storage, and container storage snapshots. FCD support goes back to vSphere 6.7.

VM Service in vSphere enables DevOps engineers to deploy and run containers, in addition to VMs, in a common Kubernetes environment. VMs and containers share the same Namespace resources and are managed with a single interface. VM Service helps developers use Kubernetes alongside non-containerized VM workloads, while allowing vSphere administrators to control service availability and resource consumption. VM Service manages virtual machines in coordination with VM Operator service.

A VM managed by VM Service is much like a regular VM, but includes VM custom resources, bootstrap Secret, and optionally FCD for persistent volume (PV) claims. It runs with Supervisor, which is a platform for natively hosting Kubernetes workloads. Sections below describe how vSphere administrators can perform backup and restore of VM Service managed VMs.

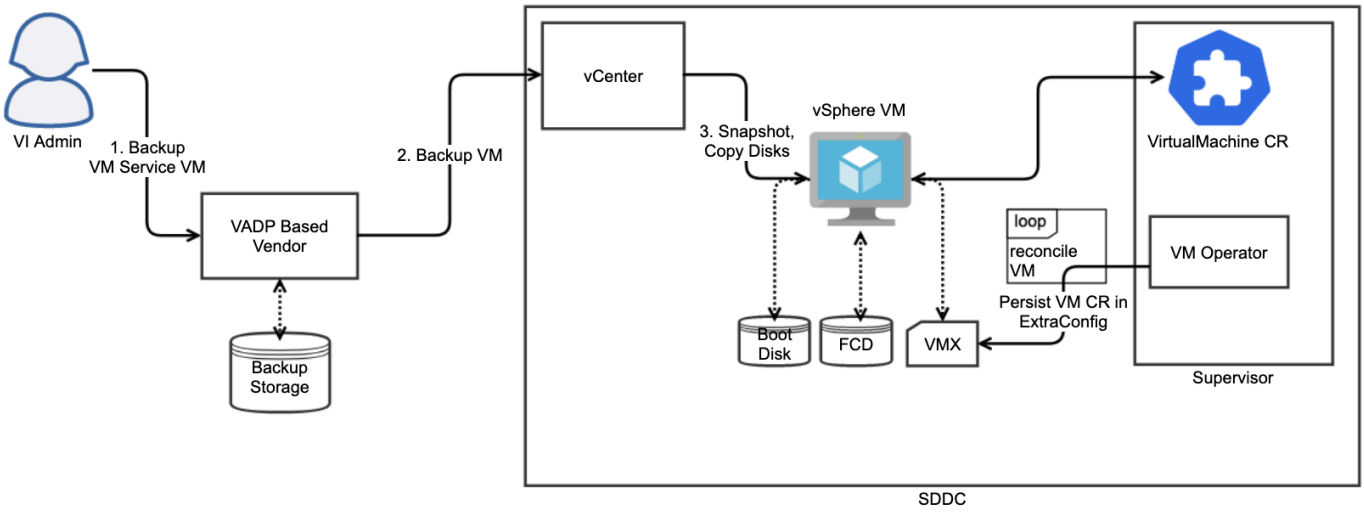
Backup: For a vendor of backup software, a VM managed by VM Service is similar to a regular VM, but also includes custom resources (and so forth) in the `ExtraConfig` section of its VMX file. Backups take place as usual, with administrators setting up an automated schedule for full and incremental backups. When triggered, backup software saves the VM configuration, disk data, and (for VM Service VMs) custom resources, bootstrap Secret, and storage claims.

VM Operator persists virtual machine custom resources (CR), in other words Kubernetes state, in the `ExtraConfig` section of the virtual machine's VMX file. Custom resources include information about paths and types of disks so that disks registered as FCD will be recognized as persistent volumes (PV) during restore operations. This `ExtraConfig` data is base64 encoded and **`gzip`** compressed to conserve space. Only custom resources owned by the DevOps persona are backed up. The following files are involved in `ExtraConfig`:

- `vmservice.virtualmachine.resource.yaml` – Kubernetes state and VM configuration.
- `vmservice.virtualmachine.additional.resources.yaml` – Additional Kubernetes resources needed to re-create the VM, such as a PV claims for volume resources, and the Secret resource required to bootstrap the machine.
- `vmservice.virtualmachine.pvc.disk.data` – Information necessary to register restored disks as PV claims when restoring with the VADP partner application.

As you can see in steps 1 and 2 of Figure: Back up VM Service VM, normal backup operations are unchanged. A VM Service managed VM is exactly like any other VM that is available as a custom resource on Supervisor. This type of VM can also have additional FCD attached to it, expressed as PV claims on Supervisor. Third party software backs up a VM Service VM like any other VM by copying the VMX file and all attached disks along with other necessary files.

Figure 174: Back up VM Service VM



Restore: For a vendor of backup and restore software, restore operations are the same as for a regular VM. For vSphere administrators, restoring a VM Service VM is slightly different. They must be a member of the Administrators group. Upon restore, administrators must choose the Supervisor namespace resource pool where the VM gets restored, and specify a new name, or the same name, for the VM. After restore, automatic registration is triggered, reading `ExtraConfig` previously stored by backup, and restoring state on Supervisor. Registration is task based, so administrators can view progress and results in the vSphere Client. If errors occur, administrators can run `registerVM` as a REST command.

When required, the virtual infrastructure (VI) administrator runs their backup vendor's application to restore the VM in the resource pool corresponding with the Supervisor namespace. The VM should be restored in powered-off state. Ideally, workload control plane (WCP) service detects when a VM is restored in a Supervisor namespace resource pool by a backup application and calls `registerVM`. As a result, a VirtualMachine custom resource is created on Supervisor along with other necessary resources. If a bootstrap resource was specified when creating the VirtualMachine custom resource, the restore process also creates those resources.

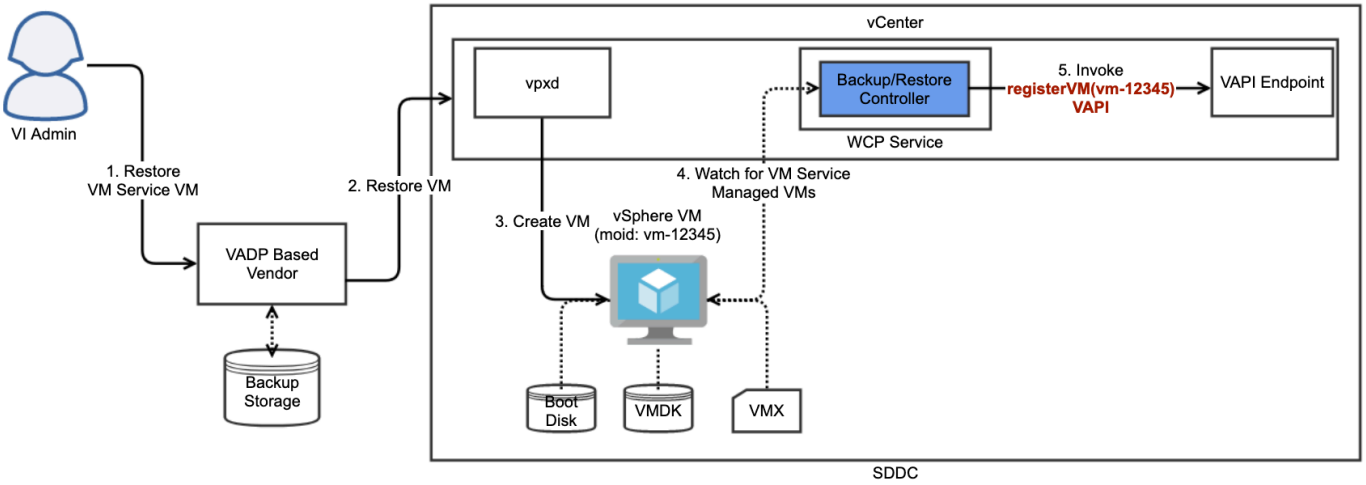
VADP based restore is followed by automatic registration of the restored VM as a VirtualMachine custom resource on Supervisor. Automatic registration is tried only once so as to allow vSphere administrators to correct registration issues in case of failure. If automatic registration fails, the administrator can run the `registerVM` vAPI manually after resolving any issues. One such issue might be Supervisor namespace lacking access to the storage policy being referenced by the VirtualMachine resource.

A common workflow is to restore the VM with a different name, such as with added suffix or prefix. The name of the restored VM takes precedence over the name recorded in `ExtraConfig` during backup.

After naming, the VM Operator applies webhooks to check that the created VirtualMachine resource is a valid configuration. Webhooks checking might fail if the underlying infrastructure changed between backup and restore, for instance if a `StorageClass` associated namespace was removed. To troubleshoot such cases, vAPI returns the exact error received from the server.

As you can see in steps 1 and 2 of Figure: Restore VM Service VM, operations are similar to normal restore, except the VM is managed by VM Service, and its VMDK could be FCD. WCP watches for a newly restored VM and attempts to auto-register it. If all goes well, the administrator does not need to intervene again.

Figure 175: Restore VM Service VM



IP address management of the restored VM is handled as for a new VM, by the VM operator making connection requests to the network provider.

Datasets Backup and Restore

The DataSets facility allows programs to read and write key-value entries related to a virtual machine. This section describes how to back up and restore DataSets files.

There may be one or two DataSets files in a VM directory. If either is present, one has file name suffix `.dsd` while the other has `.dsv` suffix. The former is durable (included in snapshots and clones) while the latter is variable (omitted from snapshots and clones). If either is present, there are corresponding entries in the VM's `.vmx` file:

```
dataSetsMgr.diskStoreFile = filename.dsd

dataSetsMgr.vmStoreFile = filename.dsv
```

The `.dsd` file should be saved by backup and restored along with the VM. The `.dsv` file should not be preserved, but backup software must account for it. There might be other files with extension `.dsd-journal` or `.dsv-journal`. These should not be backed up. If they exist in the VM directory at restore time, they should be removed.

To back up Datasets files, follow these steps:

1. For each virtual machine whose VM directory includes a `.dsd` file, make a separate copy of the `.dsd` file, associated with the VM backup.
2. Do not save copies of any existing `.dsv` files during backup
3. Back up each virtual machine using standard methods (see Low Level Backup Procedures).

To restore DataSets files, follow these steps:

1. Restore the virtual machine using standard methods (see Low Level Restore Procedures).
2. If at the time of the restore the VM directory includes a `.dsv` file and/or a `.dsv-journal` file, remove them. Also, delete any `dataSetsMgr.vmStoreFile` entries in the `.vmx` file.
3. For each VM for which a `.dsd` file was saved at the time of the backup, write the saved `.dsd` file into the VM directory.
   – If a file of the same name exists in the directory before restore, the saved `.dsd` file should overwrite it.

  - If a `.dsd` file with a different name exists in the VM directory before restore, then remove that `.dsd` file. Also, modify the `.vmx` file so its `dataSetsMgr.diskStoreFile` entry reflects the name of the restored `.dsd` file.
  - If no `.dsd` file exists in the VM directory before restore, then add to the `.vmx` file a `dataSetsMgr.diskStoreFile` entry for the restored `.dsd` file.
  - Finally, if there is a `.dsd-journal` file in the VM directory, remove it.
4. For each VM for which a `.dsd` file was **not** saved at the time of backup:
  - If at the time of the restore the VM directory includes a `.dsd` file and/or a `.dsd-journal` file, remove them. Also, delete the `dataSetsMgr.diskStoreFile` entry in the `.vmx` file.

Backing Up Virtual Disks in vSphere

This chapter documents how to write backup and restore software for virtual machines running in vSphere, and contains the following sections about the vSphere Storage APIs – Data Protection (VADP):

For an overview of backup, and help designing your top-level program structure, read the first section below. For details about implementing low-level backup code, read the remaining sections. You should be familiar with virtual machines, snapshots, ESXi, vCenter, and Java.

Design and Implementation Overview

On vSphere, backups are usually done by taking a snapshot, to efficiency obtain a static image of the virtual machine. Snapshots are a view of a virtual machine at a certain point in time, and enable quick and clean backup operation. Snapshots also provide an incremental backup mechanism called changed block tracking.

To back up virtual machines on vSphere, VMware recommends a two-language solution. First use Java to code the backup program that contacts the host, takes a temporary snapshot, records virtual machine configuration, and (later) deletes the snapshot. Then use C++ or C to code the VDDK program that transfers virtual disk data from the snapshot to backup media.

For restore, VMware recommends a two-language solution. First use Java to code the program that instructs the virtual machine to halt, or re-creates the target virtual machine from recorded configuration. Then use C or C++ to code the VDDK program that transfers saved data from backup media to virtual disk.

The VADP Backup Process

These are the high-level steps to back up a virtual machine running in vSphere:

1. Connect to the ESXi host containing the virtual machine targeted for backup. A side-effect of this step is determining the arrangement and description of virtual machines on the host.
2. Tell the host to take a snapshot of the target virtual machine, using the vSphere API. Use the **`quiesce`** flag, but not the **`memory`** flag, which is incompatible with **`quiesce`**. The virtual machine continues to run, while the snapshot provides a static (quiesced) view.
3. Capture the virtual disk data and virtual machine configuration information (**`vim.vm.ConfigInfo`**).
4. On the ESXi host, use the VDDK (programming in C or C++) to open and read the virtual disk and snapshot files. Copy them to backup media, along with configuration information.
5. Tell the host to delete the backup snapshot, using the vSphere API.

Communicating With the Server

In a typical vSphere deployment with many ESXi hosts, an instance of vCenter Server manages the ESXi hosts, and can move virtual machines from host to host (vMotion) to balance load and possibly save electricity by powering off an ESXi host. VMware therefore recommends that backup applications communicate with the vCenter Server instead of with individual ESXi hosts.

The vCenter Server provides location transparency for vSphere Web Services developers. The vCenter Server tracks virtual machines as they move (through vMotion) from one ESXi host to another, and vCenter Server directs SDK operations to the ESXi host that currently runs a virtual machine. Using the vSphere Web Services API, it is possible to back up all the virtual disks associated with a virtual machine.

The handling of the vCenter or an individual ESXi host is essentially equivalent when using the vSphere SDK. With vCenter management, there is no need to contact individual ESXi hosts directly. The remainder of this chapter uses the term vSphere to indicate either a vCenter Server or an ESXi host.

To reduce the resources used by vSphere, VMware recommends that the number of connections (or Sessions) be minimized. It is in the best interests of any program that communicates with vSphere to create one Session and share it with all elements of the program that need to exchange information with vSphere. This means that if your program supports multiple threads, your program should multiplex the use of connection objects by use of access control locks (mutex and the like).

It is also important that all vSphere SDK operations proceed from one instance of the “Session” object that your application requests after logging into vSphere. Using the vSphere API your application can create objects that are “Session specific” and therefore would not be known to other portions of your application that might use a different Session.

Information Containers as Managed Objects

VMware documentation introduces you to the concept of the managed object and its handle, called a managed object reference (**`moRef`**). You might be tempted to get configuration and status information of managed objects using a piecemeal approach. This has the severe disadvantage of creating a lot of chatter over the server connection, so it is very slow. A mechanism has been created to provide status information efficiently: the **`PropertyCollector`**, discussed in PropertyCollector Data.

More About Managed Objects

The documentation for the vSphere API and object model introduces a large number of managed objects. There are five basic types of managed objects that describe the organization of a server. Other managed objects can be considered as details expanding on these five basic types:

- Folder
- Datacenter
- ComputeResource
- ResourcePool
- VirtualMachine

It is a characteristic of all managed objects that they have a **`moRef`** to the managed object that serves as the parent to the managed object. This parent **`moRef`** allows you to reconstruct the object hierarchy exposed by the vSphere SDK. In general the hierarchy is a tree-like structure along the lines of:

Root Folder > Datacenter > ComputeResource > ResourcePool > VirtualMachine

There are variations on this theme, depending on whether you connect to vCenter or directly to an ESXi host, but the overall organization is like the structure above. Each managed object also has a **`Name`** property.

The virtual machine that you want to back up, and the snapshot you take of it (the extensible managed object **`VirtualMachineSnapshot`**) are both designated by their **`moRef`**.

Managed Object References

A managed object reference (**moRef**) is actually a handle and not the managed object itself. While it is certain that a **moRef** always contains a unique value, the unique value is only relative to the instance of vSphere to which you are connected. For example, if vCenter Server manages a cluster of ESXi hosts, each ESXi host maintains its own managed object reference namespace and the vCenter must maintain a managed object reference namespace representing all of its servers. So when an ESXi host is represented by a vCenter, the vCenter must ensure that the managed object references are unique. The vCenter accomplishes this by creating unique managed object reference names inside its own namespace, which differ from the names that ESXi uses for the same managed objects.

A vSphere instance (vCenter or ESXi) tries to keep the **moRef** for a virtual machine consistent across sessions, however consistency is not guaranteed. For example, unregistering and reregistering a virtual machine could result in a change to the **moRef** for the virtual machine. Thus, it is a bad idea to store a **moRef** and expect it to work correctly in future sessions, or with a different vCenter Server.

Unique ID for a Different vCenter

On one vCenter Server, the **moRef** uniquely identifies a virtual machine. If you need to track and inventory virtual machine backups across multiple vCenter Servers, you can use **moRef** together with **instanceUuid**. You can see the **instanceUuid** at the following browser path:

```
https://<vcserver>/mob/?moid=ServiceInstance&doPath=content.about
```

For direct connections to ESXi, the host address and **moRef** uniquely identify a virtual machine. However this **moRef** could be different from the one that vCenter Server returns, hence the fallback to **instanceUuid**. The **instanceUuid** was new in VMware vSphere 4.0. In previous releases, the fallback was to **Uuid**.

Gathering Status and Configuration Information

To save configuration of a virtual machine so you can restore it later, you can use the PropertyCollector to get the virtual machine configuration.

The PropertyCollector is the most efficient mechanism to specify, at the top level, all of the managed objects that are of interest to your application. It has methods for providing updates that indicate only changes to the previous state of these objects. There are two mechanisms for acquiring these updates:

- Polling – Check for changes. The result is either “no change” or an object containing the changes. One advantage of this mechanism is that it involves no network traffic except for a poll request and reporting.
- Wait for updates – “Wait for updates” is basically a blocking call to the **PropertyCollector**. This is only useful if you dedicate a program thread waiting for the call to unblock. The advantage of this mechanism is that there is no traffic on the communications thread unless something must be reported.

The **PropertyCollector** is powerful but requires great attention to detail. Backup-related features of the **PropertyCollector** are covered in Low Level Backup Procedures of this document. The next section provides some background about **PropertyCollector**.

PropertyCollector Data

This document assumes that you want to keep up with changes in the configuration of the vCenter Server, and therefore plan to use the update tracking capability of the **PropertyCollector**.

The **PropertyCollector** requires two fairly complex arguments: the **PropertySpec** and the **ObjectSpec**. The **ObjectSpec** contains instructions to the **PropertyCollector** describing where to look for the desired data. Because configuration information in vSphere is organized like a directory tree, the **ObjectSpec** must describe how to traverse the tree to obtain the desired information. The net result is a complex, nested, and recursive list of instructions. Fortunately, once you have determined the location of all the desired information, the **ObjectSpec** needed to determine the layout of

a vSphere object hierarchy can be a static unvarying object. See the code example in section Understanding an ObjectSpec.

The **`PropertySpec`** is a list of desired property information. Formulating a list that includes all of the desired information can take some effort to compile, but once determined, this can be a static object also.

The data returned from the **`PropertyCollector`** is a container class called **`PropertyFilterUpdate`**, which contains an **`objectSet`** with an item-by-item list of changes to object properties. Every item in this container is identified with one of the following keys: enter (add), leave (delete), and modify. On the first data request, every data item is included, and "enter" is marked for every data item.

The **`PropertyCollector`** presents its results in what amounts to random order. Since all managed objects have a "parent" property, you can reconstruct the configuration hierarchy by building a tree in memory, using the **`parent`** identification to organize. The root folder is identified as the only folder without a parent.

Useful Property Information

In the data returned from **`PropertyCollector`**, you can find most of the information that is useful for backup in the Virtual Machine managed object, including the following:

- Virtual Disks – names, types, and capacities.
- Virtual Machine Type and Configuration – Whatever would be useful in (re)creating a virtual machine. This list might include such information as memory size and number of CPUs.
- Display Names – These names appear in VMware products such as the vSphere Client. You should keep track of these names and correlate them for consistency between your product and VMware products.

VMware supports many virtual disk implementations. The disk implementation type is important because:

- On restore, you should re-create virtual disk with the same disk type as the original virtual machine used.
- A disk backed by a pass-through raw device mapping (RDM) mostly bypasses the ESXi storage stack. You cannot make a snapshot of this virtual disk type. Therefore, you cannot back up pass-through RDM disk using the snapshot method described in this document.

For more information about the Java APIs, read the first several chapters of the *`VMware vSphere Web Services SDK Programming Guide`*, and related pages of the Web-based *`VMware Web Services API Reference Documentation`*. Both are available at https://developer.broadcom.com under APIs and SDKs. Examples in this chapter assume that you have set up the Web Services SDK as described in documentation.

Doing a Backup Operation

After your program obtains information about what is available to back up, it can perform a backup. The three steps to the backup process are:

- Create a Temporary Snapshot on the Target Virtual Machine
- Extract Backup Data from the Target Virtual Machine, and save configuration information.
- Delete the Temporary Snapshot

Prerequisites for Backup

To complete a backup, the calling program requires the permissions shown in Table : Required Permissions to Complete a Backup.

Table 107: Required Permissions to Complete a Backup

Privilege Category	Privilege Subcategory	Privilege
Virtual Machine	Provisioning	Allow Virtual Machine Download
	State	Create Snapshot
		Remove Snapshot
	Configuration	Disk Lease

Create a Temporary Snapshot on the Target Virtual Machine

The low-level procedure for creating a snapshot of a virtual machine is documented in the section Creating a Snapshot. Set the **`quiesce`** flag True to make the file system quiescent, otherwise the snapshot might represent a transitional system state, with inconsistent data. Restoring such data might be destructive.

Another flag named **`memory`** allows you to include in the snapshot a dump of the powered on virtual machine's in-memory state. This is not needed for backup, so set this flag to False.

Changed Block Tracking

This feature, first available in vSphere 4, provides the foundation for incremental (or differential) backup of virtual disks. Your application can back up only changed data as indicated by the **`QueryChangedDiskAreas`** method. Virtual machines with virtual hardware version 7 and later support changed block tracking. These virtual machines contain **`changeTrackingSupported`** in the **`capability`** field of the **`VirtualMachine`** managed object. See Changed Block Tracking on Virtual Disks for details.

Extract Backup Data from the Target Virtual Machine

Associated with the snapshot you just created are “versions” of the virtual disks. To identify these disks, you obtain a **`moRef`** to the snapshot you just created. From this snapshot **`moRef`**, you can extract the disk names and paths. How to do this is demonstrated in section Backing Up a Virtual Disk.

To read the data in a virtual disk, it is necessary to use the **`VixDiskLib`**. This library isolates the programmer from the details of extracting data from a virtual disk and its redo logs. For example, when doing backup you call functions **`VixDiskLib_Open()`** and **`VixDiskLib_Read()`**, among others. **`VixDiskLib`** allows access to disk data on sector boundaries only; the transfer size is some multiple of the disk sector size.

When accessing disks on ESXi hosts, **`VixDiskLib`** release 1.0 transferred virtual disk data over the network. Later **`VixDiskLib`** releases contain API enhancements so you can request more efficient data paths, such as direct SAN access or HotAdding disks to a virtual backup appliance. These efficient data paths requires minor code changes, such as calling **`VixDiskLib_ConnectEx()`** instead of plain connect.

Part of virtual disk information is metadata: a number of key/value pairs describing configuration of the virtual disk. Metadata information can be extracted from a virtual disk using the **`VixDiskLib`** functions **`VixDiskLib_GetMetadataKeys()`** and **`VixDiskLib_ReadMetadata()`**. You should save metadata keys along with the backup, in case you need to re-create the virtual disk.

The **`VixDiskLib`** API allows a backup application to perform a full backup of a virtual machine. The newer **`VixMntapi`** library can extract information about a guest operating system from its virtual disks, so your backup application can determine the type of operating system that is involved. This allows mounting the volumes to device nodes, so your application can perform file-oriented backups and restores.

Delete the Temporary Snapshot

As the last part of the backup process, you should delete the temporary snapshot. It is no longer needed, worsens virtual machine performance, and takes up storage space that could be put to better use.

Restore a Virtual Machine

Your software can follow one of two restore scenarios: either revert to a saved state, or disaster recovery:

To bring an existing virtual machine to a previous state

1. Connect to the server and command it to halt and power off the target virtual machine.
2. Use the server to gain access to the virtual disks. With SAN transport (but not HotAdd or NBDSSL) you must create a snapshot before restoring data.
3. Transfer the disk images from backup using **`VixDiskLib`**. Revert-to and delete the snapshot, if created.

To completely re-create a virtual machine (disaster recovery)

1. Connect to the server.
2. Command the server to create a new virtual machine and its virtual disks using the configuration information saved from **`vim.vm.ConfigInfo`** during backup.
3. Transfer virtual disk data to the newly created virtual disks using **`VixDiskLib`**. Virtual disk data includes disk formatting information, so you do not need to build any kind of file system on the virtual disks.

Doing a Restore Operation

The two scenarios of restore operation are described below.

- Restore an Existing Virtual Machine to a Previous State
- Create a New Virtual Machine

Prerequisites for Restore

To complete a restore, the calling process requires the permissions in Table : Required permissions to complete a restore.

Table 108: Required permissions to complete a restore

Privilege Category	Privilege Subcategory	Privilege
Virtual Machine	Inventory	Create
		Remove
	Configuration	Settings
		Change Resource
Resource		Assign Virtual Machine to Resource Pool

For security reasons, programs are not granted write access to the disks of a running virtual machine. Before you shut it down, you should determine the run-state of the virtual machine.

Run-state information is available from the **`PropertyCollector`**, and if you keep this information up-to-date, your application already knows the run-state of the virtual machine. To change the run-state you must have the **`moRef`** of the virtual machine. Use this **`moRef`** in a **`PowerOnVM_Task`** call through the server connection. For virtual machine shutdown, call the **`PowerOffVM_Task`** method.

Restore an Existing Virtual Machine to a Previous State

The following steps restore a virtual machine to a certain saved state:

1. Shut down the virtual machine (if it is not already shut down).
2. For SAN transport mode, a snapshot is required for restoring virtual disk data, and writes must go to the base disk only. The restore snapshot is optional (not necessary) for NBD(SSL) and HotAdd transport modes. On vVol datastores, if a snapshot was taken, writes must go to the leaf disk because the snapshot volume is read-only.
3. Restore contents of the virtual disk(s). If there were no pre-existing snapshots at backup time, only the snapshot just created, restore only the base disks.

   Restoring disk data requires that you obtain the current names of virtual disks. This process is similar to the one described in Extract Backup Data from the Target Virtual Machine, except in this case you obtain this information directly from the virtual machine and not from a snapshot. The target for the saved disk data must be the actual disk name (including any sequence number) because the current incarnation of a virtual machine may be derived from one or more snapshots.

   Restoring disk data requires use of **`VixDiskLib`**. The **`VixDiskLib_Write()`** function opens the virtual disks so your program can write data. **`VixDiskLib`** functions transfer data to even-sector boundaries only, and transfer length must be an even multiple of the sector size.

   The virtual disk already exists, so it is not necessary to restore the disk configuration information mentioned in Extract Backup Data from the Target Virtual Machine.

4. With SAN transport mode, revert-to and delete the snapshot that you created in step 2. Failing to perform this step with SAN mode could yield a virtual machine that cannot be powered on.

Create a New Virtual Machine

The process of building a virtual machine from backup data involves the following steps:

1. Create the virtual machine.

   To create a new virtual machine, you use the information about virtual machine configuration that you derived and saved during the backup process. You might allow users of restore software an opportunity to rename the virtual machine during recovery in case they want to clone or move the virtual machine. Also you might consider offering them an opportunity to change virtual machine layout (for instance, storing virtual disks on a different datastore). Creating the virtual disks is also done at the time when you create the virtual machine. This process is fairly complicated. See the section Low Level Backup Procedures for details.

2. Restore the virtual disk data.

   This process is similar to restoring the contents of virtual disks (step 3 under Restore an Existing Virtual Machine to a Previous State) with the following exception: you must call the **`VixDiskLib_WriteMetadata()`** function to write all the disk configuration key/value data into the virtual disk before restoring any backed-up data to the virtual disk. Then call **`VixDiskLib_Write()`** to restore the virtual disk data, as described in step 3.

3. Power on the virtual machine.

Access Files on Virtual Disks

It might be necessary for a backup application to access individual files or groups of files on the virtual disks. For example, data protection applications might need to restore individual files on demand.

You can find the interfaces to accomplish this in the **`VixMntapi`** library associated with **`VixDiskLib`**. The **`VixMntapi`** library allows disks or volumes of a virtual machine to be mounted and examined as needed. **`VixMntapi`** provides access at the file system level, whereas **`VixDiskLib`** provides access at the sector level.
To mount a virtual disk

1. Locate the path names of all the virtual disks associated with a snapshot.
2. Call **VixDiskLib_Open()** to open all of these virtual disks. This gives you a number of **VixDiskLib** handles, which you should store in an array.
3. Call **VixMntapi_OpenDiskSet()** to create a **VixDiskSetHandle**, passing in the array of **VixDiskLib** handles that you created in step 2.
4. Pass **VixDiskSetHandle** as a parameter to **VixMntapi_GetVolumeHandles()** to obtain an array of **VixVolumeHandle** pointers to all volumes in the disk set.
5. Call **VixMntapi_GetOsInfo()** to determine what kind of operating system is involved, and decide where important pieces of information are to be found.
6. For important volumes, call **VixMntapi_MountVolume()** then **VixMntapi_GetVolumeInfo()**, which reveals how the volume is set up. (Unimportant volumes include swap partitions.)
7. If you need information about how the guest operating system sees the data on this volume, you can look in the data structure **VixVolumeInfo** returned by **VixMntapi_GetVolumeInfo()**. For example, **VixVolumeInfo::symbolicLink**, obtained using **VixMntapi_GetVolumeInfo()**, is the path on the proxy where you can access the virtual disk's file system using ordinary open, read, and write calls.

Once you are done accessing files in a mounted volume, there are **VixMntapi** procedures for taking down the abstraction that you created. These calls are:

- **VixMntapi_DismountVolume()** for each volume handle
- **VixMntapi_FreeOsInfo()** and **VixMntapi_FreeVolumeInfo()**
- **VixMntapi_CloseDiskSet()**

This leaves the **VixDiskLib** handles that you obtained in the beginning; you must dispose of them properly.

More VADP Details

The preceding sections explained how to contact vSphere and extract information from it, and how to back up or restore virtual disks. The following sections cover the same information at a lower level.

Low Level Backup Procedures

This section describes low level details that may be helpful in coding a backup application. It is not the intent of this material to impose a design, but only to serve as a guideline with examples and exposition. The code samples provided below are not complete. They generally lack error handling and ignore critical details.

Communicate with the Server

Connections to the server machine require credentials: user name, password, and host name (or IP address). The following code connects to the server and extracts information useful for manipulating a service:

1. Create the service instance **moRef**:

```
ManagedObjectReference svcRef = new ManagedObjectReference();

svcRef.setType("ServiceInstance");

svcRef.setValue("ServiceInstance");
```

2. Locate the service:

```
VimServiceLocator locator = new VimServiceLocator();

locator.setMaintainSession(true);
```

```
VimPortType serviceConnection = locator.getVimPort("https://your_server/sdk");
```

3. Log in to the session manager:

```
ServiceInstanceContent serviceContent = serviceConnection.retrieveContent(svcRef);

ManagedObjectReference sessionManager = serviceInstance.getSessionManager();

UserSession us = serviceConnection.login(sessionManager, username, password, null);
```

The PropertyCollector

The **`PropertyCollector`** is used in this section to apply the above details to the backup task.

PropertyCollector Arguments

The **`PropertyCollector`** uses two relatively complicated argument structures. As was mentioned in PropertyCollector Data, these arguments are **`PropertySpec`** and **`ObjectSpec`**. **`PropertySpec`** is a list of the information desired, and **`ObjectSpec`** is a list of instructions indicating where to find the information. In theory, you could directly address an object using its **`moRef`**. In that case an **`ObjectSpec`** can be very simple. However, getting the initial **`moRef`** can be a challenge when a complicated **`ObjectSpec`** is required. To formulate a complex **`ObjectSpec`**, you need to understand the structure of the available data. This is complicated by the fact that an **`ObjectSpec`** can contain recursive elements.

Understanding an ObjectSpec

An **`ObjectSpec`** is a list of **`ObjectSpec`** elements, each specifying an object type, and giving a “selection spec” for the object. More About Managed Objects describes five types of managed objects: Folder, Datacenter, ComputeResource, ResourcePool, and VirtualMachine. VirtualApp (vApp) is a sixth type. You can “traverse” objects, because one managed object leads to another.

- Folder – One of the items contained in the Folder is called **`childEntity`**, which is a list of **`moRef`**s that can contain one of the five managed object types. A Folder can be parent to any of these managed objects.
- Datacenter – This managed object has two items that lead to other managed objects:
  - **`hostFolder`** – A **`moRef`** to a Folder containing a list of ComputeResources comprising a Datacenter.
  - **`vmFolder`** – A **`moRef`** to a Folder containing the VirtualMachines that are part of the Datacenter. If it is your objective to duplicate the display seen in a vSphere Client GUI, then this Folder is of limited use because it does not describe the ResourcePool that is the parent of a virtual machine.
- ComputeResource – A ComputeResource is basically hardware. A ComputeResource can comprise multiple systems. The hardware represents resources that can be used to implement a VirtualMachine object. VirtualMachine is a child of ResourcePool, which controls the sharing of a physical machine's resources among VirtualMachine objects. A ComputeResource contains an item named **`resourcePool`**, which is a **`moRef`** to a ResourcePool.
- VirtualApp – A VirtualApp (vApp) is a collection of VirtualMachines that make up a single application. This is a special form of ResourcePool (defined below). A VirtualApp may have three types of children:
  - VirtualMachine – A folder named **`vm`** contains a list of **`moRef`**s to child VirtualMachines.
  - **`resourcePool`** – A folder containing a list of **`moRef`**s pointing to child ResourcePools or VirtualApps.
    - VirtualApp – A VirtualApp can be composed of other VirtualApps.
    - ResourcePool – You can segment the resources of a VirtualApp using a ResourcePool.
- ResourcePool – This managed object contains two child items:
  - **`resourcePool`** – A folder containing a list of **`moRef`**s pointing to child ResourcePools or VirtualApps.

  - **vm** – A list of **moRef**s to child VirtualMachines that employ the resources of the parent ResourcPool. A VirtualMachine always lists a ResourcePool as its parent.
- VirtualMachine – The VirtualMachine is often considered an “end object” – so you do not need to describe any traversal for this object.

The **ObjectSpec** does not have to lead you any farther than the **moRef** of a target object. You can gather information about the managed object itself using the **moRef** and the **PropertySpec**. This is described in detail in the section Understanding a PropertySpec.

A **TraversalSpec** extends **SelectionSpec**, a property of **ObjectSpec**, and contains the following elements:

- **Path** – The element contained in the object that is used to steer traversal.
- **SelectSet** – An array containing either **SelectionSpec** or **TraversalSpec** elements.
- **Skip** – Whether or not to filter the object in the **Path** element.
- **Type** – The type of object being referenced.
- **Name** – Optional name you can use to reference the **TraversalSpec**, inherited from **SelectionSpec**.

SelectionSpec is a direct target for traversal, as is **TraversalSpec** (a class extending **SelectionSpec**). It is in the **SelectSet** that recursion can occur.

If you wish to traverse the entire configuration tree for a server, then you need only the “root node” **moRef**, which is always a Folder. This root folder **moRef** is available in the property **rootFolder** of the **ObjectSpec** service instance content. All of the above goes into this Java code sample.

```
// Traversal objects can use a symbolic name.

// First we define the TraversalSpec objects used to fill in the ObjectSpec.

//

// This TraversalSpec traverses Datacenter to vmFolder

TraversalSpec dc2vmFolder = new TraversalSpec();

dc2vmFolder.setType("Datacenter");  // Type of object for this spec

dc2vmFolder.setPath("vmFolder");  // Property name defining the next object

dc2vmFolder.setSelectSet(new SelectionSpec[] {"folderTSpec"});

//

// This TraversalSpec traverses Datacenter to hostFolder

TraversalSpec dc2hostFolder = new TraversalSpec();

dc2hostFolder.setType("Datacenter");

dc2hostFolder.setPath("hostFolder");

//

// We use the symbolic name "folderTSpec" which will be defined when we create the
folderTSpec.

dc2vmFolder.setSelectSet(new SelectionSpec[] {"folderTSpec"});

//

// This TraversalSpec traverses ComputeResource to resourcePool

TraversalSpec cr2resourcePool = new TraversalSpec();
```

```
cr2resourcePool.setType("ComputeResource");

cr2resourcePool.setPath("resourcePool");

//

// This TraversalSpec traverses ComputeResource to host

TraversalSpec cr2host = new TraversalSpec();

cr2host.setType("ComputeResource");

cr2host.setPath("host");

//

// This TraversalSpec traverses ResourcePool to resourcePool

TraversalSpec rp2rp = new TraversalSpec();

rp2rp.setType("ResourcePool");

rp2rp.setPath("resourcePool");

//

// Finally, we tie it all together with the Folder TraversalSpec

TraversalSpec folderTS = new TraversalSpec();

folderTS.setName{"folderTSpec");  // Used for symbolic reference

folderTS.setType("Folder");

folderTS.setPath("childEntity");

folderTS.setSelectSet(new SelectionSpec[]{ "folderTSpec",

                                           dc2vmFolder, dc2hostFolder, cr2resourcePool,
rp2rp});

ObjectSpec ospec = new ObjectSpec();

ospec.setObj(startingPoint); // This is where you supply the starting moRef (usually root
folder)

ospec.setSkip(Boolean.FALSE);

ospec.setSelectSet(folderTS); // Attach the TraversalSpec we designed above
```

Understanding a PropertySpec

A **`PropertySpec`** is a list of individual properties that can be found at places identified by the **`ObjectSpec`** and its **`TraversalSpec`**. Once the **`PropertyCollector`** has a **`moRef`**, it can then return the properties associated with that **`moRef`**. This can include “nested” properties. Nested properties are properties that can be found inside of properties identified at the top level of the managed object. Nested properties are identified by a “dot” notation.

An example of nested properties can be drawn from the VirtualMachine managed object.A VirtualMachine has the property identified as **`summary`**, which identifies a **`VirtualMachineSummary`** data object. The **`VirtualMachineSummary`** contains property **`config`**, which identifies a **`VirtualMachineConfigSummary`** data object. The **`VirtualMachineConfigSummary`** has a property called **`name`**, which is a string containing the display name

of the VirtualMachine. You can access this name property using the `summary.config.name` string value. To address all the properties of the **`VirtualMachineConfigSummary`** object, you would use the **`summary.config`** string value.

The **`PropertyCollector`** requires an array of **`PropertySpec`** elements. Each element includes:

- **`Type`** – The type of object that contains the enclosed list of properties.
- **`PathSet`** – An array of strings containing names of properties to be returned, including nested properties.

It is necessary to add an element for each type of object that you wish to query for properties. The following is a code sample of a **`PropertySpec`**:

```
// This code demonstrates how to specify a PropertySpec for several types of target
objects:

PropertySpec folderSp = new PropertySpec();

folderSp.setType("Folder");

folderSp.setAll(Boolean.FALSE);

folderSp.setPathSet(new String [] {"parent", "name"});

PropertySpec dcSp = new PropertySpec();

dcSp.setType("Datacenter");

dcSp.setAll(Boolean.FALSE);

dcSp.setPathSet(new String [] {"parent","name"});

PropertySpec rpSp = new PropertySpec();

rpSp.setType("ResourcePool");

rpSp.setAll(Boolean.FALSE);

rpSp.setPathSet(new String [] {"parent","name","vm"});

PropertySpec crSp = new PropertySpec();

crSp.setType("ComputeResource");

crSp.setAll(Boolean.FALSE);

crSp.set:PathSet(new String [] {"parent","name"});

PropertySpec vmSp = new PropertySpec();

vmSp.setType("VirtualMachine");

vmSp.setAll(Boolean.FALSE);

vmSp.setPathSet(new String [] {"parent",                                    "name",
"summary.config", "snapshot", "config.hardware.device"});

// Tie it all together

PropertySpec [] pspec = new PropertySpec [] {folderSp, dcSp, rpSp, crSp, vmSp};
```

Getting the Data from the PropertyCollector

Now that we have defined **ObjectSpec** and **PropertySpec** (the where and what), we need to put them into a **FilterSpec** that combines the two. An array of **FilterSpec** elements is passed to the **PropertyCollector** (the minimum number of elements is one). Two mechanisms can retrieve data from **PropertyCollector**:

- **RetrieveProperties** – A one-time request for all of the desired properties. This can involve a lot of data, and has no refresh option. **RetrievePropertiesEx** has an additional **options** parameter.
- Update requests – **PropertyCollector** update requests take two forms: polling and waiting (see below).

Requesting Updates

The update method is the way to keep properties up to date. In either Polling or Waiting, it is first necessary to register your **FilterSpec** array object with the **PropertyCollector**. You do this using the **CreateFilter** method, which sends a copy of your **FilterSpec** to the server. Unlike the **RetrieveProperties** method, **FilterSpec** is retained after **CreateFilter** operation. The following code shows how to set **FilterSpec**:

```
// We already showed examples of creating pspec and ospec in the examples above.

// The PropertyCollector wants an array of FilterSpec objects, so:

PropertyFilterSpec fs = new PropertyFilterSpec();

fs.setPropSet(pspec);

fs.setObjectSet(ospec);

PropertyFilterSpec [] fsa = new PropertyFilterSpec [] {fs};

ManagedObjectReference pcRef = serviceContent.getPropertyCollector();

// This next statement sends the filter to the server for reference by the
PropertyCollector

ManagedObjectReference pFilter = serviceConnection.CreateFilter(pcRef, fsa,
Boolean.FALSE);
```

If you wish to begin polling, you may then call the function **CheckForUpdates**, which on the first try (when it must contain an empty string for the version number) returns a complete dump of all the requested properties from all the eligible objects, along with a version number. Subsequent calls to **CheckForUpdates** must contain this version number to indicate to the PropertyCollector that you seek any changes that deviate from this version. The result is either a partial list containing only the changes from the previous version (including a new version number), or a return code indicating no data has changed. The following code sample shows how to check for updates:

```
String updateVersion = "";  // Start with no version

UpdateSet changeData = serviceConnection.CheckForUpdates(pcRef, updateVersion);

if (changeData != nil) {

   updateVersion = changeData.getVersion();  // Extract the version of the data set

}

// ...

// Get changes since the last version was sent.

UpdateSet latestData = serviceConnection.CheckForUpdates(pcRef, updateVersion);
```

If instead you wish to wait for updates to occur, you must create a task thread that blocks on the call **WaitForUpdates**. This task thread would return changes only as they occur and not at any other time. However if the request times out, you must renew it.

> **NOTE**
> The order of property retrieval is not guaranteed. Multiple update requests may be needed.

Extracting Information from the Change Data

The data returned from **CheckForUpdates** (or **WaitForUpdates**) is an array of **PropertyFilterUpdate** entries. Since a **PropertyFilterUpdate** entry is very generic, here is some code showing how to extract information from the **PropertyFilterUpdate**.

```
// Extract the PropertyFilterUpdate set from the changeData

PropertyFilterUpdate [] updateSet = changeData.getFilterSet();

// There is one entry in the updateSet for each filter you registered with the
PropertyCollector.

// Since we currently have only one filter, the array length should be one.

PropertyFilterUpdate myUpdate = updateSet[0];

ObjectUpdate [] changes = myUpdate.getObjectSet();

for (a = 0; a < changes.length; a++) {

     ObjectUpdate theObject = changes[a];

     String objName = theObject.getObj().getMoType().getName();

     // Must decide how to handle the value based on the name returned.

     // The only names returned are names found in the PropertySpec lists.

     // Get propertyName and value ...

}
```

Getting Specific Data

From time to time, you might need to get data that is relevant to a single item. In that case you can create a simple **ObjectSpec** including the **moRef** for the item of interest. The **PropertySpec** can then be set to obtain the properties you want, and you can use **RetrieveProperties** to get the data. Hopefully you can deduce **moRef** from a general examination of the properties, by searching for information from the **rootFolder**.

Identifying Virtual Disks for Backup and Restore

To back up a virtual machine, you first need to create a snapshot. Once the snapshot is created, you then need to identify the virtual disks associated with this snapshot. A virtual machine might have multiple snapshots associated with it. Each snapshot has a virtual “copy” of the virtual disks for the virtual machine. These copies are named with the base name of the disk, and a unique decimal number appended to the name. The format of the number is a hyphen character followed by a 6-digit zero-filled number. An example a disk copy name might be `mydisk-NNNNNN.vmdk` where **NNNNNN** would be some number like: **000032**.

The vSphere API identifies virtual disk files by prefixing the datastore name onto the file system pathname and the filename: `[storageN] myvmname/mydisk-NNNNNN.vmdk`. The name in square brackets corresponds to the short name of the datastore that contains this virtual disk, while the remainder of the path string represents the location relative to the root of this datastore.

To get the name and characteristics of a virtual disk file, you use the **`PropertyCollector`** to select the property: **`config.hardware.device`** from a VirtualMachine managed object. This returns an array of virtual devices associated with a VirtualMachine or Snapshot. You must scan this list of devices to extract the list of virtual disks. All that is necessary is to see if each **`VirtualDevice`** entry extends to **`VirtualDisk`**. When you find such an entry, examine the **`BackingInfo`** property. You must extend the type of the backing property to one of the following, or a **`VirtualMachineSnapshot`** managed object:

- **`VirtualDiskFlatVer1BackingInfo`**
- **`VirtualDiskFlatVer2BackingInfo`**
- **`VirtualDiskRawDiskMappingVer1BackingInfo`**
- **`VirtualDiskSparseVer1BackingInfo`**
- **`VirtualDiskSparseVer2BackingInfo`**

It is important to know which backing type is in use in order to be able to re-create the Virtual Disk.It is also important to know that you cannot snapshot a disk of type **`VirtualDiskRawDiskMappingVer1BackingInfo`**, and therefore you cannot back up this type of Virtual Disk.

The properties of interest are the backing **`fileName`** and the **`VirtualDisk capacityInKB`**. Additionally, when change tracking is in place, you should also save the **`changeID`**.

Creating a Snapshot

Before performing a backup operation, you must create a snapshot of the target virtual machine. Both full and incremental backup rely on the snapshot in vSphere.

With SAN transport on VMFS volumes, the virtual machine should not have any pre-existing snapshots, so that reporting of in-use disk sectors will work. For details see About Changed Block Tracking.

As a best practice, you should search for and delete any pre-existing snapshots with the same name that you selected for the temporary snapshot. These snapshots are possibly remnants from failed backup attempts.

Within a specific snapshot, the names of virtual disk files (with extension `.vmdk`) can be modified with a zero-filled 6-digit decimal sequence number to ensure that the `.vmdk` files are uniquely named. Depending on whether or not the current virtual machine had a pre-existing snapshot, the disk name for a snapshot could have this format: `<diskname>-<NNNNNN>.vmdk`. This unique name is no longer valid after the snapshot is destroyed, so any data for a snapshot disk should be stored in the backup program under its base disk name.

The following code sample shows how to create a snapshot on a specific virtual machine:

```
// At this point we assume the virtual machine is identified as ManagedObjectReference
vmMoRef.

String SnapshotName = "Backup";

String SnapshotDescription = "Temporary Snapshot for Backup";

boolean memory_files = false;

boolean quiesce_filesystem = true;

ManagedObjectReference taskRef =
serviceConnection.getservice().CreateSnapshotEx_Task(vmMoRef,
```

```
                              SnapshotName, SnapshotDescription, memory_files,
quiesce_filesystem);
```

You can use the **taskRef** return value as a **moRef** to track progress of the snapshot operation. After successful completion, **taskRef.info.result** contains the **moRef** of the snapshot.

Backing Up a Virtual Disk

This section describes how to get data from the Virtual Disk after you have identified it. In order to access a virtual disk, you must use the **VixDiskLib**. The following code shows how to initialize the **VixDiskLib** and use it for accessing a virtual disk. All operations require a **VixDiskLib** connection to access virtual disk data. At the present time **VixDiskLib** is not implemented for the Java language, so this code is C++ language:

```
VixDiskLibConnectParams connectParams;

VixDiskLibConnection srcConnection;

connectParams.serverName = strdup("TargetServer");

connectParams.creds.uid.userName = strdup("root");

connectParams.creds.uid.password = strdup("yourPasswd");

connectParams.port = 902;

VixError vixError = VixDiskLib_Init(1, 0, &logFunc, &warnFunc, &panicFunc, libDir);

vixError = VixDiskLib_Connect(&connectParams, &srcConnection);
```

This next section of code shows how to open and read a specific virtual disk:

```
VixDiskLibHandle diskHandle;

vixError = VixDiskLib_Open(srcConnection, diskPath, flags, &diskHandle);

uint8 mybuffer[some_multiple_of_512];

vixError = VixDiskLib_Read(diskHandle, startSector, numSectors, &mybuffer);

// Also getting the disk metadata:

size_t requiredLength = 1;

char *buf = new char [1];

// This next operation fails, but updates "requiredLength" with the proper buffer size

vixError = VixDiskLib_GetMetadataKeys(diskHandle, buf, requiredLength, &requiredLength);

delete [] buf;

buf = new char[requiredLength];  // Create a large enough buffer

vixError = VixDiskLib_GetMetadataKeys(diskHandle, buf, requiredLength, NULL);

// And finally, close the diskHandle:

vixError = VixDiskLib_Close(diskHandle);

// And if you are completely done with the VixDiskLib

VixDiskLib_Disconnect(srcConnection);
```

```
VixDiskLib_Exit();
```

Deleting a Snapshot

When you are done performing a backup, you need to delete the temporary snapshot. You can get the **moRef** for the snapshot from taskRef.info.result as describe above for the create snapshot operation. The following Java code demonstrates how to delete the snapshot:

```
ManagedObjectReference removeSnapshotTask;

ManagedObjectReference snapshot; // Already initialized.

removeSnapshotTask = serviceConnection.getservice().removeSnapshot_Task(snapshot, Boolean
FALSE);
```

Query Allocated Blocks Function

As of vSphere 6.7, developers can call `VixDiskLib_QueryAllocatedBlocks` to accelerate the backup process, especially with large virtual disks. By knowing which blocks are allocated, an application can avoid reading irrelevant sectors and thus reduce data transfer during backup.

Function Signatures

The `VixDiskLib_QueryAllocatedBlocks` function synchronously retrieves a list of allocated blocks. On an open VMDK, it queries for allocated blocks by chunks. A chunk is a defined as a block of consecutive sectors. The calling program can specify chunk size to control granularity, along with starting sector and extent.

```
VixError

VixDiskLib_QueryAllocatedBlocks(VixDiskLibHandle diskHandle,

                                VixDiskLibSectorType startSector,

                                VixDiskLibSectorType numSectors,

                                VixDiskLibSectorType chunkSize,

                                VixDiskLibBlockList **blockList )
```

Parameters

- `diskHandle` [in] Handle to an open virtual disk.
- `startSector` [in] Absolute offset, possibly zero (0).
- `numSectors` [in] Number of sectors to query.
- `chunkSize` [in] Minimum number of sectors covered by a chunk containing data.
- `blockList` [out] Buffer containing a `VixDiskLibBlockList`.

Return: `VIX_OK` on success, otherwise a suitable VIX error code.

Backup programs must call the `VixDiskLib_FreeBlockList` function to deallocate the returned `blockList` when done with it.

```
VixError

VixDiskLib_FreeBlockList(VixDiskLibBlockList *blockList)
```

Parameters

- `blockList` [out] Buffer with above returned `VixDiskLibBlockList`.

Return: `VIX_OK` on success, otherwise a suitable VIX error code.

About Query Allocated Blocks

This function supplements `QueryChangedDiskAreas`, the vSphere API for changed block tracking (CBT). Backup software can start by calling `VixDiskLib_QueryAllocatedBlocks` to get a list of allocated blocks, instead of calling `QueryChangedDiskAreas` with special change ID ("*") to return only active portions of a virtual disk. Calling `VixDiskLib_QueryAllocatedBlocks` is easier, and its implementation is simpler. Consequently the special change ID ("*") is deprecated. It has issues with pre-existing snapshots, and is slow when used on extremely large disks.

Query allocated blocks does not depend on CBT. You can enable CBT either before or after the call. New in vSphere 6.7, `VixDiskLib_QueryAllocatedBlocks` is backward compatible to vSphere 6.0.

> **NOTE**
> On a Windows proxy, a read/write thread that calls `VixDiskLib_QueryAllocatedBlocks` must be created after the `VixDiskLib_InitEx` call. On a Linux proxy there is no such limitation.

The storage system may allocate more space than needed by a guest OS. For example, suppose VMFS block size is 1MB and the guest OS has 4KB block size. When the guest requests 4KB to be written on disk, vSphere allocates 1MB but only 64KB is recorded as changed in CBT (64KB is CBT's minimum sector size). So **`VixDiskLib_QueryAllocatedBlocks`** may report pre-allocated blocks on which nothing is yet written.

If disk size is not a multiple of the chunk size passed to **`VixDiskLib_QueryAllocatedBlocks`**, it may not check whether the final part is allocated. For example, with thick-provisioned disk 1024.25 MB in size, query allocated blocks returns the entire disk (1024.25MB) if the chunk size is 64KB, but if chunk size is 1MB, it returns only 1024MB and does not check that the final 0.25 MB is allocated. VMware recommends that developers treat the remaining 0.25 MB as allocated.

`VixDiskLib_QueryAllocatedBlocks` can be used on object-oriented datastores such as vSAN and vVols (virtual volumes). It is especially helpful for backing up thin-provisioned disks. To skip copying non-allocated regions, programs call `VixDiskLib_QueryAllocatedBlocks` to get the list of allocated blocks returned in the final parameter. This works for large sparsely allocated virtual disks even when CBT is not enabled.

> **NOTE**
> `VixDiskLib_QueryAllocatedBlocks` uses an underlying NFC session (NBD) to get allocated block information. Network connectivity, including DNS address resolution, is required for NFC protocol between the backup proxy and ESXi hosts on port 902. DNS name may not be required if using direct IP addresses without FQDN set on ESXi hosts.

VMware Cloud does not support NBD transport mode, so backup applications on VMware Cloud must use `QueryChangedDiskAreas("*")` instead.

Another limitation is that `VixDiskLib_QueryAllocatedBlocks` returns the whole VMDK for NFS, rather than just the allocated blocks as for VMFS, vSAN, and vVol datastores.

In file transport mode, `VixDiskLib_QueryAllocatedBlocks` depends on the disk library to support local VMDK files. In NBD(SSL) transport mode, an NFC message queries the bitmap of allocated sectors. In SAN and HotAdd transport modes, raw disk in the guest OS does not contain allocated sector information, so the function depends on the aforementioned NFC message to get a bitmap of allocated sectors.

Use Cases for Query Allocated Blocks

To accelerate the initial full backup of allocated disk:

1. Open the VMDK file.
2. Query allocated blocks.
3. Read and save the allocated blocks.
4. Close the VMDK file. Repeat steps for subsequent VMDK files.

For an incremental or differential backup:

1. Call `QueryChangedDiskAreas` with `changeId` of the epoch (point in time of last backup) to find all areas that have changed since the last backup. This is **A1**.
2. Use `VixDiskLib_QueryAllocatedBlocks` to find all valid data sectors. This is **A2**.
3. The intersection of the two sets (A1 ∩ A2) yields the regions that have changed and are allocated. The backup application should save the data in these regions (the intersection).
4. The area A1 minus the intersection (A1 – (A1 ∩ A2)) yields the regions that were changed but unmapped. The backup application can record those regions as zeroed without reading them.

This is highly useful when backing up an `AUTO-UNMAP` enabled VMDK. Since unmapped blocks can be recorded as "changed block" by CBT, without the `VixDiskLib_QueryAllocatedBlocks` call, those unmapped blocks get backed up unnecessarily. With the features together, as described in above, the real changed blocks (namely excluding unmapped) can be calculated.

Code Sample for Query Allocated Blocks

In the development kit, see the `vixDiskLibSample.cpp` program for its sample code related to `chunkSize` . The `DoGetAllocatedBlocks` routine calls the query allocated blocks function, after being requested by the **`-getallocatedblocks`** command line argument.

Changed Block Tracking on Virtual Disks

On ESXi hosts release 4.0 and later, virtual machines can keep track of disk sectors that have changed. This is called changed block tracking. Its method in the VMware vSphere API is **`QueryChangedDiskAreas`**, which takes the following parameters:

- **`_this`** – Managed object reference to the virtual machine.
- **`snapshot`** – Managed object reference to a Snapshot of the virtual machine.
- **`deviceKey`** – Virtual disk for which to compute the changes.
- **`startOffset`** – Byte offset where to start computing changes to virtual disk. The length of virtual disk sector(s) examined is returned in **`DiskChangeInfo`**.
- **`changeId`** – An identifier for the state of a virtual disk at a specific point in time. A new **`ChangeId`** results every time someone creates a snapshot. You should retain this value with the version of change data that you extract (using **`QueryChangedDiskAreas`**) from the snapshot's virtual disk.

When you back up a snapshot for the first time, **`ChangeId`** should be unset, or unsaved, indicating that a baseline (full) backup is required. If you have a saved **`ChangeId`**, it identifies the last time a backup was taken, and tells the changed block tracking logic to identify changes that have occurred since the time indicated by the saved **`ChangeId`**.

There are two ways to get this baseline backup:

- Directly save the entire contents of the virtual disk.
- Call the **`VixDiskLib_QueryAllocatedBlocks`** function, which returns the allocated portions of a virtual disk. Usually this substantially reduces the amount of data to save, especially for thin provisioned and sparse virtual disks.

To summarize, **changeID** is an identifier for a time in the past. It should be a **changeId** string saved at the time when a pre-backup snapshot was taken. If a previous **ChangeId** does exist, then **QueryChangedDiskAreas** returns the disk sectors that changed since the new **ChangeId** was collected. Table : Use of Change ID for Changed Block Tracking shows the algorithm.

Table 109: Use of Change ID for Changed Block Tracking

New Change ID	Old Change ID	Used for Query	Result
change 0	none	null	All sectors, use Query Allocated Sectors instead.
change 1	change 0	change 0	All sectors altered since change 0.

Enable Changed Block Tracking

This feature is deactivated by default, because it reduces performance by a small but measurable amount. If you query the virtual machine configuration, you can determine if it is capable of changed block tracking. Use the property collector to retrieve the capability field from the **VirtualMachineManagedObject**. If the capability field contains the flag **changeTrackingSupported**, then you can proceed. The virtual machine version must be 7 or higher to support this. If the virtual machine version is lower than 7, upgrade the virtual hardware.

If supported, you enable changed block tracking using an abbreviated form of **VirtualMachineConfigSpec**, then use the **ReconfigVM_Task** method to reconfigure the virtual machine with changed block tracking:

```
VirtualMachineConfigSpec configSpec = new VirtualMachineConfigSpec();

configSpec.changeTrackingEnabled = new Boolean(true);

ManagedObjectReference taskMoRef =

    serviceConnection.getService().ReconfigVm_Task(targetVM_MoRef, configSpec);
```

Powered-on virtual machines must go through a stun-unstun cycle (triggered either by power on, migrate, resume after suspend, or snapshot create/delete/revert) before the virtual machine reconfiguration takes effect.
To enable changed block tracking with the vSphere Client:

1. Select the virtual machine and ensure that **Summary › VM Version** says “7” or higher compatibility.
2. In the Summary tab, click **Edit Settings › Options › Advanced › General**.
3. In the right side of the dialog box, click **Configuration Parameters...**
4. In the new dialog box, locate or create a row with name **ctkEnabled**, and set its value to **true** not **false**. See above concerning the stun-unstun cycle.

To enable changed block tracking and back up with the VMware vSphere API:

1. Query change tracking status of the virtual machine. If false, activate changed block tracking.
   ```
   configSpec.changeTrackingEnabled = new Boolean(true);
   ```
2. Create a snapshot of the virtual machine. The snapshot operation causes a stun-unstun cycle.
   ```
   CreateSnapshotEx_Task(VMmoRef, SnapshotName, Description, memory_files,
   quiesce_filesystem);
   ```
3. Starting from the snapshot’s **ConfigInfo**, work your way to the **BackingInfo** of all virtual disks in the snapshot. This gives you the change IDs for all the disks of the virtual machine.
4. Hold onto the change IDs and do a full backup of the snapshot, since this is the first time for backup.
   ```
   VixDiskLib_Read(snapshotDiskHandle, startSector, numSectors, &buffer); /* C not Java
   */
   ```
5. Delete the snapshot when your backup has completed.
   ```
   removeSnapshot_Task(SnapshotName, Boolean FALSE);
   ```

6. Next time you back up this virtual machine, create a snapshot and use **QueryChangedDiskAreas** with the change IDs from your previous backup to take advantage of changed block tracking.
```
changes = theVM.queryChangedDiskAreas(SnapshotMoRef, diskDeviceKey, startPosition,
changeId);
```

Gathering Changed Block Information

Associated with changed block tracking is **changeId**, an identifier for versions of changed block data. Whenever a virtual machine snapshot is created, associated with that snapshot is a **changeId** that functions as a landmark to identify changes in virtual disk data. So it follows that when a snapshot is created for the purpose of creating an initial virtual disk backup, the **changeId** associated with that snapshot can be used to retrieve changes that have occurred since snapshot creation.

To obtain the **changeId** associated with any disk in a snapshot, you examine the “hardware” array from the snapshot. Any item in the devices table that is of type **vim.vm.device.VirtualDevice.VirtualDisk** encloses a class describing the “backing storage” (obtained using **getBacking**) that implements virtual disk. If backing storage is one of the following disk types, you can use the **changeId** property of the **BackingInfo** data object to obtain the **changeId**:

```
vim.vm.device.VirtualDevice.VirtualDiskFlatVer2BackingInfo

vim.vm.device.VirtualDevice.VirtualDiskSparseVer2BackingInfo

vim.vm.device.VirtualDevice.VirtualDiskRawDiskMappingVer1BackingInfo

vim.vm.device.VirtualDevice.VirtualDiskRawDiskVer2BackingInfo
```

Information returned by the **QueryChangedDiskAreas** method is a **DiskChangeInfo** data object containing an array of **DiskChangeInfo.DiskChangeExtent** items that enumerate the start offset and length of various disk areas that changed, and the length and start offset of the entire disk area covered by **DiskChangeInfo**.

When using **QueryChangedDiskAreas** to gather information about snapshots, enable change tracking before taking a snapshot. Attempts to collect information about changes that occurred before change tracking was enabled result in a FileFault error. Enabling change tracking provides the additional benefit of saving space because it enables backup of only information that has changed. If change tracking is not enabled, the entire virtual machine must be backed up each time, rather than incrementally.

Changed block tracking is supported whenever the I/O operations are processed by the ESXi storage stack:

- For a virtual disk stored on VMFS, no matter what backs the VMFS volume (SAN or local disk).
- For a virtual disk stored on NFS (though thick vs thin might be an issue).
- For an RDM in virtual compatibility mode.

When I/O operations are not processed by the ESXi storage stack, changed block tracking is not usable:

- For an RDM in physical compatibility mode.
- A disk that is accessed directly from inside a VM. For example if you are running an iSCSI initiator within the virtual machine to access an iSCSI LUN from inside the VM, vSphere cannot track it.

If the guest actually wrote to each block of a virtual disk (long format or secure erase), or if the virtual disk is thick and eager zeroed, or cloned thick disk, then the query may report the entire disk as being in use.

To find change information, you can use the managed object browser at http://<ESXhost>/mob to follow path **content › rootFolder › datacenter › datastore › vm › snapshot › config › hardware › virtualDisk › backing**. Changed block tracking information (**changeId**) appears in the BackingInfo.

The following C++ code sample assumes that, in the past, you obtained a complete copy of the virtual disk, and at the time when the **changeId** associated with the snapshot was collected, you stored it for use at a later time, which is now. A new snapshot has been created, and the appropriate **moRef** is available:

```
String changeId;      // Already initialized: changeId, snapshotMoRef, theVM

ManagedObjectReference snapshotMoRef;

ManagedObjectReference theVM;

int diskDeviceKey;    // Identifies the virtual disk.

VirtualMachine.DiskChangeInfo changes;

long startPosition = 0;

do {

   changes = theVM.queryChangedDiskAreas(snapshotMoRef, diskDeviceKey, startPosition,
changeId);

   for (int i = 0; i < changes.changedArea.length; i++) {

      long length = changes.changedArea[i].length;

      long offset = changes.changedArea[i].startOffset;

      //

      // Go get and save disk data here

   }

   startPosition = changes.startOffset + changes.length;

} while (startPosition < diskCapacity);
```

In the above code, **`QueryChangedDiskAreas`** is called repeatedly, as **`position`** moves through the virtual disk. This is because the number of entries in the **`ChangedDiskArea`** array could occupy a large amount of memory for describing changes to a large virtual disk. Some disk areas may have no changes for a given **`changeId`**.

The **`changeId`** (changed block ID) contains a sequence number in the form `<UUID>/<nnn>`. If **`<UUID>`** changes, it indicates that tracking information has become invalid, necessitating a full backup. Otherwise incremental backups can continue in the usual pattern.

CBT Enhancements in vSphere 7 and 8

Changed block tracking (CBT) received several important enhancements in vSphere 7.0.

For VMs of recent virtual hardware versions, CBT version 2 uses a smaller and adaptable block size. Although not user visible, this improves the resolution of tracking changes, decreases the amount of backup data, and improves perspectives for block-based recovery. As of vSphere 7.0, this feature is applied automatically when a VM is created or upgraded to hardware version 17 or higher, after CBT set or reset.

If backup customers want to have more than 1024 CBT-enabled disks open simultaneously, they can increase the CBT memory limit. CBT memory configuration is in the vSphere Client under Advanced System Settings, key **`MemCBTBitmapMaxAlloc`**. The allowed range is 128MB to 2048MB. The higher limit might allow as many open CBT disks as the ESXi open disk limit, currently 2048.

The **`bin64/vixDiskCheck`** utility was provided to verify data consistency of CBT enabled incremental backups of a VMDK. In addition to authorization options, similar to options of the VixDiskLib sample program, the **`-cbtdir`** option specifies a temporary directory that should be as large as the VMDK being checked. As of VDDK 8.0.1, the utility also measures I/O performance to help debug customer backup environments. Examples:

```
vixDiskCheck cbtCheck [authOptions] -cbtdir folder -disk disk.vmdk
vixDiskCheck readPerf [authOptions] -cbtdir folder -disk disk.vmdk [-disk disk2.vmdk ...]
vixDiskCheck writePerf [authOptions] -cbtdir folder -disk disk.vmdk [-disk disk2.vmdk ...]
```

CBT Troubleshooting

To debug CBT on a customer site, follow steps in section Troubleshooting Changed Block Tracking (CBT)

Limitations on Changed Block Tracking

Changed block tracking does not work if the virtual hardware version is 6 or earlier, in physical compatibility RDM mode, or when the virtual disk is attached to a shared virtual SCSI bus. ESXi 3.5 supported only up to virtual hardware version 4.

Changed block tracking can be enabled on virtual machines that have disks like these, but when queried for their change ID, these disks always return an empty string. So if you have a virtual machine with a regular system disk and a pass-through RDM as a data disk, you can track changes only on the system disk.

Low Level Restore Procedures

The following sections describe how to recover virtual machines and restore virtual disk data.

- Restoring a Virtual Machine and Disk
- Restore Incremental Backup Data

Restoring a Virtual Machine and Disk

You cannot get write access to a virtual disk that is in active use. For a full restore, you first must ensure that the virtual disk is not in use by halting the parent virtual machine, then performing the “power off” sequence. The following code sample demonstrates how to “power off” a Virtual Machine:

```
// At this point we assume that you have a ManagedObjectReference to the VM - vmMoRef.
// Power on would need a ManagedObjectReference to the host running the VM - hostMoRef.
ManagedObjectReference taskRef = serviceConnection.powerOffVm(vmMoRef);
```

With SAN transport mode, you must create a snapshot of the virtual machine before virtual disk restore. See Creating a Snapshot. If at restore time the virtual machine had a pre-existing snapshot, you must delete it, otherwise SAN mode restore will fail. For other transport modes, the restore snapshot is optional.

With a vVol datastore, the restore snapshot is forbidden when restoring to the parent (backing disk). A null **`MoRef`** is allowed as of vSphere 6.0 to minimize code changes when applications restore to a vVol datastore. To get the unique ID of a vVol object in the MOB (managed object browser) click VirtualMachine > Config > Hardware > Device (VirtualDisk) > Backing.

Table : Summary of restore snapshot requirements summarizes the snapshot requirements.

Table 110: Summary of restore snapshot requirements

	NBDSSL	HotAdd	SAN
Non-vVol datastore	A snapshot is optional. If a snapshot is used, writing to either the leaf disk or the parent disk		A snapshot is required. Writing to the leaf disk is not supported, and writes must only go to

Table continued on next page

Continued from previous page

	NBDSSL	HotAdd	SAN
	is supported. If writing to the parent disk, the snapshot must be reverted and then deleted.		the base disk. The snapshot must be reverted and then deleted.
vVol datastore	A snapshot is optional. If a snapshot is used, writing must be to the leaf disk only.		Not applicable, because SAN transport is not supported on vVol datastores.

In this phase you use VixDiskLib to reload contents of the Virtual Disk, so the following code is C++ not Java:

```
// At this point we assume that you already have a VixDiskLib connection to the server
machine.

uint8 mybuffer[some_multiple_of_512];

int mylocalfile = open("localfile", openflags); // Contains backup copy of virtual disk.

read(mylocalfile, mybuffer, sizeof mybuffer);

vixError = VixDiskLib_Open(srcConnection, path, flags, &diskHandle);

VixDiskLib_Write(diskHandle, startsector, (sizeof mybuffer) / 512, mybuffer);
```

With SAN transport mode, you must revert-to and delete the snapshot. If you forget the snapshot revert, snapshot delete will fail due to CID mismatch, so the virtual machine cannot be powered on. If you forget the snapshot delete, the extraneous snapshot will cause restore problems for subsequent backups.

Creating a Virtual Machine

This section shows how to create a VirtualMachine object, which is complicated but necessary so you can restore data into it. Before creating this object, you must create a **`VirtualMachineConfigSpec`** describing the virtual machine and all of its supporting virtual devices. Almost all the required information is available from the virtual machine property **`config.hardware.device`**, which is a table containing the device configuration information. The relationships between devices are described by the value **`key`**, which is a unique identifier for the device. In turn, each device has a **`controllerKey`**, which is the key identifier of the controller where the device is connected. Use negative integers as temporary key values in the **`VirtualMachineConfigSpec`** to guarantee that temporary key numbers do not conflict with real key numbers when they are assigned by the server. When associating virtual devices with default devices, the **`controllerKey`** property should be reset with the key property of the controller. Below are the settings for a sample **`VirtualMachineConfigSpec`** used to create a virtual machine.

```
// beginning of VirtualMachineConfigSpec, ends several pages later

{

   dynamicType = <unset>,

   changeVersion = <unset>,

//This is the display name of the VM

   name = "My New VM",

   version = "vmx-04",

   uuid = <unset>,

   instanceUuid = <unset>,

   locationId = <unset>,
```

```
// This is advisory, the disk determines the O/S
guestId = "winNetStandardGuest",
   alternateGuestName = "Microsoft Windows Server 2008, Enterprise Edition",
   annotation = <unset>,
   files = (vim.vm.FileInfo) {
      dynamicType = <unset>,
      vmPathName = "[plat004-local]",
      snapshotDirectory = "[plat004-local]",
      suspendDirectory = <unset>,
      logDirectory = <unset>,
   },
   tools = (vim.vm.ToolsConfigInfo) {
      dynamicType = <unset>,
      toolsVersion = <unset>,
      afterPowerOn = true,
      afterResume = true,
      beforeGuestStandby = true,
      beforeGuestShutdown = true,
      beforeGuestReboot = true,
      toolsUpgradePolicy = <unset>,
      pendingCustomization = <unset>,
      syncTimeWithHost = <unset>,
   },
   flags = (vim.vm.FlagInfo) {
      dynamicType = <unset>,
      disableAcceleration = <unset>,
      enableLogging = <unset>,
      useToe = <unset>,
      runWithDebugInfo = <unset>,
      monitorType = <unset>,
      htSharing = <unset>,
      snapshotDisabled = <unset>,
```

```
      snapshotLocked = <unset>,
      diskUuidEnabled = <unset>,
      virtualMmuUsage = <unset>,
      snapshotPowerOffBehavior = "powerOff",
      recordReplayEnabled = <unset>,
   },
   consolePreferences = (vim.vm.ConsolePreferences) null,
   powerOpInfo = (vim.vm.DefaultPowerOpInfo) {
      dynamicType = <unset>,
      powerOffType = "preset",
      suspendType = "preset",
      resetType = "preset",
      defaultPowerOffType = <unset>,
      defaultSuspendType = <unset>,
      defaultResetType = <unset>,
      standbyAction = "powerOnSuspend",
   },
   // the number of CPUs
   numCPUs = 1,
   // the number of memory megabytes
   memoryMB = 256,
   memoryHotAddEnabled = <unset>,
   cpuHotAddEnabled = <unset>,
   cpuHotRemoveEnabled = <unset>,
   deviceChange = (vim.vm.device.VirtualDeviceSpec) [
      (vim.vm.device.VirtualDeviceSpec) {
         dynamicType = <unset>,
         operation = "add",
         fileOperation = <unset>,
         // CDROM
         device = (vim.vm.device.VirtualCdrom) {
            dynamicType = <unset>,
```

```
            // key number of CDROM
            key = -42,
            deviceInfo = (vim.Description) null,
            backing = (vim.vm.device.VirtualCdrom.RemotePassthroughBackingInfo) {
               dynamicType = <unset>,
               deviceName = "",
               useAutoDetect = <unset>,
               exclusive = false,
            },
            connectable = (vim.vm.device.VirtualDevice.ConnectInfo) {
               dynamicType = <unset>,
               startConnected = false,
               allowGuestControl = true,
               connected = false,
            },
            // connects to this controller
            controllerKey = 200,
            unitNumber = 0,
         },
      },
      (vim.vm.device.VirtualDeviceSpec) {
         dynamicType = <unset>,
         operation = "add",
         fileOperation = <unset>,
         // SCSI controller
         device = (vim.vm.device.VirtualLsiLogicController) {
            dynamicType = <unset>,
// key number of SCSI controller
            key = -44,
            deviceInfo = (vim.Description) null,
            backing = (vim.vm.device.VirtualDevice.BackingInfo) null,
            connectable = (vim.vm.device.VirtualDevice.ConnectInfo) null,
```

```
      controllerKey = <unset>,
      unitNumber = <unset>,
      busNumber = 0,
      hotAddRemove = <unset>,
      sharedBus = "noSharing",
      scsiCtlrUnitNumber = <unset>,
   },
},
(vim.vm.device.VirtualDeviceSpec) {
   dynamicType = <unset>,
   operation = "add",
   fileOperation = <unset>,
   // Network controller
   device = (vim.vm.device.VirtualPCNet32) {
      dynamicType = <unset>,
      // key number of Network controller
      key = -48,
      deviceInfo = (vim.Description) null,
      backing = (vim.vm.device.VirtualEthernetCard.NetworkBackingInfo) {
         dynamicType = <unset>,
         deviceName = "Virtual Machine Network",
         useAutoDetect = <unset>,
         network = <unset>,
         inPassthroughMode = <unset>,
      },
      connectable = (vim.vm.device.VirtualDevice.ConnectInfo) {
         dynamicType = <unset>,
         startConnected = true,
         allowGuestControl = true,
         connected = true,
      },
      controllerKey = <unset>,
```

```
      unitNumber = <unset>,
      addressType = "generated",
      macAddress = <unset>,
      wakeOnLanEnabled = true,
   },
},
(vim.vm.device.VirtualDeviceSpec) {
   dynamicType = <unset>,
   operation = "add",
   fileOperation = "create",
   // SCSI disk one
   device = (vim.vm.device.VirtualDisk) {
      dynamicType = <unset>,
      // key number for SCSI disk one
      key = -1000000,
      deviceInfo = (vim.Description) null,
      backing = (vim.vm.device.VirtualDisk.FlatVer2BackingInfo) {
         dynamicType = <unset>,
         fileName = "",
         datastore = <unset>,
         diskMode = "persistent",
         split = false,
         writeThrough = false,
         thinProvisioned = <unset>,
         eagerlyScrub = <unset>,
         uuid = <unset>,
         contentId = <unset>,
         changeId = <unset>,
         parent = (vim.vm.device.VirtualDisk.FlatVer2BackingInfo) null,
      },
      connectable = (vim.vm.device.VirtualDevice.ConnectInfo) {
         dynamicType = <unset>,
```

```
               startConnected = true,
               allowGuestControl = false,
               connected = true,
            },
            // controller for SCSI disk one
            controllerKey = -44,
            unitNumber = 0,
            // size in MB SCSI disk one
            capacityInKB = 524288,
            committedSpace = <unset>,
            shares = (vim.SharesInfo) null,
         },
      },
      (vim.vm.device.VirtualDeviceSpec) {
         dynamicType = <unset>,
         operation = "add",
         fileOperation = "create",
         // SCSI disk two
         device = (vim.vm.device.VirtualDisk) {
            dynamicType = <unset>,
// key number of SCSI disk two
            key = -100,
            deviceInfo = (vim.Description) null,
            backing = (vim.vm.device.VirtualDisk.FlatVer2BackingInfo) {
               dynamicType = <unset>,
               fileName = "",
               datastore = <unset>,
               diskMode = "persistent",
               split = false,
               writeThrough = false,
               thinProvisioned = <unset>,
               eagerlyScrub = <unset>,
```

```
            uuid = <unset>,
            contentId = <unset>,
            changeId = <unset>,
            parent = (vim.vm.device.VirtualDisk.FlatVer2BackingInfo) null,
         },
         connectable = (vim.vm.device.VirtualDevice.ConnectInfo) {
            dynamicType = <unset>,
            startConnected = true,
            allowGuestControl = false,
            connected = true,
         },
         // controller for SCSI disk two
         controllerKey = -44,
         unitNumber = 1,
         // size in MB SCSI disk two
         capacityInKB = 131072,
         committedSpace = <unset>,
         shares = (vim.SharesInfo) null,
      },
   }
},
cpuAllocation = (vim.ResourceAllocationInfo) {
   dynamicType = <unset>,
   reservation = 0,
   expandableReservation = <unset>,
   limit = <unset>,
   shares = (vim.SharesInfo) {
      dynamicType = <unset>,
      shares = 100,
      level = "normal",
   },
   overheadLimit = <unset>,
```

```
   },
   memoryAllocation = (vim.ResourceAllocationInfo) {
      dynamicType = <unset>,
      reservation = 0,
      expandableReservation = <unset>,
      limit = <unset>,
      shares = (vim.SharesInfo) {
         dynamicType = <unset>,
         shares = 100,
         level = "normal",
      },
      overheadLimit = <unset>,
   },
   cpuAffinity = (vim.vm.AffinityInfo) null,
   memoryAffinity = (vim.vm.AffinityInfo) null,
   networkShaper = (vim.vm.NetworkShaperInfo) null,
   swapPlacement = <unset>,
   swapDirectory = <unset>,
   preserveSwapOnPowerOff = <unset>,
   bootOptions = (vim.vm.BootOptions) null,
   appliance = (vim.vService.ConfigSpec) null,
   ftInfo = (vim.vm.FaultToleranceConfigInfo) null,
   applianceConfigRemoved = <unset>,
   vAssertsEnabled = <unset>,
   changeTrackingEnabled = <unset>,
}
// end of VirtualMachineConfigSpec
```

The information above is quite complex, but much of the input consists of defaulted values that are assigned by the system. The remainder of the supplied information can be extracted from the output of the **`config.hardware.device`** table returned from **`PropertyCollector`**. Borrowing heavily from an SDK code example, the following code sets up the configuration specification:

```
// Duplicate virtual machine configuration
VirtualMachineConfigSpec configSpec = new VirtualMachineConfigSpec();
```

```
// Set the VM values
configSpec.setName("My New VM");
configSpec.setVersion("vmx-04");
configSpec.setGuestId("winNetStandardGuest");
configSpec.setNumCPUs(1);
configSpec.setMemoryMB(256);
// Set up file storage info
VirtualMachineFileInfo vmfi = new VirtualMachineFileInfo();
vmfi.setVmPathName("[plat004-local]");
configSpec.setFiles(vmfi);
vmfi.setSnapshotDirectory("[plat004-local]");
// Set up tools config info
ToolsConfigInfo tools = new ToolsConfigInfo();
configSpec.setTools(tools);
tools.setAfterPowerOn(new Boolean(true));
tools.setAfterResume(new Boolean(true));
tools.setBeforeGuestStandby(new Boolean(true));
tools.setBeforeGuestShutdown(new Boolean(true));
tools.setBeforeGuestReboot(new Boolean(true));
// Set flags
VirtualMachineFlagInfo flags = new VirtualMachineFlagInfo();
configSpec.setFlags(flags);
flags.setSnapshotPowerOffBehavior("powerOff");
// Set power op info
VirtualMachineDefaultPowerOpInfo powerInfo = new VirtualMachineDefaultPowerOpInfo();
configSpec.setPowerOpInfo(powerInfo);
powerInfo.setPowerOffType("preset");
powerInfo.setSuspendType("preset");
powerInfo.setResetType("preset");
powerInfo.setStandbyAction("powerOnSuspend");
// Now add in the devices
VirtualDeviceConfigSpec[] deviceConfigSpec = new VirtualDeviceConfigSpec [5];
```

```
configSpec.setDeviceChange(deviceConfigSpec);

// Formulate the CDROM

deviceConfigSpec[0].setOperation(VirtualDeviceConfigSpecOperation.add);

VirtualCdrom cdrom = new VirtualCdrom();

VirtualCdromIsoBackingInfo cdDeviceBacking = new
 VirtualCdromRemotePassthroughBackingInfo();

cdDeviceBacking.setDatastore(datastoreRef);

cdrom.setBacking(cdDeviceBacking);

cdrom.setKey(-42);

cdrom.setControllerKey(new Integer(-200)); // Older Java required type for optional
properties

cdrom.setUnitNumber(new Integer(0));

deviceConfigSpec[0].setDevice(cdrom);

// Formulate the SCSI controller

deviceConfigSpec[1].setOperation(VirtualDeviceConfigSpecOperation.add);

VirtualLsiLogicController scsiCtrl = new VirtualLsiLogicController();

scsiCtrl.setBusNumber(0);

deviceConfigSpec[1].setDevice(scsiCtrl);

scsiCtrl.setKey(-44);

scsiCtrl.setSharedBus(VirtualSCSISharing.noSharing);

// Formulate SCSI disk one

deviceConfigSpec[2].setFileOperation(VirtualDeviceConfigSpecFileOperation.create);

deviceConfigSpec[2].setOperation(VirtualDeviceConfigSpecOperation.add);

VirtualDisk disk =  new VirtualDisk();

VirtualDiskFlatVer2BackingInfo diskfileBacking = new VirtualDiskFlatVer2BackingInfo();

diskfileBacking.setDatastore(datastoreRef);

diskfileBacking.setFileName(volumeName);

diskfileBacking.setDiskMode("persistent");

diskfileBacking.setSplit(new Boolean(false));

diskfileBacking.setWriteThrough(new Boolean(false));

disk.setKey(-1000000);

disk.setControllerKey(new Integer(-44));

disk.setUnitNumber(new Integer(0));
```

```
disk.setBacking(diskfileBacking);

disk.setCapacityInKB(524288);

deviceConfigSpec[2].setDevice(disk);

// Formulate SCSI disk two

deviceConfigSpec[3].setFileOperation(VirtualDeviceConfigSpecFileOperation.create);

deviceConfigSpec[3].setOperation(VirtualDeviceConfigSpecOperation.add);

VirtualDisk disk2 =  new VirtualDisk();

VirtualDiskFlatVer2BackingInfo diskfileBacking2 = new VirtualDiskFlatVer2BackingInfo();

diskfileBacking2.setDatastore(datastoreRef);

diskfileBacking2.setFileName(volumeName);

diskfileBacking2.setDiskMode("persistent");

diskfileBacking2.setSplit(new Boolean(false));

diskfileBacking2.setWriteThrough(new Boolean(false));

disk2.setKey(-100);

disk2.setControllerKey(new Integer(-44));

disk2.setUnitNumber(new Integer(1));

disk2.setBacking(diskfileBacking2);

disk2.setCapacityInKB(131072);

deviceConfigSpec[3].setDevice(disk2);

// Finally, formulate the NIC

deviceConfigSpec[4].setOperation(VirtualDeviceConfigSpecOperation.add);

com.VMware.vim.VirtualEthernetCard nic =  new VirtualPCNet32();

VirtualEthernetCardNetworkBackingInfo nicBacking = new
VirtualEthernetCardNetworkBackingInfo();

nicBacking.setNetwork(networkRef);

nicBacking.setDeviceName(networkName);

nic.setAddressType("generated");

nic.setBacking(nicBacking);

nic.setKey(-48);

deviceConfigSpec[4].setDevice(nic);

// Now that it is all put together, create the virtual machine.

// Note that folderMo, resourcePool, and hostMo, are moRefs to

// the Folder, ResourcePool, and Host where the VM is to be created.
```

```
ManagedObjectReference taskMoRef =

    serviceConnection.getService().createVM_Task(folderMo, configSpec, resourcePool,
hostMo);
```

Using the VirtualMachineConfigInfo

A backup application can also use information contained in a **VirtualMachineConfigInfo**. If at backup time you preserve all the **VirtualMachineConfigInfo** details that describe the virtual machine, you can transfer much of this information into a **VirtualMachineConfigSpec** to create a virtual machine at restore time. However, some of the information in **VirtualMachineConfigInfo** is not needed, and if used in the **Spec**, virtual machine creation can fail. For example, a **VirtualMachineConfigSpec** that contains information about so called “Default Devices” usually fails. The list of default devices includes:

```
vim.vm.device.VirtualIDEController

vim.vm.device.VirtualPS2Controller

vim.vm.device.VirtualPCIController

vim.vm.device.VirtualSIOController

vim.vm.device.VirtualKeyboard

vim.vm.device.VirtualVMCIDevice

vim.vm.device.VirtualPointingDevice
```

However, other controllers and devices must be explicitly included in the **VirtualMachineConfigSpec**.

Some information about devices is unneeded and can cause problems if supplied. Each controller device has its **vim.vm.device.VirtualController.device** field, which is an array of devices that report to the controller. The server rebuilds this list when a virtual machine is created, using the (negative) device key numbers supplied as a guide. The relationship between controller and device must be preserved using negative key numbers in the same relationship as in the hardware array of **VirtualMachineConfigInfo**.

The parent property for virtual disk backing information must be set to null. In the sample code for creating a virtual machine, find **vim.vm.device.VirtualDisk.FlatVer2BackingInfo** under SCSI disk one and SCSI disk two. The null setting is required because the pre-backup snapshot causes the parent property to be populated with a reference to the base disk.

One other configuration needs substitution. **VirtualMachineConfigInfo** contains the **cpuFeatureMask**, field, which is an array of **HostCpuIdInfo**. The array entries must be converted to **ArrayUpdateSpec** entries containing the **VirtualMachineCpuIdInfoSpec** along with the “operation” field, which must contain the value **ArrayUpdateOperation::add**. The **VirtualMachineCpuIdInfoSpec** also contains a **HostCpuIdInfo** array that you can copy from the **cpuFeatureMask** array in **VirtualMachineConfigInfo**. These items are not reflected in the sample code. Everything else can be copied intact from **VirtualMachineConfigInfo** data.

To summarize: when creating a virtual machine in which to restore virtual disk:

- Exclude default devices, and **VirtualController.device**, from the **VirtualMachineConfigSpec**.
- Set the parent virtual disk backing information (**VirtualDisk.FlatVer2BackingInfo**) to null.
- Convert **HostCpuIdInfo** array entries to **ArrayUpdateSpec**, insert **ArrayUpdateOperation::add**, and copy the **HostCpuIdInfo** array from **cpuFeatureMask** into **VirtualMachineConfigInfo**.

Editing or Deleting a Device

If backup clients want to edit or delete a device, they must use the server-provided key when referring to an existing device. For the definition of **key**, see Creating a Virtual Machine. For example, see the **key** and **controllerKey** below comments in the source code under CDROM. The **key** uniquely identifies a device, while the **controllerKey** uniquely identifies the controller where it is connected.

Restoring Virtual Disk Data

As in the section Low Level Restore Procedures, **VixDiskLib** functions provide interfaces for writing the data to virtual disk, either locally or remotely.

Raw Device Mapping (RDM) Disks

To create an RDM disk using **CreateVM_Task**, use a LUN that is not occupied and thus is still available. Developers sometimes use the same LUN **uuid** that is available in the **configInfo** object, which can cause errors because the LUN **uuid** is datastore specific.

Call **QueryConfigTarget** to fetch the **ConfigTarget.ScsiDisk.Disk.CanonicalName** property, set in **VirtualDiskRawDiskMappingVer1BackInfo.deviceName**. Also call **QueryConfigTarget** to fetch `ConfigTarget.ScsiDisk.Disk.uuid`, set in **VirtualDiskRawDiskMappingVer1BackInfo.lunUuid**. When creating the virtual machine, avoid host-specific properties of **configInfo**, which should be set according to host configuration where the virtual machine is restored.

Restore Incremental Backup Data

At some point you might need to restore a virtual disk from the backup data that you gathered as described in Changed Block Tracking on Virtual Disks. The essential procedure is as follows:

1. Power off the virtual machine, if powered on.
2. Using **VirtualMachineConfigInfo** that corresponds to the last known good state of the guest operating system, re-create the virtual machine as described in Using the VirtualMachineConfigInfo.
3. Completely reload the base virtual disk using the full backup that started the most recent series of incremental backups.
4. Create a snapshot. This is mandatory for SAN mode restore.

   A restore snapshot is forbidden when restoring to the parent (backing) disk on a vVol datastore.

5. For SAN mode restore, deactivate changed block tracking. SAN writes are not possible with CBT enabled.
6. Sequentially restore the incremental backup data. You can do this either forwards or backwards. If you work forwards, the restore might write some sectors more than once. If you work backwards, you must keep track of which sectors were restored so as to avoid restoring them again from older data.
   a) From your backup records, get the change ID of the incremental backup to be restored. Your software must also store the changed-block information, so it knows which sectors of virtual disk to restore. Once you start restoring virtual disk, the change tracking mechanism will misreport.
   b) Restore only changed areas to the virtual disks referred to by the snapshot. This ensures that you do not write the data to the redo log created by the snapshot. When restoring a thin provisioned (sparse) disk, avoid writing zeroes to the unallocated blocks.
   c) Repeat steps a and b as necessary by applying incremental backup data sets in order.
7. If applicable (SAN mode) revert to the base virtual disk, thus eliminating the snapshot.

Restore with Direct Connection to ESXi Host

Sometimes you must restore a virtual machine directly to an ESXi host, for example in disaster recovery when vCenter Server runs on ESXi as a virtual machine. A new vSphere 5 feature tries to prevent this if the ESXi host is managed by vCenter. To circumvent this and restore the virtual machine, you must first disassociate the host from vCenter. In earlier releases, vCenter management had less state but was revocable only from vCenter.

1. Using the vSphere Client, connect directly to the ESXi 5.0 or later host.
2. In the Inventory left-hand panel, select the host. In the right-hand panel, click Summary.
3. In the box titled Host Management, click **Disassociate host from vCenter Server**. You do not need to put the host in Maintenance Mode.
4. After the vCenter Server has been restored and is back in service, use it to reacquire the host.

   Currently there is no API to disassociate a host from vCenter Server.

Tips and Best Practices

This section discusses the various advanced transport mechanisms, and other backup issues.

VDDK 5.0 contained two new VixDiskLib calls (**`PrepareForAccess`** and **`EndAccess`**) to deactivate and re-enable Storage vMotion during backup. This prevents stale disk images from being left behind if a virtual machine has its storage moved while a backup is taking place. VMware strongly recommends use of these calls.

When an ESXi host is managed by vCenter Server, vSphere API calls cannot contact the host directly: they must go through vCenter. If necessary, especially during disaster recovery, the administrator must disassociate the ESXi host from vCenter Server before the host can be contacted directly.

Advanced transports allow programs to transfer data in the most efficient manner. SAN transport is available only when the physical-machine host has SAN access. HotAdd works for the appliance model, where backup is done from inside virtual machines. HotAdd requires the virtual machine datastore to be accessible from the backup appliance. NBDSSL is a secure fallback when over-the-network backup is your only choice.

SAN transport is supported only on physical machines, and HotAdd transport is supported only on virtual machines. SAN requires a physical proxy to share a LUN with the ESXi host where a datastore resides, enabling direct access to raw data, and bypassing the host altogether for I/O operations. HotAdd involves attaching a virtual disk to the backup proxy just like attaching the disk to a virtual machine.

Best Practices for HotAdd Transport

In HotAdd transport, virtual disks on backed-up VMs are HotAdded to the backup proxy so they can be easily saved to backup media.

Deploy the proxy on VMFS-6 or VMFS-5 volumes, so that the proxy can back up very large virtual disks. HotAdd is a SCSI feature and does not work for IDE disks. The paravirtual SCSI controller (PVSCSI) is the recommended default for HotAdd, but other controller types work too. You might want to include checks in your code to detect that your HotAdd appliance has controllers in the correct order.

The HotAdd implementation assumes that proxy can access (connect to) target VMs. This is so VADP can obtain a list of all disks on target VMs from the connection.

A redo log is created for HotAdded disks, on the same datastore as the base disks. Do not remove the target virtual machine (the one being backed up) while HotAdded disk is still attached. If removed, HotAdd fails to properly clean up redo logs so virtual disks must be removed manually from the backup appliance. Also, do not remove the snapshot until after cleanup. Removing it could result in an unconsolidated redo log.

HotAdded disks should be released with **`VixDiskLib_Cleanup()`** before snapshot delete. Cleanup might cause improper removal of the change tracking (**`ctk`**) file. You can fix it by power cycling the virtual machine.

Virtual disk created on Windows by HotAdd backup or restore may have a different disk signature than the original virtual disk. The workaround is to reread or rewrite the first disk sector in NBDSSL mode. Customers running a Windows Server 2008 or later proxy should make sure Windows automount is deactivated.

For unbuffered HotAdd restore, VMware recommends that programmers set the VDDK flag **`VIXDISKLIB_FLAG_OPEN_UNBUFFERED`** when opening virtual disks before performing a restore with HotAdd transport. In vSphere 6.7 releases and later, programs must allocate a data buffer whose memory address is sector size aligned when setting this flag. Programmers may refer to **`posix_memalign`** on Linux or **`_aligned_malloc`** on Windows. Buffer address alignment to sector size is recommended for older VDDK releases as well.

In the rare case that **`VixDiskLib_Close()`** fails with error **`VIX_E_HOST_SERVER_NOT_AVAILABLE`** in HotAdd transport mode, the backup application should run **`VixDiskLib_Cleanup()`** and notify the administrator to manually remove the snapshot after the host exits maintenance mode, then retry the backup.

Best Practices for SAN Transport

For array-based storage, SAN transport is often the best performing choice for backups when running on a physical proxy. It is unavailable inside virtual machines, so use SCSI HotAdd instead on a virtual proxy.

SAN transport is not always the best choice for restores. It offers the best performance on thick disks, but the worst performance on thin disks, because of round trips through two disk manager APIs, **`AllocateBlock`** and **`ClearLazyZero`**. For thin disk restore, NBDSSL is usually faster. Changed Block Tracking (CBT) must be deactivated for SAN restores, and a snapshot must be taken before restore. SAN transport does not support writing to redo logs (child disks including linked clones and snapshots), only to base disks. Finally, SAN transport is not supported on vVol datastores.

Provided the disk LUN backing an NVMe-oF datastore is accessible from the physical machine running backups, SAN transport mode might be supported. However, VDDK searches only SCSI LUNs during initialization, so backup administrators would probably have to specify all NVMe-oF disks in `vixDiskLib.transport.san.allowList` (see Initialize Virtual Disk API).

Before vSphere 5.5, when writing to SAN during restore, disk size had to be a multiple of the underlying VMFS block size, otherwise the write to the last fraction of a disk would fail. This was fixed in the ESXi 5.5 release.

Programs that open a local virtual disk in SAN mode might be able to read (if the disk is empty) but writing will throw an error. Even if programs call **`VixDiskLib_ConnextEx()`** with NULL parameter to accept the default transport mode, SAN is selected as the preferred mode if SAN storage is connected to the ESXi host. VixDiskLib should, but does not, check SAN accessibility on open. With local disk, programs must explicitly request NBDSSL mode.

For a Windows Server 2008 and later proxy, set SAN policy to **`onlineAll`**. Set SAN disk to read-only except for restore. You can use the diskpart utility to clear the read-only flag. SAN policy varies by Windows Server 2008 edition. For Enterprise and Datacenter editions, the default Windows SAN policy is **`offline`**, which is unnecessary when vSphere mediates SAN storage.

For SAN transport, one major factor impacting performance is that the read buffer should be aligned with the sector size, currently 512. You can specify three parameters for **`VixDiskLib_Read`**: start sector, number of sectors to read, and the buffer to hold data. The proper read buffer size can be allocated using, for example, **`_aligned_malloc`** on Windows or **`posix_memalign`** on Linux. SAN mode performs best with about six concurrent streams per ESXi host; more than six streams usually results in slower total throughput.

VDDK 8.0.1 and later provide a cache file to assist SAN disk device discovery. Before doing a SAN mode backup, each instance of VDDK scans disks to determine the device name and VMFS ID of the LUN. In large datacenters with many SAN based disks, this had a significant negative performance impact. Partners requested a cache mechanism to reduce frequency of scans. As implemented, cache file `scsiDiskList.json` is created in the `tmpDirectory` specified by the VDDK configuration file. When VDDK fails to open a disk with cached LUN path, because disk was added or deleted since

last scan, LUNs are rescanned and the JSON file overwritten. This cache mechanism operates without customer intervention.

Best Practices for NBD Transport

NBD (network block device) is the most universal of VDDK transport modes. It does not require dedicated backup proxy VMs as does HotAdd, and works on all datastore types, not just SAN. Sections below give tips for improving NFC (network file copy) performance for NBD backups.

Parallel jobs on NFC servers: On ESXi hosts, **`hostd`** contains an NFC server. The **`hostd`** NFC memory limit is 96MB. VMware recommends backing up 50 or fewer disks in parallel through each ESXi host. NFC cannot handle too many requests at the same time. It will queue requests until previous jobs complete.

Dedicated backup network: As of vSphere 7.0, ESXi hosts support a dedicated network for NBD transport. When the tag `vSphereBackupNFC` is applied to a VMkernel adapter's NIC type, NBD backup traffic goes through the chosen virtual NIC. Programmers can apply the tag by making the following vSphere API call:
`HostVirtualNicManager->SelectVnicForNicType(nicType,device);`

Customers can use an ESXi command like this, which designates interface `vmk2` for NBD backup:

```
esxcli network ip interface tag add -t vSphereBackupNFC -i vmk2
```

Network I/O Control (NIOC) for NFC backup: When NIOC is enabled in the virtual distributed switch (VDS or DVS), switch traffic is divided into various predefined network resource pools, now including one dedicated to vSphere Backup NFC. The API enumeration for this network resource pool is `VADP_NIOConBackupNfc`. System administrators can set this up in the vSphere Client with **System Traffic > Configure > Edit**, then optionally change resource settings. Thereafter any VADP NBD traffic is shaped by these VDS settings. NIOC may be used together with the dedicated backup network feature above, but this is not a requirement.

VDDK 7.0.1 introduced two new error codes, `VIX_E_HOST_SERVER_SHUTDOWN` and `VIX_E_HOST_SERVER_NOT_AVAILABLE`, to indicate entering maintenance mode (EMM) and in maintenance mode. After `VixDiskLib_ConnectEx` to vCenter, if the backup application calls `VixDiskLib_Open` for a virtual disk on an EMM host, vCenter switches to a different host if possible. Host switch is non-disruptive; backup continues. If it's too late for host switch, vCenter returns the `SHUTDOWN` code, saying the backup application should retry after a short delay, hoping for host switch. If no other hosts are available and the original host is in maintenance mode, vCenter returns `NOT_AVAILABLE`. The backup application may choose to wait, or fail the backup.

Error Code	Retry	Comment
`VIX_E_HOST_NETWORK_CONN_REFUSED`	Frequently	Usually caused by network error.
`VIX_E_HOST_SERVER_SHUTDOWN`	Soon, 3 times	Host will enter maintenance mode (EMM).
`VIX_E_HOST_SERVER_NOT_AVAILABLE`	After waiting?	Host is in maintenance mode (post EMM).

Host switch to avoid EMM could fail if encryption keys are not shared among hosts.

NFC compress flags: In vSphere 6.5 and later, NBD performance can be significantly improved using data compression. Three types are available (zlib, fastlz, and skipz) specified as flags when opening virtual disks with the `VixDiskLib_Open()` call. Data layout may impact the performance of these different algorithms.

- `VIXDISKLIB_FLAG_OPEN_COMPRESSION_ZLIB` – zlib compression

- `VIXDISKLIB_FLAG_OPEN_COMPRESSION_FASTLZ` – fastlz compression
- `VIXDISKLIB_FLAG_OPEN_COMPRESSION_SKIPZ` – skipz compression

Asynchronous I/O: In vSphere 6.7 and later, asynchronous I/O for NBD transport mode is available. It can greatly improve data transfer speed of NBD transport mode. To implement asynchronous I/O for NBD, use the new functions `VixDiskLib_ReadAsync()` and `VixDiskLib_WriteAsync()` with callback `VixDiskLib_Wait()` to wait for all asynchronous operations to complete. In the development kit, see `vixDiskLibSample.cpp` for code examples, following the logic for `-readasyncbench` and `-writeasyncbench` options.

Many factors impact write performance. Network latency is not necessarily a significant factor. Here are test results showing improvements with VDDK 6.7:

- stream read over 10 Gbps network with asynchronous I/O, speed of NBD is ~210 MBps
- stream read over 10 Gbps network with block I/O, speed of NBD is ~160 MBps
- stream write over 10 Gbps network with asynchronous I/O, speed of NBD is ~70 MBps
- stream write over 10 Gbps network with block I/O, speed of NBD is ~60 MBps

I/O buffer improvements: As of vSphere 7.0, changed block tracking (CBT) has adaptable block size and configurable VMkernel memory limits for higher performance. This feature requires no developer intervention and is transparent to users. It is applied automatically by vSphere when a VM is created or upgraded to hardware version 17, after CBT set or reset. Adaptable block size is up to four times more space efficient.

As of VDDK 7.0.3, users can configure asynchronous I/O buffers for NBD or NBDSSL transport. With high latency storage, backup and restore performance may improve after increasing NFC AIO buffer size. If servers are capable of high concurrency, backup and restore throughput may improve with more NFC AIO buffers. The defaults are buffer size 64K (64KB * 1) and buffer count 4. The maximum buffer size is 2MB (64KB * 32) and the maximum buffer count is 16, as below.

```
vixDiskLib.nfcAio.Session.BufSizeIn64KB=32

vixDiskLib.nfcAio.Session.BufCount=16
```

Memory consumption in the NFC server increases with larger NFC AIO buffer sizes and buffer counts. The memory for each session can be calculated by `BufSizeIn64KB` * `BufCount`. If the value (`BufSizeIn64KB` * `BufCount`) is over 16MB, the disk open operation will fail. In testing, InitEx configuration file settings of buffer size 1MB (64KB * 16) and buffer count 4 performed best, but it depends on hardware setup.

In vSphere 6.7 and later, VDDK splits read and write buffers into 64KB chunks. Changing the buffer size on the VDDK side does not lead to different memory consumption results on the NFC server side.

In vSphere 6.5 and earlier, the larger the buffer size on the VDDK side, the more memory was consumed on the NFC server side. With buffer size set to 1MB, VMware recommended backing up no more than 20 disks in parallel on an ESXi host. For a 2MB I/O buffer, no more than 10 disks, and so on.

Session limits and vCenter session reuse. In vSphere 6.5 and later, programs can reuse a vCenter Server session to avoid session overflow. For details see "Reuse a vCenter Server Session" in chapter 4.

Network bandwidth considerations: VMware suggests that NBD backups should be done on a network with bandwidth of 10 Gbps or higher. Operations such as VM cloning or offline migration will also consume memory in the NFC server. Users must try to arrange their backup window to avoid conflict.

Log analysis for performance issues: The VDDK sample code can be run to assist with I/O performance analysis. In the configuration file, set the NFC log level to its highest value `vixDiskLib.nfc.LogLevel=4`. There is no need to set log level in the server for NFC asynchronous I/O. Then run sample code and investigate `vddk.log` and the `vpxa log` to assess performance.

General Backup and Restore

With SSL certificate checking in vSphere 5.1 and after, DNS services must be configured in the backup proxy, otherwise SSL_Verify will fail with the “no host found” error.

For incremental backup of virtual disk, always enable changed block tracking (CBT) before the first snapshot. When doing full restores of virtual disk, deactivate CBT for the duration of the restore. File-based restores affect change tracking, but disabling CBT is optional for partial restore (file level restore), except with SAN transport. CBT must be deactivated for SAN writes because of thin-provisioning and clear-lazy-zero operations.

Backup software should ignore independent disks (those not capable of snapshots). These virtual disks are unsuitable for backup. They throw an error if a snapshot is attempted on them.

When using VMware Tools debug logging with quiesced snapshots, do not log `vmtoolsd.data` to a local file on the VM, such as `C:\Temp\vmtoolsd.log`. Instead set `vmtoolsd.handler=vmx` to use the tools service.

To back up thick disk, the proxy's datastore must have at least as much free space as the maximum configured disk size for the backed-up virtual machine. Thick disk takes up all its allocated size in the datastore. To save space, you can choose thin-provisioned disk, which consumes only the space actually containing data.

If you do a full backup of lazy-zeroed thick disk with CBT deactivated, the software reads all sectors, converting data in empty (lazy-zero) sectors to actual zeros. Upon restore, this full backup data will produce eager-zeroed thick disk. This is one reason why VMware recommends enabling CBT before the first snapshot.

With CBT enabled for backups on an NFS datastore, thin-provisioned virtual disk may be turned thick upon restore, unless the NFS server supports **`lseek(...SEEK_DATA)`**, **`ioctl(...FS_IOC_FIEMAP)`**, or equivalent function.

Do not make verbatim copies of configuration files, which can change. For example, entries in the `.vmx` file point to the snapshot, not the base disk. The `.vmx` file contains virtual-machine specific information about current disks, and attempting to restore this information could fail. Instead use **`PropertyCollector`** and keep a record of the **`ConfigInfo`** structure.

Best Practice for Restore with Pre-existing Snapshot

When a backed-up VM had snapshots, disaster recovery software can restore a replacement VM with its pre-existing snapshots in place. However for partial file recovery, or to recover a VM back to a point in time, it is difficult to determine how to restore a VM that has an existing snapshot. Should parts of the snapshot be restored to a previous state (if applicable), or only the VM by ignoring the snapshot?

Certain backup-restore applications do not handle this situation well; snapshot contention causes a `GenericVmConfigFault`. One good solution is for customers to delete any pre-existing snapshots before recovery, especially if not needed. This is mandatory for SAN transport recovery, anyway. Restore applications could refuse to recover a VM until any pre-existing snapshot is deleted. If not, a customer work-around is to recover the VM to a different location, avoiding the issue, then retain the recovered VM and delete the VM with pre-existing snapshot.

Backup and Restore of Thin-Provisioned Disk

Thin-provisioned virtual disk is created on first write. So the first-time write to thin-provisioned disk involves extra overhead compared to thick disk, whether using NBDSSL or HotAdd. This is due to block allocation overhead, not VDDK advanced transports. However once thin disk has been created, performance is similar to thick disk, as discussed in the *`Performance Study of VMware vStorage Thin Provisioning`*.

When applications perform random I/O or write to previously unallocated areas of thin-provisioned disk, subsequent backups can be larger than expected, even with CBT enabled. In some cases, disk defragmentation might help reduce the size of backups.

About Changed Block Tracking

`VixDiskLib_QueryAllocatedBlocks` returns the sectors of a virtual disk that are in use. `QueryChangedDiskAreas` returns the sectors that changed since the time of a specified `changeId`.

The current implementation depends on VMFS properties, similar to properties that SAN transport mode uses to locate data on a SCSI LUN. Both rely on unallocated areas (file holes) in virtual disk, and the **`LazyZero`** designation for VMFS blocks. Thus, changed block tracking yields best results on VMFS. On other storage types, it could fail, or returns a single extent covering the entire disk.

You should enable changed block tracking in the order recommended by Enable Changed Block Tracking. At first call **`VixDiskLib_QueryAllocatedBlocks`** to get allocated areas of virtual disk, and back them up. Subsequently call `QueryChangedDiskAreas` to get changed areas, and back them up if they were allocated. If you call **`QueryChangedDiskAreas`** after a snapshot but before you enable changed block tracking, it also returns unallocated areas of virtual disk. With thin-provisioned virtual disk this could be a large amount of zero data.

The guest operating system has no visibility of changed block tracking. Once a virtual machine has written to a block on virtual disk, the block is considered in use. The information required for tracking is computed when changed block tracking is enabled, and the `.ctk` file is pre-filled with allocated blocks. The mechanism cannot report changes made to virtual disk before changed block tracking was enabled.

Configure SCSI Controllers in Proxy VM

When a backed-up VM has multiple SCSI devices, use of a proxy VM may result in disordered SCSI controllers. VDDK 8.0 fixed this issue for Linux proxies, but here is a customer work-around for Windows.

When the backup proxy must add multiple SCSI controllers to back up a VM with multiple virtual disks, HotAdd transport could fail to recognize some disks correctly. Error messages might indicate "cannot use mode HotAdd to access" or "unable to locate appropriate transport mode" or "mount failed" or "cannot open disk" or "read error: Error 7 (a file access error occurred on the host or guest operating system)" – causing backup to fail.

These errors result when SCSI controller order in the proxy VM does not correspond one-to-one with SCSI initiator order in the backed-up VM. For example, SCSI controller 3 matches with SCSI initiator 1, throwing off subsequent numbering. Rather, SCSI initiator 0 in the guest VM must correspond to SCSI controller 0 in the proxy VM, initiator 1 in the guest to controller 1 in the proxy, initiator 2 in the guest to controller 2 in the proxy, and initiator 3 in the guest to controller 3 in the proxy.

> **NOTE**
> VMware strongly recommends use of paravirtual SCSI controllers (pvSCSI) as SCSI controllers in virtual machines. VMware also recommends limiting guest VMs to no more than two SCSI controllers, if possible.

Workaround for Disordered SCSI Controllers

When the issue occurs customers should perform this procedure on a proxy VM after every reboot (power off and on) to check for mismatched SCSI controller numbering. The KB article https://kb.vmware.com/kb/2141281 was written before the issue was fixed for Linux.

Prerequisites:

- The OS disk(s) must be on SCSI controller 0 of the proxy VM.
- Customers must apply this workaround before deploying any backup software. If backup jobs are running in the proxy VM, customers should reschedule those jobs on another proxy VM to avoid any issues.
- Be familiar with topics "Add a Hard Disk to a Virtual Machine" and "Add a SCSI Controller to a Virtual Machine" in the *`vSphere Virtual Machine Administration`* manual on http://docs.vmware.com. Also see "Add a Paravirtualized SCSI Adapter" for details about pvSCSI.

Verify SCSI controller ordering:

Let's assume the proxy VM has four SCSI controllers and four SCSI host initiators. By adding disks to each controller, we can determine how SCSI controllers are matched to SCSI initiators.

1. Confirm four SCSI host initiators of type `vmw_pvscsi` (numbers are not necessarily 1, 2, 3).
   ```
   # lsscsi -H

   [0] vmw_pvscsi

   [1] vmw_pvscsi

   [2] vmw_pvscsi

   [3] vmw_pvscsi
   ```
2. Confirm disks are only on host0 with no disks on hosts 1, 2, 3.
   ```
   # lsscsi -d

   [0:0:0:0] disk VMware Virtual disk 2.0 /dev/sda [8:0]
   ```
3. Using the vSphere Client, add a dummy disk to SCSI controller 1.
   ```
   Disk Size: 1G

   Virtual Device Note: SCSI controller 1 SCSI (1:0) New Hard disk
   ```
4. Confirm that the disk was added on SCSI host initiator 1. In this example, host 1 is correctly matched to controller 1.
   ```
   # lsscsi -d

   [0:0:0:0] disk VMware Virtual disk 2.0 /dev/sda [8:0]

   [1:0:0:0] disk VMware Virtual disk 2.0 /dev/sdb [8:16]
   ```
5. Remove the disk added from SCSI controller 1.
6. Continue with similar steps for SCSI controllers 2 and 3.

If a non-correspondence is found at any step, continue by reconfiguring the SCSI controllers.

Reconfigure SCSI controllers:

1. Power on the proxy VM, if powered off. Verify backup software is not running.
2. Wait until the OS boots up completely to the login screen.
3. Confirm there are no disks on SCSI controllers 1, 2, or 3.
4. Remove SCSI controller 1, SCSI controller 2, and SCSI controller 3.
   Do not add multiple SCSI controllers in a single step.
5. Add back SCSI controller 1.
6. Add back SCSI controller 2.
7. Add back SCSI controller 3.

Encrypted VM Backup and Restore

Starting with vSphere 6.5, customers can take advantage of virtual machine encryption. This section describes how to design backup and restore to handle VM encryption.

Encryption protects sensitive parts of a VM and some or all of its virtual disks. The vCenter Server retrieves keys from a key management server and pushes them to ESXi hosts, which use the keys to encrypt virtual disk data, NVRAM, portions of the VMX file, and so forth. See the *vSphere Security* manual for details.

To back up encrypted virtual disks, VDDK obtains the encryption keys and decrypts virtual disk data before copying to backup media, so virtual disk data on backup media are in the clear (not encrypted). VMware recommends that backup agents use their own encryption to protect data on backup media.

Backup of encrypted disks is supported with NBDSSL and HotAdd transport modes, but not supported with SAN transport. To back up encrypted virtual disks using HotAdd, the backup proxy VM must have been encrypted as well. Both NBDSSL backup and HotAdd backup require **Cryptographer.Access** permission. HotAdd backup requires **Cryptographer.AddDisk** also.

Application quiesced snapshots are not supported for encrypted VMs. Use file-system quiescing instead. The snapshot manifest file will contain only a dummy `backup.xml` file.

Encrypted Virtual Disks

When customers restore virtual disks that were encrypted at backup time, they likely want them to remain encrypted after restore. If a disk was encrypted when it was backed up, the backup agent can remember the storage policy and apply it to the restored disk. If that storage policy no longer exists in the system, the backup agent could ask the administrator what policy to apply for encrypted disks, or use the sample VM encryption storage policy. Optionally the backup agent can remember a disk's key ID and provide it at restore time, which will result in the restored disk using the same key as the original disk. If not specified, vCenter Server gets a new key from the key server.

Recommendations for VM Home

When a VM is encrypted, its VMX configuration file contains portions that are encrypted, and the entire NVRAM file is encrypted. To completely recover a VM from backups, the NVRAM file must be restored, and two additional properties from the VirtualMachine object's configuration must be restored: (1) the encryption key identifier, found in the VirtualMachine `ConfigInfo.keyId` property, and (2) an encrypted blob containing VM secrets, called the `encryption.bundle`, found in the VirtualMachine `ConfigInfo.extraConfig` list.

VMware recommends against directly copying the entire VMX configuration file, but instead using the PropertyCollector to keep a record of configuration structures as documented earlier in this chapter. When a VM is saved, the backup agent should include the `ConfigInfo.keyId`, the `encryption.bundle` from `ConfigInfo.extraConfig`, and the current storage policy.

Later when the backup is restored, these values should be provided in the new VirtualMachine `ConfigSpec`. This will ensure that the recovered VM metadata files are protected with the same key as the original VM. If a VM is restored to a vCenter Server with key server access, the VM will boot. However if the vCenter Server lacks key server access, the VM will not power on after restore, because vCenter Server cannot push encryption keys to its ESXi host.

The NVRAM file can be handled as it was in previous releases, using HTTP download and upload, but without saving the additional ConfigInfo properties described above, the NVRAM will be unusable. This is because it is encrypted with the key that is found in the `encryption.bundle`, which is in turn sealed with the key identified by `ConfigInfo.keyId`. The recovered VM must be created in a vCenter Server that has access to the same key server as the original, or a replicated key server instance with the same cluster name.

What If Something Fails

If the `ConfigInfo.keyId` and `encryption.bundle` were not saved, it will be impossible to encrypt the recovered VM using its original encryption keys. However the VM may be encrypted with new keys from the key server. If the NVRAM file was saved, it is unusable without the original `ConfigInfo.keyId` and `encryption.bundle` information. If the NVRAM file was not saved, or unusable, a generic one can be used instead. After NVRAM is lost and regenerated, UEFI enabled VMs may require Secure Boot to be reconfigured, or the boot disk to be reconfigured.

If the vCenter Server has lost access to the original keys from the key server, then a restore operation may remove the `encryption.bundle` from `ConfigInfo.extraConfig` and specify only a VM encryption storage policy for the VM and its disks. Again this will cause any data present in the NVRAM to be unusable.

Backup and Restore With vTPM

Trusted platform module (TPM) is the standard for a dedicated microchip that can store sensitive data, perform cryptographic tasks, and ensure platform integrity by establishing a chain of trust for software loaded onto a machine. It assures integrity by calculating a message digest for each software component that gets loaded, storing the message digest in platform configuration register.

Enabling vTPM in a Virtual Machine

Virtual TPM (vTPM) is a software implementation of TPM provided in virtual hardware version 14. In other words, vSphere 6.7 offers vTPM for newly created or upgraded VMs. Because vTPM is encrypted, encryption services must be present on the network. Backup and restore of a vTPM enabled VM is similar to backup and restore of an encrypted VM, with these additional requirements.

- Each involved vCenter Server must be configured with the same key management server (KMS).
- Before adding the vTPM device to a VM, the `ConfigInfo.firmware` type must be set to `efi`, not `bios`. When you add a VM with encryption storage policy, vSphere encrypts the VM Home including vTPM.
- To preserve vTPM in a restored VM, the `ConfigInfo.keyId`, `encryption.bundle`, NVRAM file, and vTPM device of the source VM must be saved at backup time, for later restore. Saving an NVRAM file requires use of the HTTP file service.

Backup with vTPM

To back up a vTPM enabled VM, follow these steps, as in the sample code below.

1. Back up the `keyId` and `encryption.bundle` of the source VM from `configInfo`.
2. Back up the vTPM device of the source VM from `configInfo`.
3. Back up property `firmware` of the source VM from `configInfo`.

```
// get source VM config

VirtualMachineConfigInfo sourceVmConfigInfo = ... ;

// save keyId

CryptoKeyId keyId = sourceVmConfigInfo.getKeyId();

// save encryption.bundle, which is in extraConfig

List<OptionValue> extraCfg = sourceVmConfigInfo .getExtraConfig();

// save firmware

String firmware = sourceVmConfigInfo.getFirmware();

// save vTPM device

VirtualDevice vtpmDevice = null;

for (VirtualDevice virtualDevice : sourceVmConfigInfo.getHardware().getDevice()) {

   if (virtualDevice instanceof VirtualTPM) {

      vtpmDevice = virtualDevice;

   }

   // save other devices
```

```
   // ...
}
// save nvram file
byte[] nvramByteAry = vsphereFileServiceClient.download(sourceVmNvramFilePath);
```

Restoring With vTPM

To restore a vTPM enabled VM, follow these steps, as in the sample code below.

1. Configure a VM with the same `keyId` and `encryption.bundle` as source (requires same KMS).
2. Make sure an encryption storage policy exists and is assigned to the VM. See "Create an Encryption Storage Policy" in the *vSphere Web Services SDK Programming Guide*.
3. Configure this VM with the same firmware property and vTPM device as the source VM.
4. Restore NVRAM using HTTP service. Again, see section "HTTP Access to vSphere Server Files" in the *vSphere Web Services SDK Programming Guide*.

```
// create configSpec for VM to be created
VirtualMachineConfigSpec configSpec = new VirtualMachineConfigSpec() ;
// set keyId
CryptoSpecEncrypt cryptoSpec = new CryptoSpecEncrypt();
cryptoSpec.setCryptoKeyId(keyId);
configSpec.setCrypto(cryptoSpec);
// set encryption.bundle
configSpec.setExtraConfig(extraCfg);
//
// set PbmProfile for encryption
// For complete code, see Example: Java program to set storage policy for encryption.
// public class CreateVMEncryptionProfile extends ConnectedServiceBase {
// private PbmServiceInstanceContent spbmsc;
// private String profileName;
// ...
// for (PbmCapabilityVendorResourceTypeInfo vendor : vendorInfo)
//    for (PbmCapabilityVendorNamespaceInfo vnsi : vendor .getVendorNamespaceInfo())
//       if (vnsi.getNamespaceInfo().getNamespace().equals("vmwarevmcrypt")) {
//          encryptionCapable = true;
//          break;
//       }
```

```
// ...
// set firmware
configSpec.setFirmware(firmware);
// set vTPM device
VirtualDeviceConfigSpec vtpmDeviceConfig = new VirtualDeviceConfigSpec();
vtpmDeviceConfig.setOperation(VirtualDeviceConfigSpecOperation.ADD);
vtpmDeviceConfig.setFileOperation(null);
vtpmDeviceConfig.setDevice(vtpmDevice);
configSpec.getDeviceChange().add(vtpmDeviceConfig);
// set other properties and then create restore VM
// ...
// upload nvram
vsphereFileServiceClient.upload(restoreVmNvramFilePath, nvramByteAry
```

Windows Backup Implementations

The following sections discuss issues when backing up Windows virtual machines.

Deactivate Automount in Windows Proxy

When using HotAdd transport from a Windows proxy, it can make unwanted changes to HotAdded volumes. To prevent this, backup vendors and customers should deactivate Windows automount on the backup proxy.

To deactivate Windows automount:

1. Start DiskPart.
   ```
   C:\> diskpart
   ```
2. Deactivate automounting.
   ```
   DISKPART> automount disable
   ```
3. Prevent any previously mounted volumes from being re-mounted the next time.
   ```
   DISKPART> automount scrub
   ```
4. Exit DiskPart.
   ```
   DISKPART> exit
   ```

Security and Remote Desktop

For security reasons, remote desktop protocol (RDP) should be deactivated on Windows proxies.

Working with Microsoft Shadow Copy

Microsoft Shadow Copy, also called Volume Snapshot Service (VSS), is a Windows Server data backup feature for creating consistent point-in-time copies of data (called shadow copies).

The type of quiescing used varies depending on the operating system of the backed-up virtual machine, as shown in Table : Driver Type and Quiescing Mechanisms Used According to Guest Operating Systems. ESXi 4.1 added support for Windows 2008 guests using application level quiescing.

Table 111: Driver Type and Quiescing Mechanisms Used According to Guest Operating Systems

Guest Operating System	Driver Type Used	Quiescing Type Used
Windows XP 32-bit Windows 2000 32-bit	Sync Driver	File-system consistent quiescing.
Windows Vista 32- or 64-bit Windows 7 32- or 64-bit	VMware VSS component	File-system consistent quiescing.
Windows 2003 32- or 64-bit	VMware VSS component	Application-consistent quiescing.
Windows 2008 32- or 64-bit Windows Server 2008 R2	VMware VSS component	Application-consistent quiescing. For this to be available, several conditions must be met: • Virtual machine must be running on ESXi 4.1 or later. • The UUID attribute must be enabled. It is enabled by default for virtual machines created on 4.1 or later. For details see Enable Virtual Machine Application Consistent Quiescing. • The virtual machine must use SCSI disks only and have as many free SCSI slots as the number of disks. Application-consistent quiescing is not supported for virtual machines with IDE disks. • The Windows VM must not use dynamic disks. If the VM has dynamic disks, the snapshot will be file-system consistent.
Windows Server 2012	VMware VSS component	Same as for Windows 2008.
Windows Server 2016	VMware VSS component	Same as for Windows 2008. VMware Tools 10.1.0 or later is required for quiescing support. Application quiesced snapshots are not supported for encrypted VMs, nor for virtual disks managed by Storage Spaces. Use file-system quiescing instead.
Windows Server 2019	VMware VSS component	Same as for Windows 2016.
Windows Server 2022	VMware VSS component	Same as for Windows 2016.
Other guest operating system	Not applicable	Crash-consistent quiescing.

File-system consistent quiescing prevents file systems from becoming corrupted, for example, journaled file systems are allowed to write out pending transactions. Crash-consistent quiescing is the ability to restore an application as if it suddenly crashed and lost all stateful information. This involves minimal effort because only data already written to disk is guaranteed safe. Application-consistent quiescing is the ability to restore stateful information as well.

Restore must be done using the backup application's guest agent. The vSphere APIs for Data Protection provide no host agent support for this. Applications authenticating with SSPI might not work right because HTTP access will demand a user name and password, unless the session was recently authenticated.

When performing VSS quiescing while creating the snapshot of a Windows virtual machine, VMware Tools generate a `vss-manifest.zip` file containing the backup components document (BCD) and writer manifests. The host agent stores this manifest file in the **`snapshotDir`** of the virtual machine. Backup applications should get the `vss-manifest.zip` file so they can save it to backup media. There are several ways to get this file:

- Using the datastore access HTTPS protocol, for example by browsing to https://<server-or-host>/folder/ and continuing downward to the snapshot directory until you find the `vss-manifest.zip` file.
- By calling the **`CopyDatastoreFile_Task`** method in the vSphere API. This method accepts the URL formulated above for HTTPS, or a datastore path. (**`CopyVirtualDisk_Task`** is for VMDK files).
- With the **`vifs`** command in the vMA or vCLI.
- With the **`Copy-DatastoreItem`** cmdlet in the PowerCLI (requires PowerShell and VMware snap-in).

Windows 2008 application level quiescing is performed using a hardware snapshot provider. After quiescing the virtual machine, the hardware snapshot provider creates two redo logs per disk: one for the live virtual machine writes and another for the VSS and writers in the guest to modify the disks after the snapshot operation as part of the quiescing operations.

The snapshot configuration information reports this second redo log as part of the snapshot. This redo log represented the quiesced state of all the applications in the guest. This redo log must be opened for backup with VDDK 1.2 or later. The older VDDK 1.1 software cannot open the second redo log for backup.

Application consistent quiescing of Windows 2008 virtual machines is only available when those virtual machines are created in vSphere 4.1 or later. Virtual machines created in vSphere 4.0 can be updated to enable application consistent quiescing by modifying a virtual machine's enableUUID attribute.

For information about VSS, see the Microsoft TechNet article, *`How Volume Shadow Copy Service Works`*. For information about Security Support Provider Interface (SSPI), see the MSDN Web site.

Enable Virtual Machine Application Consistent Quiescing

To enable virtual machine application consistent quiescing on Windows 2008 or later:

1. Start the vSphere Client, and log in to a vCenter Server.
2. Select **Virtual Machines and Templates** and click the **Virtual Machines** tab.
3. Right-click the Windows 2008 virtual machine for which you are enabling the disk UUID attribute, and select **Power > Power Off**. Wait for the virtual machine to power off.
4. Right-click the virtual machine, and click **Edit Settings**.
5. Click the **Options** tab, and select the **General** entry in the settings column.
6. Click **Configuration Parameters...** The Configuration Parameters window appears.
7. Click **Add Row**.
8. In the Name column, enter **disk.EnableUUID**. In the Value column, enter **TRUE**.
9. Click **OK** and click **Save**.
10. Power on the virtual machine.

    Application consistent quiescing is available for this virtual machine after the UUID property is enabled.

Application-Consistent Backup and Restore

Here is the approximate procedure for software to performs application-consistent backup and restore:

1. Call **`CreateSnapshotEx_Task`** with quiesce set appropriately for VSS (see below).
2. Open the leaf node of the disk with VDDK and read both the base VMDK and the snapshot at once.

3. Delete the snapshots created in the first step.
4. During restore, create a new virtual machine.
5. Write the VMDK to disk with VDDK. It should have both base and quiesced information.

   During backup, if the snapshot was created with quiesce flag set to true, and all the quiescing conditions are met, so the snapshot is created involving VSS and the snapshot disks represent application consistent state of the guest OS. You should be able to confirm this by downloading the VSS **`manifest zip`** file, unzipping it to check if it has just the backup component document (in which case file system quiescing was performed) or also writer manifests (in which case application quiescing was performed).

   Quiescing involves the VSS mechanism designed by Microsoft. So, regarding VSS backup-restore verification, refer to the VSS documentation provided by Microsoft. VMware helps by providing a `vss-manifest.zip` file that contains Backup/Writers Components details. This is generated by the VSS mechanism after backup. By cross verifying these backup/writers components details according to Microsoft VSS documentation, you can verify if a particular application-consistent quiescing was completed successfully or not.

   VMware Tools is responsible for initiating the VSS snapshot process as the VSS requester. Users send a request to **`hostd`** for a quiesced snapshot of the virtual machine. The request goes from **`hostd`** to the VMware Tools for a VSS snapshot. Once the VSS snapshot is completed (with success or error) it communicates back to the **`hostd`** process. The VSS snapshot is created with the **`vss-manifest`** file, or without this file in the error case.

   The VSS requester sets up the overall configuration for the backup operation, including whether the snapshot should be performed in component mode or not, whether to take a snapshot with a bootable system state, and whether the snapshot should be for a full copy or differential backup. If application-consistent quiescing is performed, then all writers and all components are involved.

   VMware Tools initiates VSS quiescing using **`VSS_CTX_BACKUP`** context for application quiescing capable guests with backup state set to select components, backup bootable system state with backup type **`VSS_BT_COPY`** and no partial file support and **`VSS_CTX_FILE_SHARE_BACKUP`** for file system quiescing capable guests. Currently there is no way to control any of these parameters.

New VSS Support Added in vSphere 6.5

The vSphere 6.5 release included additional volume shadow-copy service (VSS) configurations and features. To support these new configuration and for more granular control over Windows guest OS quiescing, the function **`CreateSnapshotEx_Task`** was added to the 6.5 vSphere API, superseding **`CreateSnapshot_Task`**. **`CreateSnapshotEx_Task`** takes a **`quiesceSpec`** parameter, of type **`VirtualMachineGuestQuiesceSpec`** or **`VirtualMachineWindowsQuiesceSpec`**. The latter type can specify several important fields such as:

- **`vssBackupType`** – **`VSS_BT_COPY`** was previously used as the default for **`CreateSnapshot_Task`** but now **`VSS_BT_FULL`**, **`VSS_BT_INCREMENTAL`**, **`VSS_BT_DIFFERENTIAL`**, and **`VSS_BT_LOG`** are available also. Log truncation may be triggered according to application settings.
- **`vssBackupContext`** – this was introduced to enforce application (context **`VSS_CTX_BACKUP`**) quiescing or file system (context **`VSS_CTX_FILE_SHARE_BACKUP`**) quiescing.
- The timeout (default 15 minutes) for quiescing virtual machines can now be configured anywhere from five minutes to four hours.

`CreateSnapshotEx_Task` requires VMware Tools 10.1.0 or higher installed on the backed-up virtual machine.

The VMware VSS Implementation

As of Windows Server 2008, disk UUIDs must be enabled for VSS quiesced snapshots. Disk UUIDs might not be enabled if a virtual machine was upgraded from virtual hardware version 4.

VMware VSS does not support virtual machines with IDE disks, nor does it support virtual machines with an insufficient number of free SCSI slots.

Before vSphere 5.1, reverting to a writable snapshot sometimes left orphaned virtual disks that the system never removed. In the vSphere 5.1 release, writable snapshots are correctly accounted for as sibling snapshots. This permits cleaner management, because the disk chain matches the snapshot hierarchy, and it avoids orphaned disks. Linux backup software takes a read-only snapshot so is not affected. On Windows, VSS backup software may create two snapshots, one made writable by calling **`CreateSnapshotEx_Task`** with **`quiesceSpec`** flag set.

To add support for granular application control, specify:

- whether pre-freeze and post-thaw scripts get invoked
- whether quiescing gets invoked
- VSS snapshot context (application, file system quiescing, and so forth)
- VSS backup context (full, differential, incremental)
- writers/components to be involved during quiescing
- whether to fail quiescing or continue if one of the writers fails to quiesce
- retry count

A VSS quiesced snapshot reports as **`VSS_BT_COPY`** to VSS, hence no log truncation. The VSS manifest can be downloaded with HTTP. By default, all VSS writers are involved, but a mechanism for excluding writers exists; see the VMware KB article 1031200. For help troubleshooting, see KB article 1007696.

Linux Backup Implementation

On Linux virtual machines, VMware supports only crash-consistent backups, however application-consistent backups are possible through the use of customer pre-freeze and post-thaw scripts.

Two methods are attempted sequentially when a quiesced snapshot is requested of a Linux virtual machine:

1. Using the **`ioctl`**(2) mechanisms **`FIFREEZE`** and **`FITHAW`** built into the Linux kernel.

   This option is available in only with kernel versions newer than 2.6.32 on Linux virtual machines running VMware Tools 5.x or higher (vSphere 5.0 and later).

2. Using the **`sync`**(2) system call.

   The first method allows you define custom scripts that get called by VMware Tools before and after quiesce. Here are two commands that can be called from pre-freeze and post-freeze scripts:

   ```
   # -f means freeze

        fsfreeze -f /


   # -u means unfreeze

        fsfreeze -u /
   ```

   The mount point used in the scripts (root in the examples above) must be mounted with the **`noatime`** option, meaning do not update **`inode`** access times on the file system, for freeze and unfreeze to work.

Backing Up vApps in vCloud Director

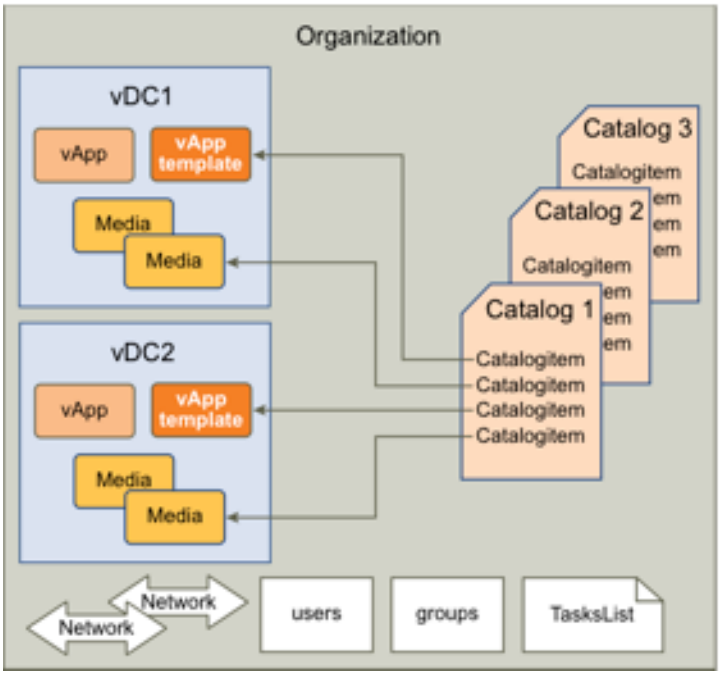
This chapter introduces developers to concepts and procedures for creating backup and restore solutions for vCloud Director.

Tenant vApps in VMware Cloud

The vApp is a management construct that encapsulates one or more virtual machines running in the vSphere environment. The tenant vApp is a higher-level construct that allows vCloud Director to manage vApps and virtual machines running in a multi-tenant datacenter, or in a multi-tenant cloud, based on vSphere.

Figure: vCloud API Object Taxonomy shows the objects within a single organization that you can access with the vCloud API.

Figure 176: vCloud API Object Taxonomy



Multi-tenant and self-service capabilities of vCloud Director provide multiple levels of protection for a vApp. A service provider can offer vApp protection at the system level, the tenant level, or the end-user level, managed by the system administrator, Organization administrator, and end user, respectively. This chapter focuses on the protection provided at the system level, where service providers can employ backup solutions from vendors of data protection software.

This chapter describes how to design software to back up and restore the vApps in a vCloud. To back up or restore a vApp, you need to deal with both the vCloud configuration and the virtual machines that belong to the vApp. In vSphere, a virtual machine is represented by configuration files and virtual disk files.

Prerequisites

You should be familiar with programming concepts and techniques. You should also be familiar with vCloud, vCloud API, vCloud SDK for .NET, and vSphere concepts. VMware also provides the vCloud SDK for Java and the vCloud SDK for PHP, but this chapter focuses on .NET for the backup and restore examples.

VMware recommends that you design backup and restore software for the vCloud environment using the following APIs:

Table 112: APIs Used To Back Up vApps

Product	API	Data
vCloud Director	vCloud API or vCloud SDK wrapper	vApp metadata

Table continued on next page

Continued from previous page

Product	API	Data
vSphere	WS API	virtual machine configuration
VDDK	VixDiskLib API or VixMntapi	virtual disk contents

You use the vCloud API or SDK to identify vApp targets for backup and restore operations. The vApp metadata identifies the virtual machines that constitute the vApp. You use the WS API to back up and restore virtual machine configurations. You use the VDDK API to back up and restore virtual disk files.

> **NOTE**
> This chapter uses the term “metadata” in a general sense to mean all the vApp configuration data, in addition to user-defined data that the vCloud REST API serializes in the **`<Metadata>`** element.

You should be familiar with the use of the WS API and the VDDK API for backup and restore of individual virtual machines.

Other Information

This chapter does not, in general, duplicate information available in other documents. In particular, this chapter does not provide details about any storage or data protection API that you need to use for backing up and restoring virtual machines in vSphere. You should consult separate reference documentation for details about specific API calls.

This chapter emphasizes the use of the vCloud API and SDK for the purpose of managing metadata of the virtual machines and related artifacts in vCloud Director. The vCloud SDK for .NET translates your C# code into REST operations using the vCloud API.

To learn about VMware vCloud and vSphere concepts and usage, refer to the vCloud Director documentation available from the VMware documentation site, https://docs.vmware.com. You can also visit one of the VMware SDK community forums.

Conceptual Overview

This section summarizes the backup and restore processes for vApps managed by vCloud Director. It explains how to use VMware APIs to collect the metadata needed to control backup and restore operations. The actual backup and restore operations are performed using the VMware vStorage APIs for Data Protection (VADP).

VMware vCloud Director uses one or more vCenter servers to manage virtualized resources. At the same time, it manages the vCloud feature of multi-tenancy by maintaining metadata related to various tenant artifacts such as vApp, users, networks, storage, and so on.

When a system administrator chooses to back up a vApp, certain vApp metadata must be retrieved from vCloud Director. The metadata includes general information about the vApp (name, description, virtual machine descriptions), networking information (organization network connectivity, external network connectivity), user information, lease, and quota. This information becomes particularly important when restoring the vApp, in addition to the names of virtual disk files and `.vmx` files typically retrieved from vSphere using the VADP.

The Backup Process

The backup process requires the backup/restore software to collect and store information both from vCloud Director and from vSphere. This process assumes that you use vCloud Director system administrator credentials to connect to vCloud Director. System administrator credentials allow the software to access vApps belonging to any Organization, and to access all the necessary information about a vApp and associated vCloud constructs.

A vApp in vCloud Director can comprise one or more virtual machines. When you work with a single vApp in vCloud Director, you might be working with a number of virtual machines in vSphere.
To back up a vApp or set of vApps:

1. Connect to vCloud Director and access the organization where vApp (or vApps) will be backed up.
2. When backing up a vApp for a given Organization or VDC in vCloud Director, access the vCloud Director inventory for a list of all desired vApps.
3. Select maintenance mode for each vApp to prevent updates during the backup process.
4. Collect all the metadata related to the vApp(s), including any user-defined metadata associated with any given vApp.
5. Use the vApp metadata to identify the virtual machines associated with each vApp.
6. Connect to vCenter Server as a user with sufficient permissions to access the virtual machines. Use the vSphere inventory to locate the virtual machine configuration and virtual disk files.
7. Use the VMware APIs for Data Protection to back up the vSphere virtual machine files:
   a) (optional) Save a snapshot of the virtual machine.
   b) Save the virtual machine configuration, using the WS API.
   c) Save the virtual disks using the VDDK API.
   d) (optional) Delete the virtual machine snapshot, if applicable.
8. Store the vApp metadata in an appropriate format along with the associated virtual machine files.
9. Deselect maintenance mode for each vApp.

The Restore Process

The restore process offers some options to the administrator.

When you restore a vApp, you can choose to overwrite an existing vApp. For instance, the restore software might need to overwrite a vApp with data corruption. You can also choose to restore a vApp that no longer exists, for instance, a vApp that was accidentally deleted.
You can choose whether to keep the same vApp name and other vApp attributes, or you can choose to change attributes during the restore process. If the attributes of the restored vApp no longer conform to the environment because of changes since the backup was taken, you can select new values for the non-conforming attributes.
You might want to restore an existing vApp to an earlier state, or you might want to replace it because it has become corrupted.
To restore an existing vApp:

1. Identify the child virtual machines of the vApp, using the metadata stored with the backup.
2. Connect to vCenter Server as a user with sufficient permissions to access the virtual machines and restore the virtual machines in the vSphere environment. This step restores the virtual disk files and virtual machine configuration. If you are overwriting an existing vApp, you generally restore the files to the same data store that vCloud Director currently uses for the vApp.
3. Connect to vCloud Director and authenticate as an administrator, which gives you backup and restore privileges.
4. Locate the corrupted vApp, using the ID retrieved from the metadata in the backup store.
5. Select maintenance mode for the vApp, to prevent changes while restoring metadata.
6. Edit vApp settings such as network, user privileges, lease, and quota as needed. Make sure to include any user-defined metadata from the backup store. If you restored a virtual machine to a different location from the original, you might need to adjust the vApp settings.
7. Deselect maintenance mode for the vApp.

You might want to restore a missing vApp because somebody deleted it, or as part of disaster recovery.
To restore a missing vApp:

1. Identify the child virtual machines of the vApp, using the metadata stored with the backup.
2. Connect to vCenter Server as a user with sufficient permissions to access the virtual machines and restore the virtual machines in the vSphere environment. This step restores the virtual disk files and virtual machine configuration.
3. Connect to vCloud Director and authenticate as an administrator, which gives you backup and restore privileges.
4. Compose a new vApp or import the virtual machine(s) into vCloud Director to create a new vApp with these characteristics:

a. It has the same name as the lost vApp.
b. It belongs to the same Organization as the lost vApp.
c. It obtains resources from the same provider VDC as the lost vApp.

5. Select maintenance mode for the vApp, to prevent changes while restoring metadata.
6. Edit vApp settings such as network, user privileges, lease, and quota as needed. Make sure to include any user-defined metadata from the backup store.
7. Deselect maintenance mode for the vApp.

NOTE
This is a simplified view of the restore process. The exact process you use will depend on the features provided by your software. For instance, if the datastore is full, the software could offer to migrate the vApp to a different datastore.

Use Cases Overview

The following sections give an overview of use cases related to the backup and restore processes.

Managing Credentials

Backup software needs vCloud Director access to manage vApps at the metadata level, and vCenter Server access to manage vApps at the virtual machine and virtual disk level. The backup software must collect and retain authentication credentials for both vCloud Director and vCenter Server.

For information about vCloud Director authentication, see Getting Access to vCloud Director. For information about vSphere authentication, see the *vSphere Web Services SDK Programming Guide*.

Finding a vApp

There are different ways to locate a vApp managed by vCloud Director. One way is to traverse the vCloud Director inventory. Another way is to use the query service.

Inventory Traversal

Using the vCloud Director inventory to locate a vApp requires navigating a hierarchy of containers based on organizational and resource divisions. The process is explained in Inventory Access.

Using the Query Service

The vCloud SDK for .NET also supports the query service of the vCloud API for finding vApps. Consult the sample programs in the SDK for more information about how to use the query service in the SDK.

Protecting Specified vApps

Backup systems typically identify vApps to be backed up in a given Organization based on their identity, using vApp attributes such as name and ID or user defined metadata. A set of vApps to be backed up can also be created based on their Organization (for example, all vApps in the Human Resources Organization), the VDC where they are deployed, and so forth.

In all these cases you must traverse the given Organization and its contents to locate and make a list of vApps.

Recovering an Older Version of a vApp

If a vApp has become corrupted, or if users need to revert to an older state of the vApp, the administrator can restore a version of the vApp from backup storage even when the vApp still exists in vCloud Director. The backup/restore application in these cases can access vCloud Director to get vApp identity information and metadata before restoring the backup copy.

The backup/restore application has a choice between overwriting the current vApp instance or deleting it and creating a new vApp. The choice to delete the vApp can be convenient when the vApp configuration has changed since the last backup, especially when a virtual machine has been added to or deleted from the vApp.

Recovering a Deleted vApp

When recovering a deleted vApp, the backup/restore application must identify the vApp from user input to locate the vApp metadata and virtual machine files on the backup storage medium. After the virtual machines have been restored using vSphere APIs, the vApp can be recomposed using the vCloud API. The backup software must first create a vApp from one of the virtual machines, then import the remaining virtual machines into the same vApp.

Recovering a Single Virtual Machine

The process of recovering a single virtual machine from the backup storage medium is a special case of recovering a deleted vApp. In the case of a deleted vApp, the backup software must re-create the vApp in vCloud Director, then import the remaining virtual machines. For a single lost virtual machine, the backup software must only import the one virtual machine into the existing vApp.

Backing Up vCloud Director

The vCloud SDK for .NET does not offer any special features for backing up or restoring the vCloud Director application and its data. Users should follow standard industry advice for protecting Tomcat applications and Oracle or SQL Server databases.

vCloud API Operations

The following sections describe commonly used vCloud API operations using vCloud SDK for .NET. The API descriptions in this chapter do not provide complete backup/restore implementation details, but focus instead on identifying a set of vCloud API methods that facilitate certain operations that use vCloud Director.

You should be familiar with vCloud Director and vCloud API concepts. Every resource in vCloud Director can be accessed using either its unique ID or HREF (the reference URL) in the vCloud API. The .NET SDK provides wrapper utility classes for commonly-used resources to make the programming easier.

The operations described in the following sections are:

- Getting Access to vCloud Director – Shows how to connect and authenticate with the vCloud API.
- Inventory Access – Shows how to retrieve data for different Organization types.
- Retrieving Catalog information – Shows how to retrieve Catalog entries for backup.
- Retrieving vApp Configuration – Shows how to list virtual machines and vApp configuration data.
- Preventing Updates to a vApp During Backup or Restore – Shows how to use maintenance mode to quiesce vApp configuration.
- Associating vCloud Resources with vSphere Entities – Shows how to get Managed Object References of virtual machines and storage resources from vCloud Director.
- Restoring vApps – Shows how to import virtual machines into vApps.

Getting Access to vCloud Director

The backup/restore software component must use system administrator privileges to connect to vCloud Director, so that it can access any Organization. The system administrator always logs into the System organization. When **`Administrator@System`** is used as the user name for the API, **`Administrator`** is the login name and **`System`** is the System Organization name.

Using system administrator privileges to connect to vCloud Director also allows the backup/restore software to access additional information relating a vApp to the corresponding resources in vSphere. This is described in Inventory Access.

The following example shows how to log in using C# with the vCloud SDK for .NET. After logging in, the code shows how to access Organization data.

```
// vCloud Director login code sample using Administrator@System/<password>

using com.vmware.vcloud.sdk;

using com.vmware.vcloud.api.rest.schema;


public static vCloudClient client = null;

client = new vCloudClient(vCloudURL, com.vmware.vcloud.sdk.constants.Version.V1_5);

client.Login(username, password);

// Get references to all Organizations:

Dictionary<string,ReferenceType> organizationsMap = client.GetOrgRefsByName();

// Get reference to a specific Organization:

string orgName = "Org1";

ReferenceType orgRef = client.GetOrgRefByName(orgName);

// Convert Organization reference to Organization object:

Organization org = Organization.GetOrganizationByReference(client, orgRef);
```

Inventory Access

In general, you locate a desired vApp for backup in the context of a given Organization and VDC. To locate a vApp that you want to back up, you first need a reference to its parent Organization.

You use the Organization reference to get the Organization object, which you use to get a list of references to the VDCs that belong to the Organization. You use a VDC reference to get a VDC object, which you then use to get a list of references to the vApps that belong to the Organization. You convert the desired vApp reference to a vApp object, which you use to list the virtual machines that belong to the vApp.

The following example shows how to get a reference to the user view of an Organization, and how to get a reference to the admin view of an Organization and a VDC.

```
// Get admin Org and vDC

public static vCloudClient client = null;

// Login

...
```

```
// Get admin view of Org

VcloudAdmin admin = client.GetVcloudAdmin();

string orgName = "Org1";

ReferenceType orgRef = admin.GetAdminOrgRefByName(orgName);

AdminOrganization adminOrg = Organization.AdminGetOrgByReference(client, orgRef);

// Get admin vDC

string vdcName = "VDC1";

ReferenceType vdcRef = adminOrg.GetAdminVdcRefByName(vdcName);

...

AdminVdc adminVdc = AdminVdc.GetAdminVdcByReference(client, vdcRef);
```

Admin Views

The admin view of resources such as Organization, VDC, and vApp provides extra information that is useful to users with administrative privileges. For example, in the case of a vApp, admin view provides information about vCenter and the virtual machines that belong to the vApp. The admin view provides information such as Managed Object References that vCenter uses for those entities. See Associating vCloud Resources with vSphere Entities for more information about getting vCenter Managed Object References.

To access admin views, you use a method of the client connection object to create an admin client proxy. The admin proxy has methods similar to those of the client connection object to get references to Organizations and other vCloud objects. However, the objects you get from the admin proxy have additional properties not present in user objects.

Admin Extensions

Similar to the admin views, you can use a different method of the client connection object to create an admin extension client proxy. You use the admin extension proxy to find provider VDC. A provider VDC includes one or more resource pools and allocates resources from those pools to the Org VDCs that it supports.

The following example shows how to get a Provider VDC.

```
// Login

…

// Get dictionary of Provider vDCs:

AdminExtension.VcloudAdminExtension adminExt = client.GetVcloudAdminExtension();

string pvdcName = "ProvVDC1";

Dictionary<string, ReferenceType> refs = adminExt.GetVMWProviderVdcRefsByName();

…

// Get reference for pvdcName -> pvdcRef

ReferenceType pvdcRef = refs[pvdcName];
```

```
VMWProviderVdc vmwPvdc = VMWProviderVdc.GetVMWProviderVdcByReference(client, pvdcRef);
```

Using the vCloud SDK for .NET allows you to access vCloud Director from a C# development environment. These examples show how to use .NET methods. The vCloud SDK for .NET simplifies access to the vCloud API. For more information about using the SDK, see the *vCloud SDK for .NET Developer's Guide*.

The vCloud API is REST-based. For more information about the vCloud API, see the *vCloud API Programming Guide*. The following example shows the REST API calls that accomplish the tasks shown in examples above, after logging in. REST API Calls To Get Provider VDC:

```
GET https://vCloud/api/admin

GET https://vCloud/api/admin/org/id

GET https://vCloud/api/admin/vdc/id

GET https://vCloud/api/admin/extension

GET https://vCloud/api/admin/extension/providervdc/id
```

In general, if you do not need admin views or provider views, you can use an Organization reference to get a VDC reference, and you can use the VDC reference to get a list of vApps belonging to the VDC. The following example shows how to list the hierarchy of Organizations, VDCs, and vApps known to vCloud Director. This example assumes you have already logged in to vCloud Director.

```
// List vApps in a VDC for a Given Organization

Dictionary<string, ReferenceType> organizationsMap = client.GetOrgRefsByName();

if (organizationsMap != null)

{

   foreach (string organizationName in organizationsMap.Keys)

   {

      ReferenceType organizationReference = organizationsMap[organizationName];

      Organization org = Organization.GetOrganizationByReference(client,
organizationReference);

      string OrgID = org.Resource.id;

      Console.WriteLine("Organization Name:" + organizationName);

      Console.WriteLine("Organization Id :" + OrgID);

   }

   foreach (ReferenceType orgRef in organizationsMap.Values)

   {

      Organization org = Organization.GetOrganizationByReference(client, orgRef);


      foreach (ReferenceType vdcRef in org.GetVdcRefs())

      {
```

```
         Vdc vdc = Vdc.GetVdcByReference(client, vdcRef);
         string vdcId = vdc.Resource.id;
         Console.WriteLine("Org vDC Id:" + vdcId);
         Console.WriteLine("Org vDC Name:" + vdc.Reference.name);
         foreach (ReferenceType vAppRef in Vdc.GetVdcByReference(client,
vdcRef).GetVappRefs())
         {
            Vapp vapp = Vapp.GetVappByReference(client, vAppRef);
            Console.WriteLine("vApp Id:" + vapp.Resource.id);
            Console.WriteLine("vApp Name:" + vapp.Resource.name);
            List<VM> vms = new List<VM>();
            try
            {
               vms = vapp.GetChildrenVms();
            }
            catch
            {
               // Handle exception here
            }
            foreach (VM vm in vms)
            {
               Console.WriteLine("VM Id : " + vm.Resource.id);
               Console.WriteLine("VM Name : " + vm.Resource.name);
            }
         }
      }
   }
}
```

The .NET SDK code in the following example translates to the API calls shown in the example below this one. REST API Calls To List vApps in a VDC for a Given Organization:

```
GET https://vCloud/api/admin
GET https://vCloud/api/admin/org/id
GET https://vCloud/api/admin/vdc/id
```

```
GET https://vCloud/api/admin/extension
```

```
GET https://vCloud/api/admin/extension/providervdc/id
```

You can use a provider VDC reference to enumerate its associated datastores, as shown in the following example, which assumes you have already logged in to vCloud Director. List Datastores:

```
/// <summary>
/// Returns list of Provider vDCs.
/// </summary>
/// <returns>ReferenceType</returns>
public static List<ReferenceType> GetProviderVdc()
{
   List<ReferenceType> vdcRefList = new List<ReferenceType>();
   foreach (ReferenceType vdcRef1 in
      client.GetVcloudAdminExtension().GetVMWProviderVdcRefsByName().Values)
   {
      vdcRefList.Add(vdcRef1);
   }
   return vdcRefList;
}

/// <summary>
/// Returns the list of DataStores
/// </summary>
/// <returns>ReferenceType</returns>
public static List<ReferenceType> GetDataStore()
{
   extension = client.GetVcloudAdminExtension();
   List<ReferenceType> vmDatastorelist = new List<ReferenceType>();
   foreach (ReferenceType datastoreRef in extension.GetVMWDatastoreRefs())
   {
      vmDatastorelist.Add(datastoreRef);
   }
   return vmDatastorelist;
```

```
}

// Get the datastores for the list of Provider vDCs.

foreach (ReferenceType providerVdcRef in GetProviderVdc())
{
   string providerVdcId = GetId(providerVdcRef.href);
   Console.WriteLine("Provider vDC Id:" + providerVdcId);
   Console.WriteLine("Provider vDC Name:" + providerVdcRef.name);
   foreach (string morefitem in
      VMWProviderVdc.GetResourcePoolsByMoref(client, providerVdcRef).Keys)
   {
      Console.WriteLine("Moref :" + morefitem);
   }
   foreach (VMWProviderVdcResourcePoolType VcResourcePool in
      VMWProviderVdc.GetResourcePoolsByMoref(client, providerVdcRef).Values)
   {
      string VcResourcePoolId =
GetId(VcResourcePool.ResourcePoolVimObjectRef.VimServerRef.href);
      Console.WriteLine("VcResourcePoolId :" + VcResourcePoolId);
   }
}
foreach (ReferenceType item in GetDataStore())
{
   string DatastoreId = GetId(item.href);
   Console.WriteLine("Data Store ID:" + DatastoreId);
   Console.WriteLine("DataStore:" + item.name);
}
```

Retrieving Catalog information

Catalogs on vCloud Director store vApp templates and ISO images as Catalog items. Backup solutions can be asked to back up the items in the Catalog for a given Organization. Catalogs can be shared or private. A user can choose to back up all items or only selected items in the given catalog. For this it is necessary to traverse the given Catalog in an Organization to access the contents and extract the various metadata associated with the vApp.

The following example shows inventory traversal to access the Catalog items in a given Organization, and assumes you have logged in to vCloud Director and obtained a map of Organizations, as in examples above.

```
// List catalogs and catalog items for a given organization:

Console.WriteLine();

if (organizationsMap != null && organizationsMap.Count > 0)

{

   foreach (string organizationName in organizationsMap.Keys)

   {

      ReferenceType organizationReference = organizationsMap[organizationName];

      Console.WriteLine(organizationName);

      Console.WriteLine(organizationReference.href);

      Organization organization = Organization.GetOrganizationByReference(client,
organizationReference);

      List<ReferenceType> catalogLinks = organization.GetCatalogRefs();

      if (catalogLinks != null && catalogLinks.Count > 0)

      {

         foreach (ReferenceType catalogLink in catalogLinks)

         {

            Catalog catalog = Catalog.GetCatalogByReference(client, catalogLink);

            CatalogType catalogType = catalog.Resource;

            Console.WriteLine("   " + catalogType.name);

            Console.WriteLine("   " + catalogLink.href);

            List<ReferenceType> catalogItemReferences =
catalog.GetCatalogItemReferences();

            if (catalogItemReferences != null && catalogItemReferences.Count > 0)

            {

               foreach (ReferenceType catalogItemReference in catalogItemReferences)

               {

                  Console.WriteLine("   " + catalogItemReference.name);

                  Console.WriteLine("   " + catalogItemReference.href);

               }

               Console.WriteLine();

            }

            else
```

```
            {
               Console.WriteLine(                                                    "No
CatalogItems Found");
            }
         }
         Console.WriteLine();
      }
      else
      {
         Console.WriteLine(                         "No Catalogs Found");
      }
   }
}
else
{
   Console.WriteLine("No Organizations");
}
```

The following example shows REST API calls that accomplish some of the tasks shown in the example above. REST API calls to list catalog items:

```
GET https://vCloud/api/catalog/id
GET https://vCloud/api/catalog/id/catalogItems
GET https://vCloud/api/catalogitem/id
```

Retrieving vApp Configuration

For a typical user, a vApp is the basic unit of backup specified in vCloud Director. The current generation of backup software maps vApps to their associated virtual machines in vSphere, and thus the virtual machine becomes an actual artifact. Virtual disk and virtual machine configuration files need to be stored in a backup. Along with the associated virtual machine artifacts, the user needs to back up the metadata and properties associated with every vApp to successfully restore it in vCloud Director when needed.

When a vApp is lost or deleted from vCloud Director, backup software can restore the vApp by composing a new vApp using virtual machines restored in vSphere. In such a case it becomes imperative to restore the properties and metadata associated with the vApp in vCloud Director.

The SDK includes a number of methods that you can use to get vApp configuration information. Although some of this information is included in the OVF used to upload the vApp to vCloud Director, the information might have subsequently been modified either by using the vCloud API or through the user interface.

All of these methods apply to an object of type **`vapp`**.

Methods To Retrieve vApp Configuration

- **`GetChildrenVms()`**
  Gets a list of all child virtual machines that constitute a given vApp. Returns **`List<VM>`**.
- **`GetStartupSection()`**
  Get virtual machine startup information. Returns **`StartupSectionType`**.
- **`GetNetworksByName()`**
  Get mapping of all the network sections using their name. Returns **`Dictionary<string, NetworkSection_TypeNetwork>`**.
- **`GetNetworkConfigSection()`**
  Get network configuration details for a vApp. The information typically contains IP scope (gateway, netmask, DNS settings, IP range), Parent network, Fence Mode settings, and so on. Returns **`NetworkConfigSectionType`**.
- **`GetLeaseSettingSection()`**
  Get lease settings information. It includes deployment and storage lease settings for the vApp. Returns **`LeaseSettingsSectionType`**.
- **`GetOwner()`**
  Get owner information for the vApp. Returns **`ReferenceType`**.
- **`GetMetadata()`**
  Every resource in vCloud API can be associated with user-defined metadata. This method returns user-defined metadata associated with a vApp. Returns **`MetadataType`**.

The following example shows REST API calls used to get vApp configuration data.

```
GET https://vCloud/api/vapp/id

GET https://vCloud/api/vapp/id/startupSection

GET https://vCloud/api/vapp/id/networkConnectionSection

GET https://vCloud/api/vapp/id/networkConfigSection

GET https://vCloud/api/vapp/id/leaseSettingsSection

GET https://vCloud/api/vapp/id/owner

GET https://vCloud/api/vapp/id/metadata
```

Virtual Machine Information

vCloud Director also stores virtual machine configuration information uploaded from an OVF file into a vApp template. If you have not modified a virtual machine configuration since uploading, you can use this information to verify the configuration of the virtual machine before restoring it.

The following methods, applied to an object of type **`VM`**, retrieve configuration data structures from vCloud Director.

Configuration Data for a Virtual Machine

- **`GetVirtualHardwareSection()`**
  Get hardware requirements of the virtual machine. Returns **`VirtualhardwareSection_Type`**.
- **`GetOperatingSystemSection()`**
  Get information about the guest operating system installed on this virtual machine. Returns **`OperatingSystemSectionType`**.
- **`GetNetworkConnectionSection()`**

Get information about virtual network devices used by this virtual machine. Returns **NetworkConnectionSectionType**.

- **GetRuntimeInfoSection()**
  Get version of VMware Tools installed on the virtual machine. Returns **RuntimeInfoSectionType**.

The following example shows the REST API calls corresponding to the virtual machine configuration sections available from the SDK for .NET. REST API calls to get virtual machine configuration data:

```
GET https://vCloud/api/vapp/id/virtualhardwaresection

GET https://vCloud/api/vapp/id/operatingSystemSection

GET https://vCloud/api/vapp/id/networkConnectionSection

GET https://vCloud/api/vapp/id/runtimeInfoSection
```

Preventing Updates to a vApp During Backup or Restore

While you are backing up or restoring a vApp, you need to prevent updates to the vApp configuration and metadata so that the vApp remains internally consistent. To prevent updates during the backup/restore process, the vCloud API allows the vApp to be placed in maintenance mode, which rejects any new updates to the configuration and metadata.

The backup software must select maintenance mode for the vApp before starting backup or restore operations, and deselect maintenance mode for the vApp after the operations are completed. The following example shows how to protect a vApp by selecting and deselecting maintenance mode.

```
using com.vmware.vcloud.sdk;

using com.vmware.vcloud.api.rest.schema;

...

VApp vapp; // VApp utility class from vCloud SDK

// Identify vApp

vapp.EnableMaintenance(); // Enter maintenance mode

// Perform backup/restore here

...

vapp.DisableMaintenance(); // Exit maintenance mode
```

The following lines show corresponding REST API calls to select and deselect maintenance mode for a vApp.

```
POST https://vCloud/api/vapp/id/action/enterMaintenancemode

POST https://vCloud/api/vapp/id/action/exitMaintenanceMode
```

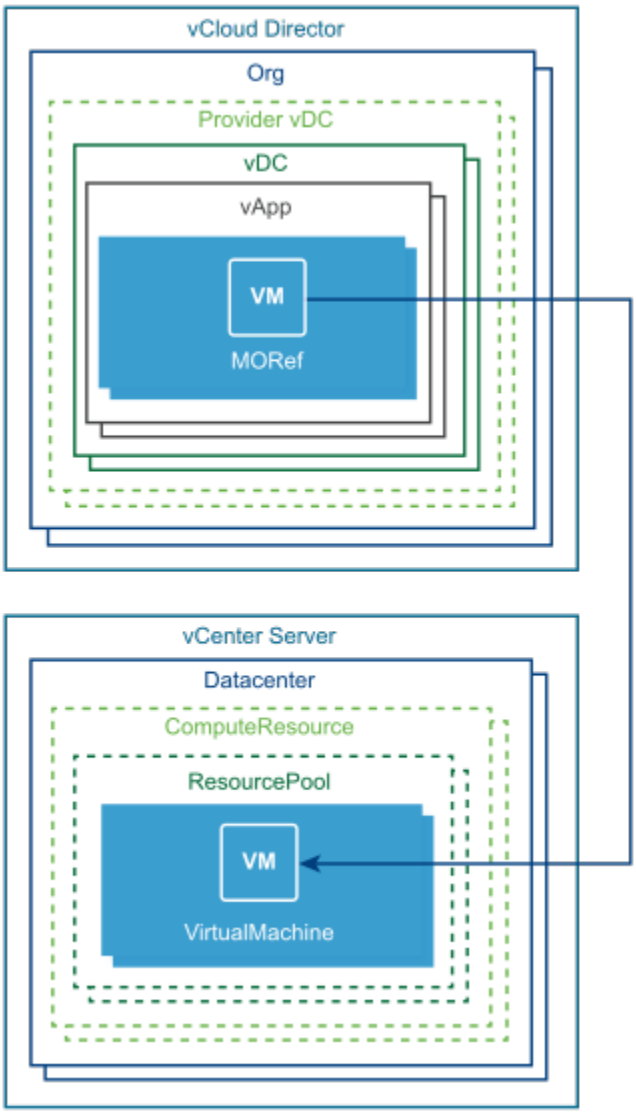
> **NOTE**
> Selecting maintenance mode does not affect current or pending tasks associated with the vApp. Current or pending tasks will run to completion concurrent with the backup or restore operation. If these tasks involve configuration changes, they could result in an inconsistent vApp configuration. The backup system must ensure that such tasks are complete before storing the vApp properties and metadata.

Associating vCloud Resources with vSphere Entities

The admin view of vCloud Director resources provides additional information about the corresponding entities relevant to the vSphere platform. This information is available only when administrative credentials are used to log in to vCloud Director. The additional information does not replace the use of the vSphere API to provide comprehensive information about the entities. It merely provides the bridge between the vCloud and vSphere by mapping the IDs known to the respective systems.

For example, any given virtual machine is known in vCloud Director by a URN that contains the UUID and resource type. The same resource is identified in vSphere using its native identification, a **`MoRef`** (Managed object reference). Additional information provided in the vCloud API makes the necessary link between the two entities by mapping their ID in the two systems. The mapping context is shown in the following figure.

Figure 177: Mapping vApp to virtual machine



The vCloud API describes the mapping in terms of XML elements, shown in the following example. The box in the example highlights XML data that maps a virtual machine from vCloud Director to vSphere. The **`MoRef`** of the virtual machine is in bold type.The object type is shown as **`VIRTUAL_MACHINE`**.

```
// XML Mapping a Virtual Machine URL to a MoRef

<Vm needsCustomization="false" deployed="false" status="3" name="RedHat6"
id="urn:vcloud:vm:f487ba71-058a-47a9-9e9a-def458c63fd5" type="application/
vnd.vmware.vcloud.vm+xml" href="https://10.20.140.167/api/vApp/vm-f487ba71-058a-47a9-9e9a-
def458c63fd5">

   <VCloudExtension required="false">

      <vmext:VmVimInfo>

         <vmext:VmVimObjectRef>

            <vmext:VimServerRef type="application/vnd.vmware.admin.vmwvirtualcenter+xml"
name="dao_w2k8_vc" href="https://10.20.140.167/api/admin/extension/vimServer/
e7026985-19f6-4b9a-9d0d-588629e63347"/>

               <vmext:MoRef>vm-63</vmext:MoRef>

               <vmext:VimObjectType>VIRTUAL_MACHINE</vmext:VimObjectType>

         </vmext:VmVimObjectRef>

         <vmext:DatastoreVimObjectRef>

            <vmext:VimServerRef type="application/vnd.vmware.admin.vmwvirtualcenter+xml"
name="dao_w2k8_vc" href="https://10.20.140.167/api/admin/extension/vimServer/
e7026985-19f6-4b9a-9d0d-588629e63347"/>

            <vmext:MoRef>datastore-29</vmext:MoRef>

            <vmext:VimObjectType>DATASTORE</vmext:VimObjectType>

         </vmext:DatastoreVimObjectRef>

         <vmext:HostVimObjectRef>

            <vmext:VimServerRef type="application/vnd.vmware.admin.vmwvirtualcenter+xml"
name="dao_w2k8_vc" href="https://10.20.140.167/api/admin/extension/vimServer/
e7026985-19f6-4b9a-9d0d-588629e63347"/>

            <vmext:MoRef>host-28</vmext:MoRef>

            <vmext:VimObjectType>HOST</vmext:VimObjectType>

         </vmext:HostVimObjectRef>

         <vmext:VirtualDisksMaxChainLength>1</vmext:VirtualDisksMaxChainLength>

      </vmext:VmVimInfo>

   </VCloudExtension>

</Vm>
```

Besides the virtual machine object itself, the **`VmVIMInfo`** element encapsulated in the **`VCloudExtension`** element of the example lists a datastore object and a host object. Each section provides the vSphere entity reference (**`MoRef`**) for the corresponding entity, along with its type. The types are **`DATASTORE`** and **`HOST`**, respectively. In vCloud Director, the virtual machine can be described as virtual machine **`vm-63`** stored in datastore **`datastore-29`** and managed by vCenter Server **`dao_w2k8_vc`**.

Similarly, the following example shows the administrative view of a VDC wherein the **VCloudExtension** element provides additional information about the corresponding entities in vSphere. In this particular case, the VDC in the example is based on a resource pool configured in vCenter Server, named **dao_w2k8_vc**. More information on this server can be obtained by using the vCloud API and its reference URL, which is available as the **href** property. The **MoRef** element provides the ID of the resource pool that backs the given VDC, as known to vSphere. Since a **MoRef** is treated as an opaque value, the **VimObjectType** element specifies the type of object that the **MoRef** points to. Combining these elements enables you to use the vSphere API and to locate the Resource Pool served by the specified vCenter Server. XML mapping a datacenter URL to a MoRef:

```
<AdminVdc … >

   <VCloudExtension required="false">

      <vmext:VimObjectRef>

         <vmext:VimServerRef type="application/vnd.vmware.admin.vmwvirtualcenter+xml"
name="dao_w2k8_vc" href="https://10.20.140.167/api/admin/extension/vimServer/
e7026985-19f6-..."/>

         <vmext:MoRef>resgroup-52</vmext:MoRef>

         <vmext:VimObjectType>RESOURCE_POOL</vmext:VimObjectType>

      </vmext:VimObjectRef>

   </VCloudExtension>

...

</AdminVdc … >
```

The following example shows how to use SDK helper methods to access the vSphere specific information for the virtual machines of a given vApp.

The return value of the methods has type **VimObjectRefType**, which provides a reference to a vCenter Server, a **MoRef** to the vSphere entity, and the type of the entity it is referring to.

```
// Using the SDK for .NET To Access MoRefs

using com.vmware.vcloud.sdk;

using com.vmware.vcloud.api.rest.schema;

…

// Log in with admin privileges and get admin view of vDC containing the vApp.

…


VApp vapp; // VApp utility class from vCloud SDK

// Identify vApp.

…


List<VM> Vms;

// Get list of children VM(s)
```

```
Vms = vapp.GetChildrenVms();
foreach (VM vm in Vms)
{
      Console.WriteLine();
  // Access vSphere information for VM
  …
  // VM Info from vSphere
  VimObjectRefType vmRef = vm.GetVMVimRef();
  Console.WriteLine("VirtualMachine: " + vmRef.moRefField);

  // Datastore Info from vSphere for VM
  VimObjectRefType datastoreRef = vm.GetVMDatastoreVimRef();
  Console.WriteLine("Datastore: " + datastoreRef.moRefField);

  // Host info form vSphere for VM
  VimObjectRefType hostRef = vm.GetVMHostVimRef();
  Console.WriteLine("Host: " + hostRef.moRefField);
}
```

Restoring vApps

During the restore process, the backup software typically restores a virtual machine in vSphere using the virtual machine configuration and disk files. In situations where the vApp has been lost from the vCloud Director inventory, the backup software needs to first restore the virtual machine in vSphere, and then import the virtual machine into vCloud Director.

Although the vApp may contain multiple virtual machines in the view of vCloud Director, the virtual machines are known individually to vSphere. To complete the restore operation, the backup software needs to re-create the restored vApp so that all the member virtual machines are created as child virtual machines of the vApp.

To re-create the vApp using the vCloud SDK for .NET, the backup software must use two method calls: **`ImportVmAsVapp`** and **`ImportVmIntoVapp`**. Use the **`ImportVmAsVapp`** method to create a vApp from any one of the child virtual machines. Then call the **`ImportVmIntoVapp`** method once for each remaining child virtual machine. The following example shows how to use both methods to create a vApp using the vCloud SDK.

```
// Importing Virtual Machines into vApps
/// <summary>
/// Reference to hold the vCloud Client reference
/// </summary>
private static VcloudAdminExtension extension = null;
```

```
vcloudClient.login(user,  password);

extension = vcloudClient.getVcloudAdminExtension();


// Get references for known VIM Servers

Dictionary<string, ReferenceType> vimServerRefsByName =
extension.GetVMWVimServerRefsByName();


// Select VIM Server Reference

VMWVimServer vimServer = VMWVimServer.GetVMWVimServerByReference(vcloudClient,
vimServerRefsByName[vimServerName]);

...

// Import first VM from VIM server as vApp:

ImportVmIntoVAppParamsType importVmIntoVAppParamsType = new ImportVmIntoVAppParamsType();

importVmIntoVAppParamsType.vmMoRefField = moref; // vSphere ID from backup data.

importVmIntoVAppParamsType.vdcField = vdcRef; // vDC where the new vApp will be created.

Vapp vapp = vimServer.ImportVmAsVApp(importVmAsVAppParamsType); // Task is embedded in
vapp.

...

foreach (VM vm in vms)

{

   // Import remaining VMs from VIM Server into existing vApp:

   …

   importVmIntoVAppParamsType.vmMoRefField = moref; // vSphere ID from backup data.

   importVmIntoVAppParamsType.vAppField = vapp; // vApp to hold restored VMs.

   Task task = vimServer.ImportVmIntoVApp(importVmIntoVAppParamsType);

   …

};
```

The following example shows the corresponding REST API calls used to rebuild a vApp in vCloud Director.

```
POST https://vCloud/api/admin/extension/vimServer/id/importVmAsVApp

POST https://vCloud/api/admin/extension/vimServer/id/importVmIntoExistingVApp
```

Conclusion for vApp Backup

This chapter provided an overview of how to use the vCloud SDK for .NET to back up and restore vApps in vCloud Director. This information serves as a guide to using the vCloud SDK for writing backup and restore software. Other documentation is require to supplement aspects not described in this chapter.

The examples in this chapter are not intended to be complete. They are intended only to illustrate the method calls you would use during backup and restore operations with vCloud Director and vCenter Server. For more detail about the SDK methods and examples of their use, see the *vCloud SDK for .NET Developer's Guide*.

Virtual Disk Mount API

To perform file based restore, you can use the disk mount API (**vixMntapi**) for local and remote mounting of virtual disks. VixMntapi involves a separate library for programs to load.

> **NOTE**
> As of VDDK 6.7.1, the VixMntapi library for Windows and Linux supports NBDSSL, HotAdd, and SAN transport. For earlier VDDK versions, and when used with earlier vSphere releases, the **vixMntapi** library for Linux supported only file and NBD transport.

The VixMntapi Library

The VixMntapi library supports guest operating systems on multiple platforms. On POSIX systems it requires FUSE mount, available on recent Linux systems, and freely available on the Github web site. Definitions are contained in the following header file, installed in the same directory as **vixDiskLib.h**:

```
#include "vixMntapi.h"
```

Types and Structures

This section summarizes the important types and structures.

Operating System Information

The **VixOsInfo** structure encapsulates the following information:

- Family of the guest operating system, **VixOsFamily**, one of the following:
  - Windows (NT-based)
  - Linux
  - Netware
  - Solaris
  - FreeBSD
  - OS/2
  - Mac OS X (Darwin)
- Major version and minor version of the operating system
- Whether it is 64-bit or 32-bit
- Vendor and edition of the operating system
- Location where the operating system is installed

Disk Volume Information

The **VixVolumeInfo** structure encapsulates the following information:

- Type of the volume, **VixVolumeType**, one of the following:
  - Basic partition.
  - GPT – GUID Partition Table.
  - Dynamic volume, including Logical Disk Manager (LDM).
  - LVM – Logical Volume Manager disk storage. Not supported with Linux.
- Whether the guest volume is mounted on the proxy.
- Path to the volume mount point on the proxy, or NULL if the volume is not mounted.
- On Windows, **numGuestMountPoints** is the number of times a basic volume is mapped to a drive letter, or 0 if the volume is not mounted. IDE and boot disk come first. Unimplemented on Linux.
- Mount points for the volume in the guest.

Function Calls

To obtain these functions, load the vixMntapi library separately from the vixDiskLib library. On Windows, compile with the `vixMntapi.lib` library so your program can load the `vixMntapi.dll` runtime.

These calls can be used to mount and read Windows virtual disks on Windows hosts (with at least one NTFS volume) or Linux virtual disks on Linux hosts. Cross-mounting is restricted, though it is possible to mount a virtual disk with a mix of formats, if the mounted partition was formatted with Windows.

You should run only one vixMntapi program at a time on a virtual machine, to avoid conflict between registry hives. See Multithreading Considerations for advice on worker threads.

Available functions in the library are listed below. Under parameters, [in] indicates input, and [out] indicates output parameters. Functions that return **vixError** return **VIX_OK** on success, otherwise a VIX error code.

VixMntapi_Init()

Initializes the library. This is similar to **VixDiskLib_InitEx()** – see Initialize Virtual Disk API. You should call **VixMntapi_Init()** only once per process.

```
VixError

VixMntapi_Init(uint32 majorVersion,

               uint32 minorVersion,

               VixDiskLibGenericLogFunc *log,

               VixDiskLibGenericLogFunc *warn,

               VixDiskLibGenericLogFunc *panic,

               const char *libDir,

               const char *configFile);
```

Parameters:

- **majorVersion** [in] VixMntapi major version number, currently must be 1 (one).
- **minorVersion** [in] VixMntapi minor version number, currently must be 0 (zero).
- **log** [in] Callback function to write log messages.
- **warn** [in] Callback function to write warning messages.
- **panic** [in] Callback function to report fatal errors.
- **libDir** [in] and **configFile** [in] as for **VixDiskLib_InitEx()** allows you to set **tmpDirectory**.

VixMntapi_Exit()

Cleans up the VixMntapi library. You should call **VixMntapi_Exit()** only once per process

```
void VixMntapi_Exit();
```

VixMntapi_OpenDiskSet()

Opens the set of disks for mounting on a Windows virtual machine. All the disks for a dynamic volume or Logical Disk Manager (LDM) must be opened together.

```
VixError

VixMntapi_OpenDiskSet(VixDiskLibHandle diskHandles[],

                      int numberOfDisks,

                      uint32 openMode,

                      VixDiskSetHandle *diskSet);
```

The **VixDiskLibHandle** type, defined in **vixDiskLib.h**, is the same as for the **diskHandle** parameter in the **VixDiskLib_Open()** function, but here it is an array instead of a single value.

Parameters:

- **diskHandles** [in] Array of handles to open disks.
- **numberOfDisks** [in] Number of disk handles in the array.
- **openMode** [in] Must be 0 (zero).
- **diskSet** [out] Disk set handle to be filled in.

If you want to mount disks on a Windows system, first call **VixDiskLib_Open()** for every disk, then use the returned disk handle array to call **VixMntapi_OpenDiskSet()**, which returns a disk set handle.

If you want to mount disks on a Linux system, call the function **VixMntapi_OpenDisks()**, which opens and creates the disk set handle, all in one function.

VixMntapi_OpenDisks()

Opens disks for mounting on a Linux virtual machine, or disk sets on a Windows virtual machine. VixMntapi supports whatever file systems the proxy supports. On Linux, VixMntapi does not support Logical Volume Manager (LVM), although vendors can write their own scripts to support LVM disks.

```
VixError

VixMntapi_OpenDisks(VixDiskLibConnection connection,

                            const char *diskNames[],

                            size_t numberOfDisks,

                            uint32 openFlags,

                            VixDiskSetHandle *handle);
```

Parameters:

- **connection** [in] The **VixDiskLibConnection** to use for opening the disks. Calls **VixDiskLib_Open()** with the specified flags for each disk to open.
- **diskNames** [in] Array of disk names to open.
- **numberOfDisks** [in] Number of disk handles in the array. Must be 1 for Linux.
- **flags** [in] Flags to open the disk.
- **handle** [out] Disk set handle to be filled in.

VixMntapi_GetDiskSetInfo()

Retrieves information about the disk set.

```
VixError

VixMntapi_GetDiskSetInfo(VixDiskSetHandle handle,

                         VixDiskSetInfo **diskSetInfo);
```

Parameters:

- **handle** [in] Handle to an open disk set.
- **diskSetInfo** [out] Disk set information to be filled in.

VixMntapi_FreeDiskSetInfo()

Frees memory allocated by **VixMntapi_GetDiskSetInfo()**.

```
void VixMntapi_FreeDiskSetInfo(VixDiskSetInfo *diskSetInfo);
```

Parameter:

- **diskSetInfo** [in] OS info to be freed.

VixMntapi_CloseDiskSet()

Closes the disk set. Note, **vixDiskLib_Disconnect()** invalidates open file handles, so call this function first.

```
VixError

VixMntapi_CloseDiskSet(VixDiskSetHandle diskSet);
```

Parameter:

- **diskSet** [in] Handle to an open disk set.

VixMntapi_GetVolumeHandles()

Retrieves handles to volumes in the disk set. The third parameter **VixVolumeHandle** can be a volume handle or an array of volume handles. If you pass an array this function returns the volume handle for the first volume only. If you pass a pointer (such as **VixVolumeHandle *volumeHandles**) it returns all the volume handles.

```
VixError

VixMntapi_GetVolumeHandles(VixDiskSetHandle diskSet,

                           int *numberOfVolumes,
```

```
                                VixVolumeHandle **volumeHandles);
```

Parameters:

- **diskSet** [in] Handle to an open disk set.
- **numberOfVolumes** [out] Number of volume handles.
- **volumeHandles** [out] Volume handles to be filled in.

VixMntapi_FreeVolumeHandles()

Frees memory allocated by **VixMntapi_GetVolumeHandles()**.

```
void VixMntapi_FreeVolumeHandles(VixVolumeHandle *volumeHandles);
```

Parameter:

- **volumeHandles** [in] Volume handle to be freed.

VixMntapi_GetOsInfo()

Retrieves information about the default operating system in the disk set.

To get operating system information, **VixMntapi_GetOsInfo()** requires the system and boot volumes to be already mounted. It does not dismount the system volume at the end of this function. Your application should be prepared to handle the “volume already mounted” error gracefully.

This function is effective only when used with operating systems of the same type. For instance, a VixMntapi program running on Windows can provide information about the VMDK of a Windows virtual machine, but not about the VMDK of a Linux virtual machine.

```
VixError

VixMntapi_GetOsInfo(VixDiskSetHandle diskSet,

                    VixOsInfo **info);
```

Parameters:

- **diskSet** [in] Handle to an open disk set.
- **info** [out] OS information to be filled in.

VixMntapi_FreeOsInfo()

Frees memory allocated by **VixMntapi_GetOsInfo()**.

```
void VixMntapi_FreeOsInfo(VixOsInfo* info);
```

Parameter:

- **info** [in] OS info to be freed.

VixMntapi_MountVolume()

Mounts the volume. After mounting the volume, use **VixMntapi_GetVolumeInfo()** to obtain the path to the mounted volume. This mount call locks the source disks until you call **VixMntapi_DismountVolume()**. The **VixMntapi_MountVolume()** function cannot mount Linux swap or extended partitions.

```
VixError

VixMntapi_MountVolume(VixVolumeHandle volumeHandle,

                      Bool isReadOnly);
```

Parameters:

- **volumeHandle** [in] Handle to a volume.
- **isReadOnly** [in] Whether to mount the volume in read-only mode. Does not override **openMode**.

VixMntapi_DismountVolume()

Unmounts the volume.

```
VixError

VixMntapi_DismountVolume(VixVolumeHandle volumeHandle,

                      Bool force);
```

Parameters:

- **volumeHandle** [in] Handle to a volume.
- **force** [in] Force unmount even if files are open on the volume.

VixMntapi_GetVolumeInfo()

Retrieves information about a disk volume. Some information, such as the number of mount points, requires you to set the open read-only flag. Some information is available only if a volume was previously mounted by **VixMntapi_MountVolume()**. The Windows registry returns volume information only for mounted disks. On Windows the **VixMntapi_GetVolumeInfo()** call returns a symbolic link from the **VixVolumeInfo** structure in the form `\\.\vstor2-mntapi10-shared-<longhexnum>\`. You can transform this symbolic link into a target path by replacing `\\.` with `\Device` and deleting the final backslash, then map a drive letter with **DefineDosDevice(DDD_RAW_TARGET_PATH,...)** and proceed as if you have a local drive. Alternatively on Windows, you can open a volume with **CreateFile()** and traverse the file system with **FindFirstFile()**.

```
VixError

VixMntapi_GetVolumeInfo(VixVolumeHandle volumeHandle,

                      VixVolumeInfo **info);
```

Parameters:

- **volumeHandle** [in] Handle to a volume.
- **info** [out] Volume information to be filled in.

VixMntapi_FreeVolumeInfo()

Frees memory allocated in **VixMntapi_GetVolumeInfo()**.

```
void VixMntapi_FreeVolumeInfo(VixVolumeInfo *info);
```

Parameter:

- **info** [in] Volume info to be freed.

Programming with VixMntapi

At the top of your program, include **vixMntapi.h** along with any other header files you need. Structures and type definitions are declared in the include file, so you do not need to create them or allocate memory for them.

Call **VixMntapi_Init()** to initialize the library in your application. This function takes major and minor version number to account for future extensions. You can provide your own logging, warning, or panic functions to substitute for the default VixMntapi handlers, and custom library and temporary directories.

Call **VixMntapi_OpenDiskSet()** to open a set of virtual disks for mounting. Pass a set of disk handles obtained from the **VixDiskLib_Open()** call. The **VixMntapi_OpenDiskSet()** function also expects number of disks to open, an optional open mode, and a parameter to pass back the resulting disk-set handle.

File System Support

Traditional Windows file systems (including FAT, FAT32, and NTFS) are supported. Linux file systems (including ext2, ext3, ext4) are supported if the proxy machine supports them. VixMntapi depends on an operating system for file attributes such as compression, encryption, hidden files, ACL, and alternate streams. If a vixMntapi-linked application runs on a virtual machine that supports these attributes, vixMntapi can support them. Windows basic volumes and simple dynamic volumes are supported for writing, but spanned, striped (RAID-0), mirrored (RAID-1) and parity (RAID-5) volumes are supported read-only for VixMntapi.

You must open a disk set read/write to obtain the OS information for dynamic volume types including LDM and LVM. If you cannot open a base disk read/write, create a child disk in front, and open it read/write. In a multi-boot setup, only the first entry #0 is opened.

The order of mounting is important. For instance, mount top-level directories before subdirectories, and drives with dependencies after drives that they depend on. Mount points are not enumerated, nor are they restored. When you mount one volume, the other volumes are not implicitly mounted also.

Diagnostic Logging for VixMntapi

As of VDDK 6.7.1, logging is available to help diagnose behavior of the VixMntapi library. Currently this facility is available for Linux but not Windows.

The output file is named `vixMntapi.log` and appears in the same directory as other log files, as set by the `tmpDirectory` line in the VDDK configuration file passed as the seventh parameter to `VixDiskLib_InitEx`. Logging levels are the same as for **vixDiskLib.transport**:

- 0 = Panic (failure)
- 1 = Error
- 2 = Warning
- 3 = Audit
- 4 = Info
- 5 = Verbose
- 6 = Trivia

Read-Only Mount on Linux

As of VDDK 6.7.1, read-only mount is supported on Linux, as it was previously on Windows. This section describes a Linux solution for earlier VDDK releases. Now applications can mount the base disk of a snapshot with the read-only flag.

Here are possible use cases for read-only mounting of virtual disks:
- Avoid accidentally writing data to the disk, which is opened read/write mode.
- Retain current disk data after the VM is powered off.
- Use previous VM as a restore point by reverting to its original snapshot.

In VDDK 6.7.0 and before, Linux VixMntapi did not support read-only access. To mount a disk read-only, applications had to either mount the virtual disk of a powered off virtual machine, or mount the snapshot of a powered on virtual machine. The VixMntapi library can combine the two techniques to create read-only disk, as below.
To mount disks while saving a read-only copy:

1. Power off the VM if it is on.
2. Create a snapshot. Optionally revert to a previous snapshot if you want.
3. Connect to the VM for read/write access with `VixDiskLib_Connect` or `VixDiskLib_ConnectEx`.
4. Call **`VixMntapi_OpenDisks`** to open the current (not the snapshot) VM disk in read/write mode.
5. Mount volumes to access files. Write operations can be performed, but they write to the redo log of the current VM disk, not to the snapshot.
6. Close the disk and disconnect.
7. Revert to the snapshot created in step 2, to eliminate any write changes. Delete the snapshot.
8. Power on the VM if it was on before.

Sample VixMntapi Code

You call the VixMntapi functions after initializing VixDiskLib, connecting to a virtual machine, and opening a disk handle. The following example shows test code for Windows with the correct order of function calls.

```
MountTest() {

     vixError = VIX_ERR_CODE(VixDiskLib_Init() );

     vixError = VIX_ERR_CODE(VixMntapi_Init() );

     VixDiskLib_ConnectEx(&connectParams, TRUE, NULL, NULL, &connection));

     diskHandles = GetMyDiskHandles(diskPaths, connection, &connectParams, flags,
&numberOfDisks);

     vixError = VIX_ERR_CODE(VixMntapi_OpenDiskSet(diskHandles, numberOfDisks, flags,
&diskSet));

     GetOsInfo(diskSet);

     vixError = VIX_ERR_CODE(VixMntapi_GetVolumeHandles(diskSet, &numberOfVolumes,
&volumeHandles));

     for(size_t i = 0; i < numberOfVolumes; i++) {

          VixVolumeHandle volumeHandle = volumeHandles[i];

          VixVolumeInfo *volumeInfo;

          vixError = VIX_ERR_CODE( VixMntapi_MountVolume(volumeHandle, TRUE) );

          vixError = VIX_ERR_CODE( VixMntapi_GetVolumeInfo(volumeHandle, &volumeInfo) );

          VixMntapi_FreeVolumeInfo(volumeInfo);
```

```
            VerifyMountedVolume();
            CleanUpMountedVolume(volumeHandle, volumeInfo);
      }
      VixMntapi_FreeVolumeHandles(volumeHandles);
      vixError = VIX_ERR_CODE( VixMntapi_CloseDiskSet(diskSet) );
      FreeMyDiskHandles(diskHandles, numberOfDisks);
      VixMntapi_Exit();
      VixDiskLib_Exit();
}
```

Restrictions on Virtual Disk Mount

The following limitations apply when mounting virtual disks:

- You cannot mount virtual disks that are in use by a running or suspended virtual machine. You can mount virtual disks from any powered off virtual machine, or base disks when a virtual machine is running read-only off a snapshot.
- You can mount the last snapshot in a chain read/write, but you must mount previous snapshots read-only.
- If you specify a virtual disk with snapshots on a powered off virtual machine, VixMntapi locates and mounts the last snapshot in the disk chain. While a disk is mounted, do not revert to a previous snapshot using another VMware interface – this would make it impossible to unmount the partition.
- You cannot mount virtual disk if any of its `.vmdk` files are encrypted, compressed, or read-only. However you can change these attributes and then mount the virtual disk.
- With Windows, you must mount virtual disks on drive D: or greater, and choose a drive letter not in use.
- With Linux, kernel version 2.6 or higher is required to run the FUSE (file system in user space) module. You cannot mount Linux swap or extended partitions. Logical Volume Manager (LVM) is not supported.
- On Linux virtual machines before VDDK 5.5, you could not mount previous snapshots in the chain.
- You can mount Windows virtual disks on Windows hosts (with an NTFS volume) or Linux virtual disks on Linux hosts. Cross-mounting is restricted but may be allowed for cross-formatted file systems.
- The C: boot driver should be on **`scsi:0:0`**, and all disks should be opened in SCSI order (0:0, 0:1, 0:2, 1:0, 1:1, 1:2, etc.) before mounting any of them.

Error Codes and Open Source

This chapter summarizes error codes and open source components of VDDK.

Recent Changes

In vSphere 6.0, VDDK libraries were changed to return **`VIX_E_OUT_OF_MEMORY`** when an ESXi host run out of memory for network file copy (NFC). Previously, VDDK would hang during I/O.

Error 2 – **`VIX_E_OUT_OF_MEMORY`**

Memory allocation failed. Out of memory.

Finding Error Code Documentation

For a list of Virtual Disk API error codes, see the online reference guide *`Introduction to the VixDiskLib API`*:

- Windows – *InstallFolder*\VDDK\doc\errorsList.html
- Linux – *InstallDir*/vddk/doc/errorsList.html

In a Web browser, click the Error Code link in the alphabetic list, and the numeric error code explanation will appear.

Association With VIX API Errors

The Virtual Disk API shares many errors with the VIX API, which explains the **VIX** prefix. The error codes for VIX API were similar to a comparable release of VDDK. However VIX API was deprecated and its documentation is no longer available. Most VIX errors are similar to the C language `errno.h` header file, so you can deduce their meaning from standard Linux documentation.

Interpreting Error Codes

A VIX error is a 64-bit value. A value of **VIX_OK** indicates success, but otherwise (if there is an error), several bit regions in the 64-bit value might be set. The least significant 16 bits are set to the error code described for VIX errors. More significant bit fields might be set to other values.

As for the VIX API, use the macro **VIX_ERROR_CODE(err)** to mask off bit fields not used by the VDDK.

Troubleshooting Dynamic Libraries

On Windows, the SSL library is placed in the same directory as other vixDiskLib dynamically loaded libraries. On Linux, functions that load the libraries **libssl.so.0.9.8** and **libcrypto.so.0.9.8** do the following:

1. Attempt to load them from the environment's **LD_LIBRARY_PATH** location.
2. Next, attempt to load them from the directory where **libvixDiskLib.so** is located.
3. Next, attempt to load them from the directory where the executable is located.
4. Failing that, exit with an error.

On install, VDDK creates the directory `/usr/lib/vimware-vix-disklib`, populated with 64-bit executables and libraries placed into subdirectories **bin64** and **lib64**. On determining the OS type, VDDK copies the vixDiskLib and vixMntapi libraries into `/usr/lib`. It does not copy **libssl.so.0.9.8** or **libcrypto.so.0.9.8** into `/usr/lib`.

On execution, the root user normally has no **LD_LIBRARY_PATH**, and `/usr/lib` is ahead of `/opt/vmware/lib` in the path. Running the **ldd** command can help diagnose where a program is getting **libvixDiskLib.so** and other libraries. The `/opt/vmware/lib` directory is neither created nor updated by the VDDK install script.

If you see the error "Failed to load library libcrypto.so.0.9.8" there are several solutions:

- Set or reset the **LD_LIBRARY_PATH** environment so it contains one of the directories above, `/lib64` and possibly `/bin64`, before it contains `/usr/lib`.
- Change the symbolic link in `/opt/vmware/lib` (or elsewhere) so it points to the directory above, `/lib64`.
- Copy the **libssl** and **libcrypto** libraries from `/usr/lib/vmware-vix-disklib/lib64` into `/usr/lib`.

Troubleshooting Changed Block Tracking (CBT)

This section provides two hints for helping customers with CBT failures.

Verify Change Tracking Enabled

If you reconfigure a virtual machine to set **changeTrackingEnabled**, but the property remains false, check that you have queried the virtual machine status with **VirtualMachine->config()** after reconfiguration with **VirtualMachine->reconfigure()** and not before. Also make sure that virtual machine compatibility is hardware version 7 or higher, and that the virtual machine has undergone a stun-unstun cycle since reconfiguration.

Run the vixDiskCheck Utility

If customers encounter CBT issues when restoring files on a destination VM, they can use **vixDiskCheck** to ascertain whether there was data corruption after backing up the source VM. The **vixDiskCheck** utility should be run on the backup proxy VM, either Linux or Windows.

When you unpack the VDDK distribution tarball or ZIP file, the **bin/vixDiskCheck** utility is available in the resulting folder. Add the pathname of this folder to the `LD_LIBRARY_PATH` environment on Linux, or to the PATH environment on Windows. Examples below use Bash and PowerShell with install locations `/opt/vddk` and VMware\VDDK.

```
$ export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/opt/vddk/lib64"
```

```
> $env:Path += 'C:\Program Files\VMware\VDDK\bin'
```

You can run **vixDiskCheck** or **vixDiskCheck.exe** to see its options with explanations. Follow these steps to diagnose:

1. Delete previous snapshots, if any. Then reset CBT by deactivating and enabling it again.
2. Power on the VM. Collect the following files and look for the `power-on` tag.
   - CTK files in the `vmfs/volumes/datastore/`*`VM`* folder, for example *`BaseDisk`*`-ctk.vmdk`
   - `hostd.log` in `var/run/log` and `vmware.log` in `vmfs/volumes/datastore/`*`VM`*
3. Take a snapshot (snapshot-1) using the vSphere Client. Collect:
   - CTK file in the VM folder under `vmfs/volumes/datastore`, *`BaseDisk`*`-ctk.vmdk` again.
4. Run **vixDiskCheck** to verify consistency of active full backup. Redirect `stdout` and `stderr` into `vixDiskCheck.log` in the current folder, as below. The CBT backup file must be large enough to hold the entire disk.

   ```
   vixDiskCheck cbtCheck -host IPaddr -user username -password password -vm moref=VmMore
   f

     -ssmoref SnapshotMoRef -libdir VDDKlibDir -thumb SSLhostThumbprint -cbtdir
   CBTbackupFile

     -disk Base-disk.vmdk -transportMode nbdssl 2>&1 | tee vixDiskCheck-01.log
   ```

5. Search for CBT DIFF in the log. If any lines are output, then the CBT file is corrupted.

   ```
   grep "CBT DIFF:" vixDiskCheck.log
   ```

6. Delete the previous snapshot and take a new snapshot (snapshot-2). Collect the CTK file and logs tagged snapshot-2.
7. Run **vixDiskCheck** again to verify consistency of the incremental backup with snapshot-2. Use the same command options as before, except for the snapshot MoRef and the log file path.

   ```
   vixDiskCheck cbtCheck -host IPaddr -user username -password password -vm
   moref=VmMoref

     -ssmoref Snapshot2MoRef -libdir VDDKlibDir -thumb SSLhostThumbprint -cbtdir
   CBTbackupFile

     -disk Base-disk.vmdk -transportMode nbdssl 2>&1 | tee vixDiskCheck-02.log
   ```

8. Repeat the previous step to check consistency of subsequent incremental backups. Make sure to change the snapshot MoRef and the log file name accordingly.

The `-cbtdir` flag specifies the CBT working directory; **vixDiskCheck** creates `backup.dat` and `cbt.inf` files in that folder. Large file `backup.dat` is the full backup of the target disk as a sparse local file, which means only allocated areas are written to it, while other areas are holes. Small text file `cbt.inf` stores the *`changeId`* of the snapshot in the latest run, and is read in the next run, so that the utility can `queryChangedDiskAreas` against the snapshot with this *`changeId`* for incremental backup.

In **`vixDiskCheck`** logs,

- "compare in range" means the range has no changes reported by CBT between the two snapshots. It compares data in the created file with data on the source disk.
- "write in range" means the range has changes reported by CBT between the two snapshots. It writes data on the created file from the source disk.

Open Source Components

VDDK contains the following open source modules and others, with license types indicated:

- Boost (BSD style license)
- Curl (MIT/X derivative license)
- Expat (BSD style license)
- FreeBSD (BSD license)
- ICU, International Components for Unicode (BSD style license)
- LibXML2 (MIT style license)
- OpenLDAP (OpenLDAP v 2.8 license)
- OpenSSL (OpenSSL license)
- Zlib (BSD license)

These open source components have the GNU library general public license:

- GetText (LGPL2.0)
- Glib (LGPL 2.0)
- LibFuse (LGPL2.0)
- LibIconv (LGPL2.0)

HTML Console SDK Programming Guide

The *VMware HTML Console SDK Programming Guide* explains how to use the HTML Console SDK API to write browser-based applications for VMware vSphere® and other VMware products.

Revision History

This book is revised with each release of the product or when necessary. A revised version can contain minor or major changes.

Table 113: Revision History

Revision Date	Description
3 July 2023	Changed some chapter and section titles for search engine optimization.
9 November 2022	Revised for the 2.2 release with code sample for remote copy.
17 April 2017	Revised manual for incorporation in the vSphere documentation center.
27 October 2016	No changes for the 2.1 release.
12 April 2016	Very lightly revised for the 2.0 release.
15 September 2015	Initial release of the *VMware HTML Console Programming Guide*.

Intended Audience

This book is intended for anyone who needs to develop applications containing a remote console. Typically this includes software developers who are creating Web applications using JavaScript, and who are targeting virtual machine remote-console functions in vSphere and vCloud Director.

Related Documentation

Documentation for the HTML Console SDK is available on the VMware developer documentation website. Go to http://developer.vmware.com/sdks/.

Background information for the tasks discussed in this book is available in the vSphere documentation set, for both ESXi hosts and vCenter Server. Go to https://techdocs.broadcom.com/.

The VMware HTML Console

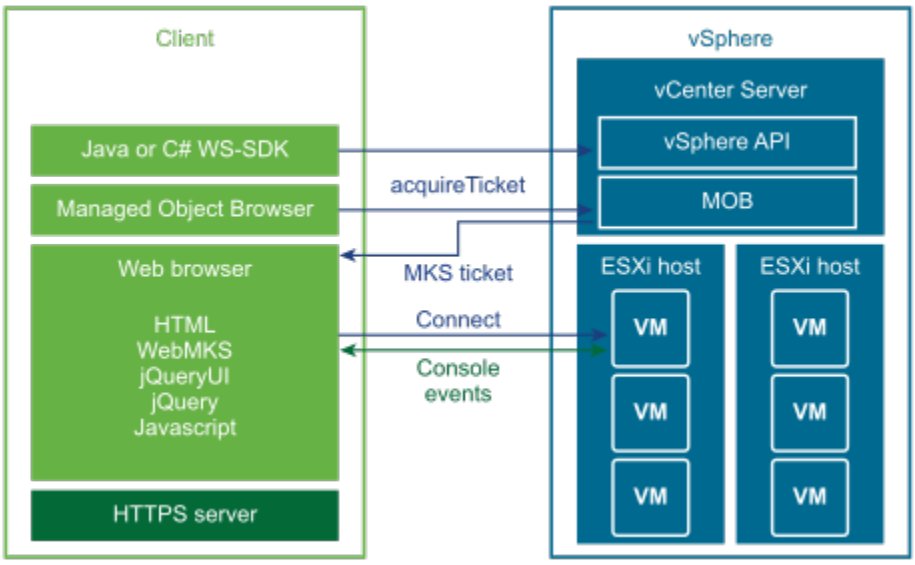
You can use the HTML Console SDK to connect a Web browser with the user interface of a virtual machine.

Architecture of the HTML Console

The HTML Console is a remote console implemented in JavaScript to manage mouse, keyboard, and screen (MKS) events. HTML Console replaces most features of VMRC, the VMware remote console application.

When used with vSphere, your client Web browser connects to a virtual machine console, as shown in WebMKS Architecture.

Figure 178: WebMKS Architecture

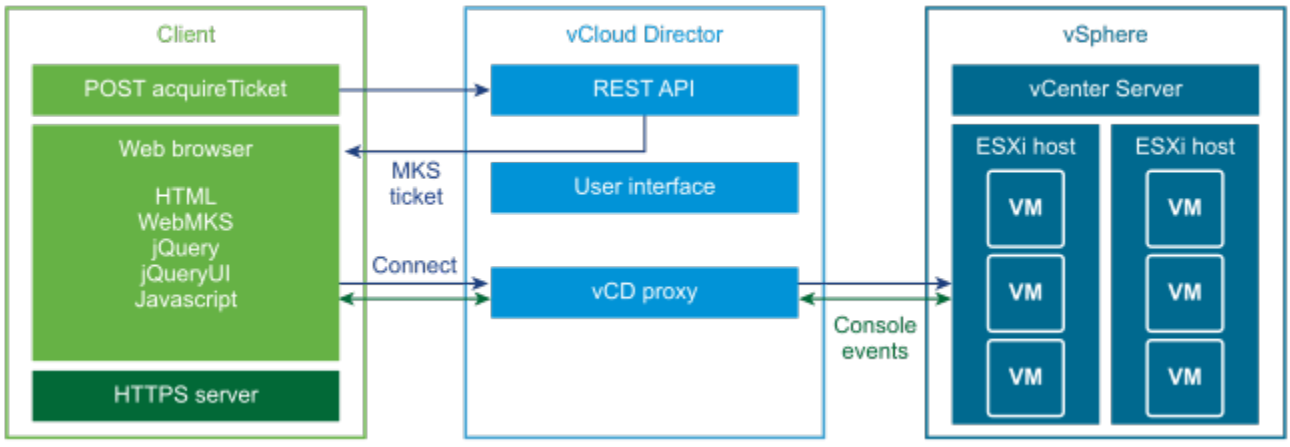


To display a remote console, the client system uses a Web browser to open an HTML file. This file loads the WebMKS Javascript application after calling the jQueryUI and jQuery libraries. Connecting to a virtual machine on vSphere requires an MKS ticket. You can obtain a ticket from the vCenter Server that manages the ESXi host where the virtual machine resides.

After the ESXi host authenticates the MKS ticket, a remote console appears and the Javascript application issues console commands by calling various methods. WebMKS is a VMware implementation to handle remote console events, written in JavaScript for HTML5 compliant browsers. WebMKS is also called WMKS.

When used with vCloud Director, the client communicates with the target virtual machine through the vCloud Director console proxy, as shown in WebMKS with vCloud Director. See the *vCloud API Programming Guide* for an example that shows how to display the virtual machine console.

Figure 179: WebMKS with vCloud Director



Remote HTML Console Features

The HTML Console SDK allows developers to add a virtual machine console to an existing Web interface running on VMware vSphere or vCloud Director. With this SDK, end users can interact with the virtual machine console by typing on the keyboard and moving the mouse.

The remote access console can display either a command line or a graphical user interface. It accepts keyboard and mouse or touch screen input, including function keys and on-screen buttons. Keyboard layout selections include French, German, Italian, Japanese, Portuguese, Spanish, Swiss-French, Swiss-German, and possibly other locals. The console display is updated according to mouse movement, typing input, and gestures on your client machine. Gestures include touchscreen scrolling and button tap on a mobile device.

The HTML Console SDK API contains various methods that connect to and communicate with a virtual machine. It also triggers events to notify users of changes to the virtual machine's state. You program these methods and callbacks so users can remotely manage a virtual machine from any system with the appropriate Web browser. Methods include console commands for changing screen mode, sending special key sequences to the virtual machine, and connecting client devices to the virtual machine. The API implements callbacks that communicate back and forth when the user creates console events, or when the virtual machine responds to user input.

The HTML Console SDK is written in JavaScript, based on jQuery and jQueryUI. The jQuery library simplifies navigation and event handling, while jQueryUI provides visual effects such as those in a window system. These libraries can be called from the public Web, or bundled with your application to run in secure environments. WebMKS provides mouse, keyboard, and screen access to a remote console from any supported Web browser. In the SDK, the minimized Javascript file `wmks.min.js` is the WebMKS implementation.

Compared to Other VMware Consoles

The HTML Console is not the only VMware virtual machine console.

The HTML Console SDK is a tool that customers and developers can use to implement remote virtual machine access in a Web application. It differs from earlier remote consoles, and ones built into vSphere.

Compared to the VMRC SDK Plug-in

With the release of vSphere 5.1 in 2012, the VMware Remote Console (VMRC) provided a way to insert a virtual machine console into a Web application. On Windows, the VMRC SDK contained an executable Win32 plug-in for Internet Explorer. On Linux, the SDK contained a 32-bit or 64-bit NPAPI shared object for Mozilla Firefox and Google Chrome. NPAPI was the Netscape plug-in API, now discontinued.

Due to the discontinuation of NPAPI and decreased use of Internet Explorer, alternatives were needed. The final VMRC release was for vSphere 6.0 in 2015.

Compared to vSphere Web Client Consoles

In the vSphere Web Client view of a virtual machine, Summary tab, click the small image of a console to open a browser tab showing a full image of the console screen. If the console is larger than your browser, scroll bars appear. This remote console implementation is similar to WebMKS, but not identical.

In the vSphere Web Client view of a virtual machine, click **Launch Remote Console** to open a stand-alone application for Windows, Mac OS X, or Linux. VMRC appears on upper left in the title bar. This application is not the discontinued VMRC SDK, but is descended from the remote console on VMware Workstation. The stand-alone VMRC offers features not available in the browser console, such as power operations and device management.

Getting Started with the HTML Console SDK

After downloading the SDK, you can see how it works with a simple example. You add Javascript code to an HTML file and start a remote console in a Web browser.

Download and Install

The SDK is available in the Downloads section of the VMware {Code} landing page for the HTML Console SDK. You can use your My VMware account or Partner login to authenticate for download.

The SDK is delivered as a ZIP file. The unzipped SDK includes the WebMKS Javascript program with CSS and image folders.

```
SDK:

WebMKS_SDK_2.x.0.zip  css  img  wmks.min.js


SDK/css:

extended-keypad.css  main-ui.css  trackpad.css  wmks-all.css


SDK/img:

touch_sprite.png  touch_sprite_feedback.png
```

The `wmks.min.js` program has been minimized so it is not human readable. You can find various Web sites to make the Javascript readable, such as jsbeautifier.com, unminify.com, and the Javascript beautifier.

The WebMKS program requires jQuery, a popular Javascript library, and jQueryUI, a related library for user interfaces. Several versions of these libraries are available on public Web sites, such as jquery.com and ajax.googleapis.com. For secure environments you can download these libraries to run in a datacenter without Internet access.

Because the WebMKS program is written in Javascript, it requires a modern browser with Javascript support. Recent versions of Google Chrome, Mozilla Firefox, Apple Safari, and Microsoft Edge all qualify. If Javascript is enabled, the jQuery and jQueryUI libraries can run.

To see how it works, place the WebMKS program on an HTML page as a script, after loading the jQuery and jQueryUI libraries.

Steps to Start a Console

To create an HTML console, load the WebMKS program as a script into an HTML page, after loading the jQuery and jQueryUI libraries.

- Your machine must have connectivity to vCenter Server and an ESXi host at a known IP address or FQDN.
- Your machine should have a local HTTP server. Two alternatives are Lighttpd and XAMPP.
- You need locate and possibly download the jQuery and jQueryUI libraries.

1. Download the HTML Console SDK from code.vmware.com/sdk/html-console.
   The WebMKS_*.zip file contains the SDK components.
2. Unzip the archive.
   The `wmks.min.js` program appears, along with `css` and `img` folders.
3. Create a console display page using HTML from the next section.
4. If lacking security certificates, permit unsafe connections to vCenter Server and ESXi host.
5. Get a `webmks` ticket using one of the different methods at the end of this chapter.
6. On the `wmks.connect` line, fill in the IP address of the ESXi host, and the VM's `webmks` ticket.

When you open the HTML file in a browser, the VM console should appear.

Sample Javascript Code

A short Javascript application can create a remote HTML Console.

The sample HTML file starts with normal items including a window title. At the beginning of the body it calls `css/wmks-all.css`, which is a compendium of other CSS format files for the keyboard, trackpad, and screen UI. It then calls Web versions of the jQuery and jQueryUI libraries, and the `wmks.min.js` program delivered in the SDK.

The HTML console is placed on a Web page by the `wmksContainer` object, and registered for keyboard and mouse events. After setup, the connect call is made to a Web service secure (`wss`) URL. You will have to supply this URL manually; it is not automated in the sample code.

Example: HTML file to display remote console.

```
<!DOCTYPE html PUBLIC"-//W3C//DTD XHTML 1.0 Strict//EN"

  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

  <meta http-equiv="content-type" content="text/html; charset=utf-8" />

  <title>Console</title>

</head>

<body>

<link rel="stylesheet" type="text/css" href="css/wmks-all.css" />

<script type="text/javascript" src="http://code.jquery.com/jquery-3.4.1.min.js"></script>

<script type="text/javascript" src="http://code.jquery.com/ui/1.12.0/jquery-ui.min.js"></
script>

<script type="text/javascript" src="wmks.min.js"></script>

<div id="wmksContainer" style="position:absolute;width:100%;height:100%"></div>

<script>

var wmks = WMKS.createWMKS("wmksContainer",{})

  .register(WMKS.CONST.Events.CONNECTION_STATE_CHANGE, function(event,data){

    if(data.state == WMKS.CONST.ConnectionState.CONNECTED){

      console.log("connection state change : connected");}

  });

wmks.connect("wss://ESXi.host.IP.Address:443/ticket/webmksTicket");

</script>

</body>
```

```
</html>
```

This HTML file should be on a machine with Web service, for instance XAMPP or Lighttpd.

Replace *ESXi.host.IP.Address* with the IP address or fully-qualified hostname of the ESXi host where the virtual machine resides. Your browser must have accepted the security certificate from this ESXi host.

Acquire MKS Ticket with a Browser

To get the *webmksTicket*, find the vCenter Server that manages the ESXi host. You can get a ticket using the vSphere API (see Acquire MKS Ticket with vSphere API) or with the managed object browser (MOB).

To get a ticket with the MOB:

1. In a browser go to `https://vCenter.Server.IP.Address/mob`
2. Provide your login credentials for Administrator@vsphere.local or another SSO account
3. Click **content** > **rootFolder** (group-dl) > **Datacenter** > **vmFolder**
4. Click the VM number of the virtual machine you want to connect to
5. Under Methods, click **AcquireTicket**
6. Type `webmks` in the value field and click `Invoke Method`
7. Click to show. Copy and paste the ticket string and replace *webmksTicket*

You can also get a *webmksTicket* from the ESXi host, but the MOB must be enabled on the ESXi host, as described in KB 2108405, https://kb.vmware.com/kb/2108405.

Once you know the managed object ID (`moid`) of the virtual machine, you can quickly get another ticket at the following URL. Replace *NN* with the VM number:

```
https://vCenter.Server.IP.Address/mob/?moid=vm-NN&method=acquireTicket
```

If you have Windows PowerShell with PowerCLI installed, you can get a ticket as follows:

```
PowerCLI C:\> $vm.ExtensionData.AcquireTicket("webmks")
```

Acquire MKS Ticket with vSphere API

You can get the *webmksTicket* using a programming language that the Web Services SDK supports, such as Java or C#.

The `acquireTicket` method creates and returns a one-time credential used to establish a connection to a virtual machine. The `webmks` ticket type verifies a remote mouse-keyboard-screen connection. It requires **VirtualMachine.Interact.ConsoleInteract** privilege.

The following Java code returns an MKS ticket, given the managed object reference of a virtual machine and ticket type `webmks`.

```
public

VirtualMachineTicket acquireTicket(final ManagedObjectReference vmMoRef, String
ticketType)

throws Exception {

   VirtualMachineTicket vmTicket = null;

   vmTicket = super.getPortType().acquireTicket(vmMor, ticketType);
```

```
   return vmTicket;
}
```

To acquire the MKS ticket with vCloud Director, you can use a REST API call such as:

```
POST https://vcloud.example.com/api/vApp/vm-NN/screen/action/acquireMksTicket
```

Run the HTML Console Sample Code

The sample HTML code should reside on a machine with an HTTP server, in a directory prescribed for Web service. Communicating machines must have accepted security certificates from each other. To use the HTML console, navigate to the location of the sample code.

Once the remote console starts, log in to the virtual machine on its ESXi host, if a login session was not already started.

When you are finished with the HTML Console, you can disconnect and destroy the WMKS component. Both happen automatically when the HTML sample code page is closed.

The Create WebMKS Method and Options

Create WebMKS or `createWMKS` is the first method to call when using the HTML Console SDK. This method generates the console widget, which displays the remote screen, and returns the WMKS core object. The core object which is later associated with a in the document object model (DOM). After connecting to a virtual machine, your application can use the available APIs to perform operations with the remote console.

createMKS Parameters

The `createWMKS` method takes two parameters and returns the WMKS core object, used in subsequent operations. The first parameter is the name of a element. The second parameter is a possibly empty list of options.

id

Type: string. The ID of a element that is the container of the canvas used to show remote screens.

options

Type: JSON object. Options affect WMKS behavior, as documented here. If not specified, option defaults are used.

In the following example, this method creates a WMKS core object named "container" with all default options, and returns the WMKS core object in variable `wmks`.

```
var wmks = WMKS.createWMKS("container",{});
```

createMKS Options

Multiple options control the behavior of WMKS.

Option	Description	Type	Notes
audioEncodeType	Indicates the type of audio encoding method being used. Possible values for WMKS.CONST.AudioEncodeType are: `vorbis`, `opus`, or `aac`.	enum	

Table continued on next page

Continued from previous page

Option	Description	Type	Notes
changeResolution	If true, WMKS sends a change resolution request to the connected VM. The requested resolution matches the container size. If the request fails, use the rescale and position operations to change the resolution manually.	Boolean	Default is true.
enableUint8Utf8	If true, enables the `uint8utf8` protocol for projects that do not support the binary protocol.	Boolean	Default is false.
fixANSIEquivalentKeys	If true, enables translation of non-ANSI US keyboard layouts to ANSI US keyboard layout equivalents. Tries to match keys on the international keyboard to keys in different locations or with different Shift status on the US keyboard.	Boolean	Default is false.
ignoredRawKeyCodes	Ignores the keycodes. Do not send keycodes to the server.	array	Default is empty.
keyboardLayoutId	Provides different language keyboard setups for the Guest OS. Remote desktop and local desktop keyboard layouts must be the same. For international mapping, WMKS supports vScancode. This option does not support mobile devices. See keyBoardLayoutId Details	string	Default value is en-US.
position	Indicates where the remote screen should appear in the container. Values are either center or top left.	enum	Default value is WMKS.CONST.Position.CENTER
rescale	Indicates whether to rescale the remote screen to fit the container size.	Boolean	Default is true.
retryConnectionInterval	The interval in milliseconds before attempting to reconnect the Web client and server after a failed attempt. If less than 0,	integer	Default value is -1.

Table continued on next page

Continued from previous page

Option	Description	Type	Notes
	WMKS does not attempt to create the connection again.		
reverseScrollY	If true, sends the opposite value of the mouse when to the connected VM. This is for touch devices that scroll in the opposite direction.	Boolean	Default is false.
sendProperMouseWheelDeltas	If true, actual mouse wheel event delta values are sent from the browser to the server. If unspecified, normalized event deltas values are either: -1, 0, or 1.	Boolean	Default is false.
useNativePixels	Enables the use of native pixel sizes on the device. For example, on the iPhone 4+ or iPad 3+ devices, the true setting enables Retina mode, providing more screen space for the guest and making everything appear smaller.	Boolean	Default is false.
useUnicodeKeyboardinput	If true, WMKS attempts to send Unicode messages from the user to the server. If unspecified, WMKS sends messages using either Unicode or keyboard scan codes.	Boolean	Default is false.
useVNCHandshake	Enables a standard VNC handshake. Implement this option when the endpoint uses standard VNC authentication. Set to false if connecting to a proxy that uses `authd` for authentication and does not perform a VNC handshake.	Boolean	Default is true.
VCDProxyHandshakeVmxPath	The string passed by the VNC protocol. If a connection request for a VMX path is received, the VNC protocol responds with the VCDProxyHandshakeVmxPath when connecting to vCloud Director.	string	Default value is null.

keyBoardLayoutId Details

The keyBoardLayoutId option supports the following browsers:

- For Windows: Internet Explorer, Firefox, and Chrome.

- For Mac OS X: Chrome and Safari.

To support different languages, both the local machine with browser and the remote virtual machine must have and be set to the correct locale. For example, when a user chooses German in the keyboard select list, both the local machine and the remote VM must be running a German input method editor (IME).

So users can select the language they prefer, you provide a selection list in HTML.

```
<select id="selectLanguage">
   <option value="en-US">English</option>
   <option value="ja-JP_106/109">Japanese</option>
   <option value="de-DE">German</option>
   <option value="it-IT">Italian</option>
   <option value="es-ES">Spanish</option>
   <option value="pt-PT">Portuguese</option>
   <option value="fr-FR">French</option>
   <option value="fr-CH">Swiss-French</option>
   <option value="de-CH">Swiss-German</option>
</select>
```

Then call the `setOption` API in a Javascript file as in the following example.

```
$('#selectLanguage').change(function(){
    if(!wmks) return;
    var keyboardLayoutId = $(this).find(":selected").val();
    wmks.setOption('keyboardLayoutId',keyboardLayoutId);
 });
```

Handling Events with the HTML Console

The HTML Console generates events when the state of the currently connected VM changes, or in response to messages from the currently connected VM.

Event Binding in jQuery

As stated in the introduction, the HTML Console SDK is based on jQuery. HTML Console applications can use the jQuery `.on()` method to associate screen events with Javascript actions.

Since version 1.7, jQuery provides the .on() method for responding to an event on the selected elements. This is called an event binding. The jQuery model works with events causing changes to elements in the HTML document object model (DOM).

```
        Event Handler ------------------> Change element in the DOM
```

```
$(object).on('event', function() {
   ...                                          $("button".attr(...)
});
```

For example, an HTML Console application can implement screen resize as follows:

```
$(window).on('resize', function(e) {
   updateSize();
   updateScreen();
});
```

Setting the Event Handler

The HTML Console SDK associates a jQuery `.on()` event with a WMKS event handler.

The WMKS object uses the HTML Console SDK `connect()` method to set up a WebSocket URL and grab the remote console of a VM. To receive event notifications from the VM, you then call the HTML Console SDK `register()` method to set up the event handler. See Event Handler API Methods.

The following table shows the events defined by `WMKS.CONST.Events`.

Event Handlers

Each event handler processes one type of event with specific data.

event
: jQuery event information received

data
: data collected about the event

Event	Description	Data Collected
audio	Generated when an audio message is received from the server.	Values: • sampleRate • numChannels • containerSize • sampleSize • length • audioTimestampLo • audioTimestampHi • frameTimestampLo • frameTimestampHi • flags • data
connectionstatechange	Generated in response to a change in the connection state of the VM.	ConnectionState values: • connecting

Table continued on next page

Continued from previous page

Event	Description	Data Collected
		• connected • disconnecting If state is WMKS.CONST.ConnectionState.CONNECTING, data collected includes: `vvc` and `vvcSession`. If state is WMKS.CONST.ConnectionState.DISCONNECTED, data collected includes: `reason` and `code`.
copy	Generated when the server sends a cut text (copy) event.	Type: string. Value of the copied string.
error	Generated when an error occurs.	ErrorType values: • authenticationfailed • websocketerror • protocolerror
fullscreenchange	Generated when the WMKS console enters or exits full screen mode.	Type: Boolean with values: • True: entering full screen. • False: exiting full screen.
heartbeat	Once a connection is established, the server generates heartbeat events.	Type: integer with value of the heartbeat interval in seconds. The client listens for the heartbeat. If two intervals pass without a heartbeat, the connection is dead.
keyboardledschanged	Generated when the keyboard LED lock state changes on the remote VM.	Type: integer with values: • 1: scroll lock • 2: num lock • 4: caps lock
screensizechange	Generated in response to changes in the screen size of the connected VM.	Type: integer with values: • pixel width of the connected VM console • pixel height of the connected VM console
toggle	Generated when an input device is shown or hidden.	InputDeviceType values: • KEYBOARD • EXTENDED_KEYPAD • TRACKPAD Boolean type values: • True: shown • False: hidden

Copy from Remote Desktop

This section presents sample Javascript code to implement copy-paste from a remote virtual machine to the local client console.

Prerequisite: Enable clipboard copy from the remote to local client.

After copy from the remote desktop, clipboard data becomes available after the mouse cursor is no longer focused on the remote desktop.

To implement remote copy, add this Javascript code as a script to existing HTML console code:

```
<script>
   $(function() {
      var container = $("#container");
      var wmks = WMKS.createWMKS("container", {});
      wmks.register(WMKS.CONST.Events.CONNECTION_STATE_CHANGE, function(evt, data) {
         switch (data.state) {
            case WMKS.CONST.ConnectionState.CONNECTING:
               console.log("The console is connecting");
               break;
            case WMKS.CONST.ConnectionState.CONNECTED:
               console.log("The console has been connected");
               // need to send grab first, bind on the canvas element
               $("#mainCanvas").focus(function() {
                  wmks.grab();
               });
               $("#mainCanvas").blur(function() {
                  wmks.ungrab();
               });
               break;
            case WMKS.CONST.ConnectionState.DISCONNECTED:
               console.log("The console has been disconnected");
               break;
         }
      });
      wmks.register(WMKS.CONST.Events.ERROR, function(evt, data) {
         console.log("Error: " + data.errorType);
      });
      wmks.register(WMKS.CONST.Events.COPY, function(evt, data) {
         // here is the remote clipboard data:
         console.log(evt, data);
```

```
      });
      wmks.connect("wss://testurl");
   });
</script>
```

Make sure the remote desktop has already enabled clipboard feature. Before ungrab, a grab event is necessary. You can get remote clipboard data from the server after sending ungrab.

HTML Console SDK Methods

The WMKS core object uses HTML Console SDK methods to connect to and perform operations on a remote VM console. HTML Console SDK methods below are grouped by related function.

Each method definition includes required parameters, return values, and an example call. In example calls, the WMKS core object is named `wmks`. WMKS Constants provides descriptions and values of parameter constants.

General API Methods

General purpose API methods provide information about WMKS. These methods can be called at any time before or during connection to a target VM.

getVersion()

Retrieves the current version number of the HTML Console SDK.

Parameters

None

Return Value

Type: string. Contains the full version number of the HTML Console SDK.

Example Call

```
var version = wmks.getVersion();
```

getConnectionState()

Retrieves the current connection state.

Parameters

None

Return Value

Type: string. Any constant value for WMKS.CONST.ConnectionState. See WMKS Constants.

Example Call

```
var version = wmks.getConnectionState();
```

Lifecycle API Methods

The WMKS core object uses lifecycle API methods to manage the connection with the target VM.

connect()

Connects the WMKS to a remote virtual machine using the WebSocket URL and sets up the UI.

Parameters

Type: string. WebSocket URL in format:

```
<ws|wss>://<host:port>/<path>/?<authentication info>
```

Return Value

None

Example Call

```
wmks.connect("ws://localhost:8080");
```

disconnect()

Disconnects from the remote virtual machine and exits the UI.

Parameters

None

Return Value

None

Example Call

```
wmks.disconnect();
```

destroy()

Terminates any active connection with the VM and removes the widget from the associated element. Always call this method before removing a WMKS element from the HTML document object model (DOM).

Parameters

None

Return Value

None

Example Call

```
wmks.destroy();
```

Display API Methods

The WMKS core object uses display API methods to control the remote screen size.

setRemoteScreenSize()

Sends a request to set the screen resolution of the currently connected VM console. If the requested width and height parameters are larger than the allocated size of the WMKS widget, the sizing is normalized.

Parameter1

Type: integer. Width of the screen in pixels.

Parameter2

Type: integer. Height of the screen in pixels.

Return Value

None

Example Call

```
wmks.setRemoteScreenSize(800,600);
```

getRemoteScreenSize()

Retrieves the screen width and height in pixels.

Parameter

None

Return Value

Object with format {width:*w*, height:*h* }

Example Call

```
var size = wmks.getRemoteScreenSize;
```

updateScreen()

Changes the resolution or rescales the remote screen to match the container size. Behavior depends on settings for changeResolution, rescale, and position options described in createMKS Options.

- If changeResolution is true, the remote screen resolution changes to match the container size.
- If changeResolution is false, check the rescale option. If true, the remote screen rescales to match the container size.
- If both changeResolution and rescale are false, check the position option. Depending on the value, the remote screen appears in the container at either the center or top left.

Parameter

None

Return Value

None

Example Call

```
wmks.updateScreen();
```

Full Screen API Methods

The WMKS core object uses full screen API methods to control full screen mode in the browser.

canFullScreen()

Returns whether the browser is enabled for full screen mode. Due to security concerns with keyboard input in full screen mode, Safari is not enabled for full screen mode.

Parameters

None

Return Value

Type: Boolean. True indicates that the browser is enabled for full screen mode.

Example Call

```
wmks.canFullScreen();
```

isFullScreen()

Returns whether the browser is in full screen mode.

Parameters

None

Return Value

Type: Boolean. True indicates that the browser is in full screen mode.

Example Call

```
wmks.isFullScreen();
```

enterFullScreen()

Forces the browser to enter full screen mode. In full screen mode, only the remote screen appears.

Parameters

None

Return Value

None

Example Call

```
wmks.enterFullScreen();
```

exitFullScreen()

Forces the browser to exit full screen mode.

Parameters

None

Return Value

None

Example Call

```
wmks.exitFullScreen();
```

Input API Methods

The WMKS core object uses input API methods to send keyboard input to the server or VM.

sendInputString()

Sends a string as keyboard input to the server.

Parameters

Type: string.

Return Value

None

Example Call

```
wmks.sendInputString("test");
```

sendKeyCodes()

Sends a series of special key codes to the VM. The input is an array of key codes and the method sends keydown events for each key code in the order listed. This method sends keyup events for each in reverse order. Use this method to send key combinations such as Alt+Tab.

For a list of key codes, see Javascript Key Codes.

Parameters

Type: array. Each element in the array can be a keycode or a Unicode character, where keycode is for non-printable keys, and Unicode is a negative number for printable characters. With Unicode, a negative integer such as -118 represents the Latin small letter v.

Return Value

None

Example Call

```
wmks.sendKeyCodes([18,9]) ; // Alt + Tab
```

sendCAD()

Sends a Control+Alt+Delete key sequence to the connected VM.

Parameters

None

Return Value

None

Example Call

```
wmks.sendCAD();
```

Mobile API Methods

The WMKS core object uses mobile input methods to control the input and display on a mobile device. The input device types are keyboard, extended keyboard, and trackpad.

enableInputDevice()

Enables and initializes the input device type on a mobile device.

Parameters

Any constant value for WMKS.CONST.InputDeviceType. See WMKS Constants.

Return Value

None

Example Call

```
wmks.enableInputDevice(WMKS.CONST.InputDeviceType.KEYBOARD);
```

disableInputDevice()

Disables and terminates the input device type on a mobile device.

Parameters

Any constant value for WMKS.CONST.InputDeviceType. See WMKS Constants.

Return Value

None

Example Call

```
wmks.disableInputDevice(WMKS.CONST.InputDeviceType.KEYBOARD);
```

showKeyboard()

Shows the keyboard on a mobile device.

Parameters

None

Return Value

None

Example Call

```
wmks.showKeyboard();
```

hideKeyboard()

Hides the keyboard on a mobile device.

Parameters

None

Return Value

None

Example Call

```
wmks.hideKeyboard();
```

toggleExtendedKeypad()

Depending on availability, shows or hides the extended keyboard on a mobile device such as the iPhone 6s or 6.

Parameters

A map such as minToggleTime(50) or {minToggleTime: 50}. If called frequently with the time between subsequent calls less than the minToggleTime in milliseconds, the call is not executed.

Return Value

None

Example Call

```
wmks.toggleExtendedKeypad({minToggleTime: 50});
```

toggleTrackpad()

Depending on current visibility, shows or hides the trackpad on a mobile device.

Parameters

A map such as minToggleTime(50) or {minToggleTime: 50}. If called frequently with the time between subsequent calls less than the minToggleTime in milliseconds, the call is not executed.

Return Value

None

Example Call

```
wmks.toggleTrackpad({minToggleTime: 50});
```

toggleRelativePad()

Depending on current visibility, shows or hides the relative trackpad on a mobile device.

WebMKS adds a toggleRelativeMouse button in the UI and binds it to the toggleRelativePad() API. If the button is clicked, the relative trackpad appears and sends relative mouse events to the remote guest. Use this method when the remote guest OS does not install VMware Tools and cannot accept an absolute mouse event.

To display the relative trackpad, the SDK includes a wmks-all.css file provided in the css and img folders.

Parameters

A map such as minToggleTime(50) or {minToggleTime: 50}. If called frequently with the time between subsequent calls less than the minToggleTime in milliseconds, the call is not executed.

Return Value

None

Example Call

```
wmks.toggleRelativePad({minToggleTime: 50});
```

setOption API Method

The WMKS core object uses the `setOption` method to change option values for `createWMKS`.

Of the options listed in createMKS Options, this method supports:

- rescale
- position
- changeResolution
- useNativePixels
- reverseScrollY
- fixANSIEquivalentKeys
- sendProperMouseWheelDeltas
- keyboardLayoutId

Parameter1

Type: string. optionName

Parameter2

Type: Boolean. optionValue

Return Value

None

Example Call

```
wmks.setOption("changeResolution",false);
```

Event Handler API Methods

Event handler API methods add and remove handlers for WMKS events.

See Handling Events with the HTML Console.

register()

Register the event handler for the WMKS.

Parameter1

Any constant value for WebMKS.CONST.Events. See WMKS Constants.

Parameter2

eventHandler (javascript callback function)

Return Value

None

Example Call

```
var connectionStateHandler = function(event,data){};

wmks.register(WebMKS.CONST.Events.CONNECTION_STATE_CHANGE,

              connectionStateHandler);
```

unregister()

Unregister the event handler for the WMKS.

- If called without parameters, it removes all event handlers.
- If called with one parameter, it removes event handlers for the given event name.
- If no such event was registered, it has no effect.

Parameter1

Any constant value for WebMKS.Events. See WMKS Constants.

Parameter2

eventHandler (javascriptCallbackFunction)

Return Value

None

Example Call

```
wmks.unregister(WebMKS.CONST.Events.CONNECTION_STATE_CHANGE,

                connectionStateHandler);
```

WMKS Constants

HTML Console SDK methods use the following constants for parameter input and return values. Each description includes an example that lists the constant values.

AudioEncodeType

Types of audio encode methods.

```
AudioEncodeType: {

        VORBIS: "vorbis",

        OPUS: "opus",

        AAC: "aac"

},
```

ConnectionState

Describes the state when attempting to connect to a remote VM.

```
ConnectionState: {

        CONNECTING: "connecting",

        CONNECTED: "connected",

        DISCONNECTED: "disconnected"

},
```

ErrorType

Errors that can occur when WMKS is connected to a remote VM.

```
ErrorType: {

        AUTHENTICATION_FAILED: "authenticationfailed",

        WEBSOCKET_ERROR: "websocketerror",

        PROTOCOL_ERROR: "protocolerror"

},
```

Events

Events that WMKS can trigger.

```
Events: {

        CONNECTION_STATE_CHANGE: "connectionstatechange",

        REMOTE_SCREEN_SIZE_CHANGE: "screensizechange",

        FULL_SCREEN_CHANGE: "fullscreenchange",

        ERROR: "error",

        KEYBOARD_LEDS_CHANGE: "keyboardledschanged",

        HEARTBEAT: "heartbeat",

        AUDIO: "audio",

        COPY: "copy",

        TOGGLE: "toggle"

},
```

InputDeviceType

Specifies the input device when viewing VM consoles on mobile devices.

```
InputDeviceType: {

        KEYBOARD: 0,

        EXTENDED_KEYBOARD: 1,

        TRACKPAD: 2

}
```

Position

WMKS displays the remote screen of the VM in equal proportions. After rescale, the remote screen size might not match the container size. Position provides two options to position the screen in the container.

```
Position: {

        CENTER:   0,

        LEFT_TOP: 1

},
```

Javascript Key Codes

Javascript assigns a numeric code to every key on a keyboard, including ones that control programs and interfaces.

Alphanumeric Keys

Key codes for alphanumeric and function keys are consecutively numbered. Alphanumeric and symbol keys also have Unicode values, expressed as a negative number with Unicode decimal code.

- The 0 to 9 keys are codes 48 to 57
- The A to Z keys are codes 65 to 90
- Keys on the number pad are codes 96 to 105
- Function keys F1 to F12 are 112 to 123.

For program clarity it is better to use Unicode negative decimal codes than alphanumeric key codes.

Table of Key Codes

These are keys that control the user interface and operating system. Not all keys appear on all keyboards.

Key	Code	Key	Code
Backspace	8	Print Screen	none
Tab	9	Insert	45
Enter	13	Delete	46
Shift	16	Windows Left	91
Control	17	Windows Right	92
Alt	18	Select	93
Break	19	Multiply	106
Caps Lock	20	Add	107
Escape	27	Separator	109
Space Bar	32	Subtract	109
Page Up	33	Decimal Point	110
Page Down	34	Divide	111
End	35	Num Lock	144
Home	36	Scroll Lock	145
Left Arrow	37		
Up Arrow	38		
Right Arrow	39		
Down Arrow	40		

Getting Started with ESXCLI

Getting Started with ESXCLI gets you started with ESXi Shell commands and ESXCLI commands.

Intended Audience

This book is for experienced system administrators who are familiar with vSphere administration tasks.

Overview of ESXCLI

You can use ESXCLI to manage many aspects of an ESXi host by running ESXCLI commands remotely or in the ESXi Shell.

You can run ESXCLI commands against a vCenter Server system and target the host indirectly. Running against vCenter Server systems by using the
`--vihost`
option is required if the host is in lockdown mode. You can also run ESXCLI commands in the local ESXi Shell to manage that host.

Using ESXCLI for Host Management

You can manage many aspects of an ESXi host by using commands from the ESXCLI command set. You can run ESXCLI commands remotely, or run them in the ESXi Shell in troubleshooting situations.

You can also run ESXCLI commands from the PowerCLI shell by using the **`Get-EsxCli`** cmdlet. See the *PowerCLI User's Guide* and the *PowerCLI Cmdlet Reference*.

The set of ESXCLI commands that are available on a host depends on the host configuration. The *ESXCLI Reference* lists help information for all ESXCLI commands. You can run `esxcli --server <MyESXi> --help` before you run a command on a host to make sure that the command is defined on the host that you are targeting.

ESXCLI Syntax

Each ESXCLI command uses the same syntax.

The following is the standard syntax structure of an ESXCLI command.

```
esxcli [connection options] <namespace> [<namespace> ...] <cmd> [cmd options]
```

Syntax Element	Description
`connection options`	Predefined options for connection information such as target host, user name, and so on. See Running ESXCLI Host Management Commands. Not required when you run the command in the ESXi Shell. If the target server is a vCenter Server system, specify the target ESXi host before any ESXCLI namespaces, commands, and supported options. Many ESXCLI commands generate output that you might want to use in your application. You can run **`esxcli`** with the `--formatter`

Table continued on next page

Continued from previous page

Syntax Element	Description
	dispatcher option and send the resulting output as input to a parser. **IMPORTANT** ESXCLI expects a trust relationship between the target host and the system on which you run the command. You can establish this relationship in one of the following ways. • Use the `--cacertsfile` option or *`VI_CACERTFILE`* variable. • Store the thumbprint in the session file. • Specify the thumbprint with the `--thumbprint` option or *`VI_THUMBPRINT`* variable. You can pass in the thumbprint that is returned in the error if you trust the host that you are targeting. See Trust Relationship Requirement for ESXCLI Commands for an example.
`namespace`	Groups ESXCLI commands. Nested namespaces are supported.

`command`	Reports on or modifies the state of the system.
	
	The following examples show how you can use this element.
	
	`esxcli --server myESXi --username user1 --password 'my_password' storage nfs list`
	
	`esxcli --server myVCServer --username user1 --password 'my_pwd' --vihost myESXi.mycompany.com storage nfs list`

`options`	Many commands support one or more of the options displayed in the help or the ESXCLI reference. For some commands, multiple option values, separated by spaces, are possible. The following example shows how you can use this element. `esxcli system module parameters set -m <module> -p "a=1 b=1 c=1"`

Running ESXCLI Commands

You can run an ESXCLI command in the ESXi Shell for troubleshooting purposes and remotely against a specific host or against a vCenter Server system.

You can install ESXCLI on one of the supported Windows, Linux, or macOS systems. Specify connection options to run commands against an ESXi host directly, or target a vCenter Server system and specify the ESXi host to run the command against. See Installing ESXCLI.

NOTE
A trust relationship must exist between the host from which you run ESXCLI commands and the target ESXi host or vCenter Server system. See Trust Relationship Requirement for ESXCLI Commands.

See Running ESXCLI Host Management Commands.

ESXCLI Command Support When Host and ESXCLI Versions Do Not Match

When you run an ESXCLI command, you must know the commands that are supported on the target host specified with `--server`.

The following examples demonstrate command support when versions do not match.

- If you run commands against ESXi 6.x hosts, ESXCLI 6.x commands are supported.
- If you run commands against ESXi 7.x hosts, ESXCLI 7.x commands are supported.
- If you run commands against ESXi 8.x hosts, ESXCLI 8.x commands are supported.

VMware partners might develop custom ESXCLI commands that you can run on hosts where the partner VIB is installed.

Run `esxcli --server <target> --help` for a list of namespaces supported on the target host. You can explore the namespaces for additional help.

Run `esxcli --server <target> esxcli command list` for a list of all commands supported on the target host.

Installing ESXCLI

You can install ESXCLI on a Linux, Microsoft Windows, or a macOS system.

Installing and Uninstalling ESXCLI

You can install and uninstall ESXCLI by running Python commands.

You must have a compatible version of Python installed on a supported operating system. The supported Python versions are listed in the *Release Notes*.

Install ESXCLI

You can install ESXCLI by downloading the installation package and running a Python command.

- Verify that there is no existing ESXCLI installation on the system.

1. Download the ESXCLI Python package.

   You can find a link to the package on the ESXCLI page, which you can find in the **Developer Tools** section of the **Resources** tab on the VMware Developer website.

2. Navigate to the folder where you downloaded the package and run the Python **`pip`** installer.

   ```
   $ pip install esxcli-8.0.0-XXXXXXXX.tgz
   ```

Uninstall ESXCLI

You can uninstall ESXCLI by running a Python command.

1. Run the uninstaller.

```
$ pip uninstall esxcli
```

Running Host Management Commands in the ESXi Shell

Usually, installing ESXCLI and running commands from a remote system, with one or more hosts as targets, is recommended. However, for maintenance and troubleshooting tasks you might prefer to run ESXCLI commands in the ESXi Shell or connect to the ESXi Shell with SSH.

To run commands, you must first establish access to the ESXi Shell.

ESXi Shell Access with the Direct Console

An ESXi system includes a Direct Console User Interface (DCUI) that you can use to start and stop the system and to perform a limited set of maintenance and troubleshooting tasks.

You can use the direct console to access the ESXi Shell, which is disabled by default. You can enable the ESXi Shell in the direct console or by using the vSphere Client. You can enable local shell access or remote shell access.

- With local shell access, you can log in to the shell directly from the Direct Console. See Enabling Local Access.
- With remote shell (SSH) access you can connect to the host by using a shell such as PuTTY, specify a user name and password, and run commands in the shell. See Remote Access with SSH.

The ESXi Shell includes all ESXCLI commands, a set of deprecated **`esxcfg-`** commands, and a set of commands for troubleshooting and remediation.

> **IMPORTANT**
> All ESXCLI commands that are available in the ESXi Shell are also included in the ESXCLI package.
> You can install the ESXCLI package on a supported Windows or Linux system, and run commands against your ESXi hosts. Run commands in the ESXi Shell directly or through SSH only in troubleshooting situations.

Enabling Local ESXi Shell Access

You can enable the ESXi Shell from the direct console or the vSphere Client.

Enable the ESXi Shell in the Direct Console

If you have access to the Direct Console User Interface, you can enable the ESXi Shell from there.

1. At the direct console of the ESXi host, press F2 and provide credentials when prompted.
2. Scroll to **Troubleshooting Options** and press Enter.
3. Select **Enable ESXi Shell** and press Enter.
   On the left, **Enable ESXi Shell** changes to **Disable ESXi Shell**. On the right, **ESXi Shell is Disabled** changes to **ESXi Shell is Enabled**.
4. Press Esc until you return to the main direct console screen.

After you enable the ESXi Shell, you can use it from that monitor or through a serial port.

Enable the ESXi Shell from the vSphere Client

If you do not have access to the Direct Console User Interface, you can enable the ESXi Shell from the vSphere Client.

1. In the vSphere Client, select the host
2. On the **Configure** tab, select **System › Services**.

3. Select **ESXi Shell**.
   – To temporarily start or stop the service, click the **Start** or **Stop** button.
   – To change the startup policy, click the **Edit Startup Policy** button.

After you enable the ESXi Shell, you can use it through a serial port.

ESXi Shell Timeout

The ESXi Shell supports a timeout for ESXi Shell availability and a timeout for idle ESXi Shell sessions.

- Availability timeout - The availability timeout setting is the amount of time that can elapse before you must log in after the ESXi Shell is enabled. After the timeout period, the service is disabled and users are not allowed to log in.
- Idle timeout - If a user enables the ESXi Shell on a host, but forgets to log out of the session, the idle session remains connected indefinitely.

You can set both timeout values from the Direct Console User Interface or from the vSphere Client. See the *vSphere Security* document for detailed instructions.

Use the Local ESXi Shell

After you enable the ESXi Shell in the direct console, you can use it from the main direct console screen or remotely through a serial port.

1. At the main direct console screen, press Alt-F1 to open a virtual console window to the host.
2. Provide credentials when prompted.

   When you enter the password, characters are not displayed on the console.
3. Enter shell commands to perform management tasks.
4. To log out, type `exit` in the shell.
5. To return to the direct console, press Alt-F2.

See the *ESXi Installation and Setup* documentation for information on serial port setup.

Remote ESXi Shell Access with SSH

If SSH connections are enabled for your ESXi host, you can run shell commands by using a Secure Shell client such as SSH or PuTTY.

Enable SSH Access in the Direct Console

If running commands remotely is disabled on an ESXi host, you cannot log in to the host by using a remote shell. You can enable running commands remotely from the direct console or from the vSphere Client.

1. At the direct console of the ESXi host, press F2 and provide credentials when prompted.
2. Scroll to **Troubleshooting Options** and press Enter.
3. Choose **Enable SSH** and press Enter.
   On the left, **Enable SSH** changes to **Disable SSH**. On the right, **SSH is Disabled** changes to **SSH is Enabled**.
4. Press Esc until you return to the main direct console menu.

After you have enabled SSH, you can log in to the ESXi Shell remotely and run ESXi Shell commands.

Enable SSH from the vSphere Client

If running commands remotely is disabled on an ESXi host, you cannot log in to the host by using a remote shell. You can enable running command remotely from the direct console or from the vSphere Client.

1. In the vSphere Client, select the host
2. On the **Configure** tab, select **System › Services**.
3. Select **SSH**.
   - To temporarily start or stop the service, click the **Start** or **Stop** button.
   - To change the startup policy, click the **Edit Startup Policy** button.

After you have enabled SSH, you can log in to the ESXi Shell remotely and run ESXi Shell commands.

Access the Remote ESXi Shell with SSH

If SSH is enabled on your ESXi host, you can run commands on that shell by using an SSH client.

1. Open an SSH client.
2. Specify the IP address or domain name of the ESXi host.

   Precise directions vary depending on the SSH client that you are using. See vendor documentation and support.
3. Provide credentials when prompted.

Lockdown Mode

To increase the security of your ESXi hosts, you can put them in lockdown mode.

In lockdown mode, all operations must be performed through vCenter Server. By default, only the vCenter Server system, represented by the vpxuser user, has authentication permissions. No other users can perform operations against a host in lockdown mode.

- In normal lockdown mode, you can add users to the `DCUI.Access` advanced option, which can access the Direct Console User Interface regardless of their privileges on the host. You can also use the vSphere Client to add Exception users, which can access the Direct Console User Interface if they have host management privileges.
- In strict lockdown mode, users cannot access the Direct Console User Interface. If vCenter Server becomes unavailable, the host can no longer be managed.

When a host is in normal or strict lockdown mode, you cannot run ESXCLI commands against the host directly. Instead, you target the vCenter Server system that manages the host with the `--server` option and specify the ESXi host with the `--vihost` option.

When you enable strict lockdown mode, the Direct Console User Interface service is disabled.

You can enable lockdown mode by using the Add Host wizard to add a host to vCenter Server, by using the vSphere Client to manage a host, or by using the Direct Console User Interface (DCUI).

See the *`vSphere Security`* documentation for details on lockdown mode.

Run an ESXCLI Command in the ESXi Shell

You can run ESXCLI commands in the ESXi Shell unless they are marked as internal in the online help.

Verify that the ESXi Shell is enabled.

The ESXi Shell is disabled by default. You must enable the ESXi Shell before you can run commands in the shell. See Access with the Direct Console.

1. Log in to the shell.
2. Run the command.
   For example, to list NFS storage devices, run the following command.
   ```
   esxcli storage nfs list
   ```

You can use `--help` at any level of `esxcli` for help on available namespaces, commands, or options.

Running ESXCLI Host Management Commands

You can run ESXCLI host management commands from the command line of the system where you installed the package and from scripts.

Host management commands require at a minimum access to the target server to run the commands on. Users must authenticate to the host, and can only perform tasks that they are authorized to perform.

> **IMPORTANT**
> If an ESXi system that you target is in lockdown mode, you cannot run ESXCLI commands against that system directly. You must target a vCenter Server system that manages the ESXi system and use the `--vihost` option to specify the ESXi target. See ESXCLI and Lockdown Mode.

Overview of Running ESXCLI Host Management Commands

You can run ESXCLI commands interactively or in scripts, and you can target the host directly or target a vCenter Server system that manages the host.

Targeting the Host Directly

You can target the host directly from an administration server on which you installed ESXCLI or by running scripts.

- Open a command prompt on a Linux or Windows system on which you installed ESXCLI. Enter commands into that command prompt, specifying connection options. See Authenticating Directly to the Host.
- Prepare scripts that contain ESXCLI commands. Then run the scripts from a system that has the ESXCLI package installed. See Using ESXCLI Commands in Scripts.

When you run commands against an ESXi host, you must be authenticated for that host.

Targeting a Host That is Managed by a vCenter Server System

When you target a host that is managed by a vCenter Server system, you can run commands in different ways.

- Specify the vCenter Single Sign-On service with the `--psc` option and, if multiple vCenter Server systems are associated with the vCenter Single Sign-On service, the vCenter Server system with the `--server` option. Specify also the host with the `--vihost` option.
- Specify the vCenter Server system with the `--server` option and the ESXi host with the `--vihost` option.
- Specify only the ESXi host with the `--vihost` option.

When you can authenticate to a vCenter Single Sign-On service or to a vCenter Server system, you can target all ESXi hosts that vCenter Server manages without additional authentication. See Authenticating Through and vCenter Single Sign-On.

Protecting Passwords

You can follow different password protection approaches depending on your environment setup.

> **CAUTION**
> If you specify passwords in plain text, you risk exposing the password to other users. The password might also become exposed in backup files. Do not provide plain-text passwords on production systems.

Follow one of the following approaches for protecting passwords.

- If you use a ESXCLI host management command interactively and do not specify a user name and password, you are prompted for them. The screen does not echo the password that you enter.
- For non-interactive use, you can create a session file by using the `save_session` option. See the *vSphere SDK for Perl Programming Guide*.
- Target a vCenter Server system and authenticate to vCenter Single Sign-On. You can save the corresponding session and use it for subsequent connections. See Authenticating Through and vCenter Single Sign-On.
- Use variables or configuration files.
- If you are running ESXCLI on a Windows system, you can use the `--passthroughauth` option. If the user who runs the command with that option is a known Active Directory user, no password is required.

With ESXCLI, you can run scripts against multiple target servers from the same administration server. You must have the correct privileges to perform the actions on each target, and you must authenticate to the target.

> **IMPORTANT**
> Administrators can place ESXi hosts in lockdown mode for enhanced security. By default, even the root user cannot run ESXCLI commands directly against ESXi hosts in lockdown mode. See ESXCLI and Lockdown Mode and the *vSphere Security* documentation.

Order of Precedence for ESXCLI Host Management Commands

When you run a ESXCLI host management command, authentication happens in order of precedence.

The order of precedence is described in the following table. This order of precedence always applies. That means, for example, that you cannot override an environment variable setting in a configuration file.

If you are authenticating through vCenter Single Sign-On, the order of precedence is preserved. For example, information you specify on the command line overrides information in an environment variable.

Authentication	Description	See
Command line	Password (`--password`), session file (`--sessionfile`), or configuration file (`--config`) specified on the command line.	*vSphere SDK for Perl Programming Guide*
Environment variable	Password specified in an environment variable.	Using Environment Variables
Configuration file	Password specified in a configuration file.	Using a Configuration File
Current account (Active Directory)	Current account information used to establish an SSPI connection. Available only on Windows.	Using the Microsoft Windows Security Support Provider Interface
Credential store	Password retrieved from the credential store.	*vSphere Web Services SDK Programming Guide* and *vSphere SDK for Perl Programming Guide*
Prompt the user for a password	Password is not echoed to screen.	

Authenticating Through vCenter Server and vCenter Single Sign-On

For all ESXi hosts that are managed by a vCenter Server system that is integrated with vCenter Single Sign-On, you can authenticate directly to the vCenter Server system, or you can authorize to vCenter Server through vCenter Single Sign-On.

The best practice is to authenticate through vCenter Single Sign-On.

You use the following options.

- `server`
  - Specifies the vCenter Server system that manages the host.
- `vihost`
  - Specifies the ESXi host.

Example

```
esxcli --server <vc_hostname_or_IP> --vihost <esxi_hostname_or_IP> --username <user_name>
--password <password> hardware clock get
```

Authenticating Directly to the Host

ESXCLI offers several options for authenticating directly to the host.

Using Environment Variables

How you use environment variables depends on the operating system that you are using.

On Linux, you can set environment variables in a Linux bash profile or on the command line by using a command like the following.

```
export VI_SERVER=<your_server_name_or_address>
```

On Windows, you can set environment variables in the Environment properties dialog box of the System control panel. For the current session, you can set environment variables at the command line by using a command like the following.

```
set VI_SERVER=<your_server_name_or_address>
```

IMPORTANT
Do not use escape characters in environment variables.

See Using ESXCLI Commands in Scripts for an environment variable example.

Using a Configuration File

You can use a text file that contains variable names and settings as a configuration file.

Variables corresponding to the options are shown in Connection Options for ESXCLI Host Management Command Execution.

CAUTION
Limit read access to a configuration file that contains user credentials.

Pass in the configuration file when you run ESXCLI commands, by using the following syntax.

```
<command> --config <my_saved_config> <option>
```

The following example illustrates how you can use the syntax.

```
esxcli --config <my_saved_config> network ip interface list
```

If you have multiple vCenter Server or ESXi systems and you administer each system individually, you can create multiple configuration files with different names. To run a command or a set of commands on a server, you pass in the `--config` option with the appropriate file name at the command line.

The following example illustrates the contents of a configuration file.

```
VI_PSC = XX.XXX.XXX.XX

VI_USERNAME = administrator@vsphere.local

VI_PASSWORD = admin_password

VI_PROTOCOL = https

VI_SERVER = my_vc
```

If you have set up your system to run this file, you can run scripts against the specified ESXi host afterwards.

Using Command-Line Options

You can pass in command-line options by using option name and option value pairs in most cases.

You can use long or short options. An equal sign between option name and option value is optional.

```
esxcli --server <vc_hostname_or_IP> --username <privileged_user> --password <password> --
vihost <esxi_hostname_or_IP> <namespace> [<namespace]...> <command> --<option_name=option_
value>
```

Some options, such as `--help`, have no value.

IMPORTANT
Enclose passwords and other text with special characters in quotation marks.
When running commands on Windows, use double quotes (`" "`). When running commands on Linux, use single quotes (`' '`) or a backslash (`\`) as an escape character.

The following examples connect to the server as user `snow-white` with password `dwarf$`.

Linux

```
esxcli --server <vc_hostname_or_IP> --username snow\-white --password dwarf\$ network ip
interface list
```

```
esxcli --server <vc_hostname_or_IP> --username snow\-white --password 'dwarf$' network ip
interface list
```

Windows

```
esxcli --server <vc_hostname_or_IP> --username "snow-white" --password "dwarf$" network ip
interface list
```

Using the Microsoft Windows Security Support Provider Interface

With the
`--passthroughauth`
option, which is available if you run ESXCLI commands from a Microsoft Windows system, you can use the Microsoft Windows Security Support Provider Interface (SSPI).

You can refer to the Microsoft Web site for detailed information on SSPI.

You can use
`--passthroughauth`
to establish a connection with a vCenter Server system. After the connection has been established, authentication for the vCenter Server system or any ESXi system that it manages is no longer required. Using
`--passthroughauth`
passes the credentials of the user who runs the command to the target vCenter Server system. No additional authentication is required if the user who runs the command is known by the computer from which you access the vCenter Server system and by the computer running the vCenter Server software.

If ESXCLI commands and the vCenter Server software run on the same computer, the user needs only a local account to run the command. If the ESXCLI command and the vCenter Server software run on different machines, the user who runs the command must have an account in a domain trusted by both machines.

SSPI supports several protocols. By default, it selects the Negotiate protocol, where client and server try to find a protocol that both support. You can use
`--passthroughauthpackage`
to explicitly specify a protocol that is supported by SSPI. Kerberos, the Windows standard for domain-level authentication, is used frequently. If the vCenter Server system is configured to accept only a specific protocol, specifying the protocol with
`--passthroughauthpackage`
might be required for successful authentication. If you use
`--passthroughauth`
, you do not have to specify authentication information by using other options.

```
esxcli --server <vc_hostname_or_IP> --passthroughauth --passthroughauthpackage "Kerberos"

--vihost <esxi_hostname_or_IP> network ip interface list
```

This example establishes a connection to a server that is set up to use SSPI. When a trusted user runs the command, the system calls the ESXCLI command with the
`--list`
option. The system does not prompt for a user name and password.

ESXCLI and Lockdown Mode

Lockdown mode can disable all direct root access to ESXi machines.

To make changes to ESXi systems in lockdown mode you must go through a vCenter Server system that manages the ESXi system. You can use the vSphere Client or ESXCLI commands that support the
`--vihost`
option.

If you have problems running a command on an ESXi host directly, without specifying a vCenter Server target, check whether lockdown mode is enabled on that host. See the *`vSphere Security`* documentation.

Trust Relationship Requirement for ESXCLI Commands

ESXCLI checks whether a trust relationship exists between the machine where you run the ESXCLI command and the ESXi host. An error results if the trust relationship does not exist.

Download and Install the vCenter Server Certificate

You can download the vCenter Server root certificate by using a Web browser and add it to the trusted certificates on the machine where you plan to run ESXCLI commands.

1. Enter the URL of the vCenter Server system into a Web browser.
2. Click the **Download trusted root CA certificates** link.
3. Verify that the extension of the downloaded file is `.zip`.

   The file is a ZIP file of all certificates in the TRUSTED_ROOTS store.

4. Extract the ZIP file.
   A certificates folder is extracted. The folder includes files with the extension `.0`, `.1`, and so on, which are certificates, and files with the extension `.r0`, `.r1`, and so on which are CRL files associated with the certificates.
5. Add the trusted root certificates to the list of trusted roots.

   The process differs depending on the platform that you are on.

You can now run ESXCLI commands against any host that is managed by the trusted vCenter Server system without supplying additional information if you specify the vCenter Server system in the
`--server`
option and the ESXi host in the
`--vihost`
option.

Using the --cacertsfile Option

Using a certificate to establish the trust relationship is the most secure option.

You can specify the certificate with the `--cacertsfile` parameter or the *`VI_CACERTFILE`* variable.

Using the --thumbprint Option

You can supply the thumbprint for the target ESXi host or vCenter Server system in the `--thumbprint` parameter or the *`VI_THUMBPRINT`* variable.

When you run a command, ESXCLI first checks whether a certificate file is available. If not, ESXCLI checks whether a thumbprint of the target server is available. If not, you receive an error of the following type.

```
Connect to sof-40583-srv failed. Server SHA-1 thumbprint:
5D:01:06:63:55:9D:DF:FE:38:81:6E:2C:FA:71:BC:Usin63:82:C5:16:51 (not trusted).
```

You can run the command with the thumbprint to establish the trust relationship, or add the thumbprint to the *`VI_THUMBPRINT`* variable. For example, using the thumbprint of the ESXi host above, you can run the following command.

```
esxcli --server myESXi --username user1 --password 'my_password' --thumbprint
5D:01:06:63:55:9D:DF:FE:38:81:6E:2C:FA:71:BC:63:82:C5:16:51 storage nfs list
```

Connection Options for ESXCLI Host Management Command Execution

You can use connection options that are available for all ESXCLI host management commands.

The following table lists options that are available for all ESXCLI host management commands in alphabetical order. The table includes options for use on the command line and variables for use in configuration files.

> **IMPORTANT**
> ESXCLI supports both IPv4 and IPv6 connections.

See Run Host Management Commands from a Windows System and Run Host Management Commands from a Linux System.

Option and Environment Variable	Description

`--cacertsfile <certsfile>`
 	
`-t <certs_file>`
 	
`VI_CACERTFILE=<cert_file_path>`
	Specifies the CA (Certificate Authority) certificate file, in PEM format, to verify the identity of the vCenter Server system or ESXi system to run the command on.
	
	You can only run ESXCLI commands if a trust relationship exists between the host you are running the command on and the system you are targeting with the `--server` option (ESXi host or vCenter Server system). You can establish the trust relationship by specifying the CA certificate file or by passing in the thumbprint for each target server (ESXi host or vCenter Server system).

`--config <cfg_file_full_path>`
 	
`VI_CONFIG=<cfg_file_full_path>`
	Uses the configuration file at the specified location.
	
	Specify a path that is readable from the current directory.

`--credstore <credstore>`
 	
`VI_CREDSTORE=<credstore>`
	Name of a credential store file. Defaults to *`<HOME>`*`/.vmware/credstore/vicredentials.xml` on Linux and *`<APPDATA>`*`/VMware/credstore/vicredentials.xml` on Windows. Commands for setting up the credential store are included in the vSphere SDK for Perl. See the *vSphere*

Table continued on next page

Continued from previous page

Option and Environment Variable	Description
	SDK for Perl Programming Guide for information on managing the credential store.

`--encoding <encoding>`
 	
`VI_ENCODING=<encoding>`
	Specifies which encoding to use. Several encodings are supported.
	• `utf8`
	• `cp936` (Simplified Chinese)
	• `shftjis` (Japanese)
	• `iso-885901` (German)
	You can use `--encoding` to specify the encoding for ESXCLI to map to when it is run on a foreign language system.

`--passthroughauth`
 	
`VI_PASSTHROUGHAUTH`
	If you specify this option, the system uses the Microsoft Windows Security Support Provider Interface (SSPI) for authentication. Trusted users are not prompted for a user name and password. See the Microsoft Web site for detailed information on SSPI.
	
	This option is supported only if you are connecting to a vCenter Server system.

`--passthroughauthpackage <package>`
 	
`VI_PASSTHROUGHAUTHPACKAGE= <package>`
	Use this option with `--passthroughauth` to specify a domain-level authentication protocol to be used by Windows. By default, SSPI uses the Negotiate protocol, which means that client and server try to negotiate a protocol that both support.
	
	If the vCenter Server system to which you are connecting is configured to use a specific protocol, you can specify that protocol by using this option.
	
	This option is supported only if you are running ESXCLI on a Windows system and connecting to a vCenter Server system.

`--password <passwd>`
 	
`VI_PASSWORD=<passwd>`
	Uses the specified password (used with `--username`) to log in to the server.
	• If `--server` specifies a vCenter Server system, the user name and password apply to that server. If you can log in to the vCenter Server system, you need no additional authentication to run commands on the ESXi hosts that server manages.
	• If `--server` specifies an ESXi host, the user name and password apply to that server.

Table continued on next page

Continued from previous page

Option and Environment Variable	Description
	Use the empty string (' ' on Linux and " " on Windows) to indicate no password. If you do not specify a user name and password on the command line, the system prompts you and does not echo your input to the screen.

`--portnumber <number>`
 	
`VI_PORTNUMBER=<number>`
	Uses the specified port to connect to the system specified by `--server`. Default is 443.

`--protocol <HTTP|HTTPS>`
 	
`VI_PROTOCOL=<HTTP|HTTPS>`
	Uses the specified protocol to connect to the system specified by `--server`. Default is HTTPS.

`--savesessionfile <file>`
 	
`VI_SAVESESSIONFILE=<file>`
	Saves a session to the specified file. The session expires if it is idle for 30 minutes.

`--server <server>`
 	
`VI_SERVER=<server>`
	Uses the specified ESXi or vCenter Server system. Default is `localhost`.
	
	Use the `--vihost` option to specify the ESXi host that you want to run the command against. See Authenticating Through and vCenter Single Sign-On.

`--servicepath <path>`
 	
`VI_SERVICEPATH=<path>`
	Uses the specified service path to connect to the ESXi host. Default is `/sdk/webService`.

`--sessionfile <file>`
 	
`VI_SESSIONFILE=<file>`
	Uses the specified session file to load a previously saved session. The session must be unexpired.

`--thumbprint <thumbprint>`
 	
`VI_THUMBPRINT=<thumbprint>`
	Expected SHA-1, SHA-256, or SHA-512 host certificate thumbprint if no CA certificates file is provided in the `--cacertsfile` option. The thumbprint is returned by the server in the error message if you attempt to run a command without specifying a thumbprint or certificate file.

`--url <url>`
 	
`VI_URL=<url>`
	Connects to the specified vSphere Web Services SDK URL.

`--username <u_name>`
 	
`VI_USERNAME=<u_name>`
	Uses the specified user name.
	
	• If `--server` specifies a vCenter Server system, the user name and password apply to that server. If you can log in to the vCenter Server system, you need no additional

Table continued on next page

Continued from previous page

Option and Environment Variable	Description
	authentication to run commands on the ESXi hosts that server manages. • If `--server` specifies an ESXi system, the user name and password apply to that system. If you do not specify a user name and password on the command line, the system prompts you and does not echo your input to the screen.

`--vihost <host>`
 	
`-h <host>`
	When you run an ESXCLI command with the `--server` option pointing to a vCenter Server system, use `--vihost` to specify the ESXi host to run the command against.
	
	NOTE
	This option is not supported for each command. If supported, the option is included when you run `<cmd> --help`.

Using ESXCLI Commands in Scripts

Most administrators run scripts to perform the same task repeatedly or to perform a task on multiple hosts. You can run ESXCLI commands from one administration server against multiple target servers.

For example, when a new data store becomes available in your environment, you must make that data store available to each ESXi host. The following sample script illustrates how to make a NAS data store available to three hosts (`esxi_server_a, esx_server_b, and esxi_server_c`).

The sample assumes that a configuration file `/home/admin/.visdkrc.<hostname>` exists for each host. For example, the configuration file for `esxi_server_a` has the following contents.

```
VI_SERVER = esxi_server_a

VI_USERNAME = root

VI_PASSWORD = xysfdjkat
```

The script adds the NAS data store to each host defined in `VIHOSTS`.

```
#!/bin/bash


VI_CONFIG_FILE=/home/admin/viconfig

VIHOSTS=(esxi_server_a esx_server_b esxi_server_c)



for VIHOST in ${VIHOSTS[@]}
```

```
do

echo "Adding NAS datastore for ${VIHOST} ..."

esxcli --config ${VI_CONFIG_FILE} storage nfs add --host ${VIHOST} --share <share point>
--volume-name <volume name>

esxcli --config ${VI_CONFIG_FILE} storage nfs list

done
```

Run Host Management Commands from a Windows System

After you install ESXCLI, you can test the installation by running a command from the Windows command prompt.

1. Open a command prompt.
2. Change to the ESXCLI installation directory.

   The default directory is `<system_drive>:\Program Files (x86)\VMware\esxcli`.

3. Run the command, including the connection options and other options.

   ```
   esxcli --server <vc_hostname_or_IP> --username <user_name> --password <password>
   network ip interface list
   ```

Run Host Management Commands from a Linux System

After installation, you can run ESXCLI commands from the command prompt.

1. Open a command prompt.
2. Change to the ESXCLI installation directory.

   The default directory is `/opt/vmware/esxcli`.

3. Run the command, including the connection options.

   ```
   <command> <conn_options> <params>
   ```

   Specify connection options in a configuration file or pass them on the command line.

   ```
   esxcli --server <vc_hostname_or_IP> --username snow\-white --password dwarf\$ network
   ip interface list
   ```

   The system prompts you for a user name and password for the target server.

ESXCLI Concepts and Examples

ESXCLI Concepts and Examples explains how to use ESXCLI commands and includes command overviews and examples.

Intended Audience

This book is for experienced system administrators who are familiar with vSphere administration tasks and data center operations and know how to use commands in scripts.

ESXCLI Commands Overview

This chapter provides an overview of ESXCLI, connection options, and discusses ESXCLI and lockdown mode.

Introduction to ESXCLI

You can use the commands in the ESXCLI package to manage many aspects of an ESXi host. You can run ESXCLI commands remotely or in the ESXi Shell.

You can install the ESXCLI command set on a supported Linux or Windows system. See *Getting Started with ESXCLI*. After installation, you can run ESXCLI commands from the Linux or Windows system. You can manage ESXi hosts with ESXCLI commands by specifying connection options such as the target host, user, and password or a configuration file. See Connection Options for ESXCLI Host Management Commands.

Targets and Protocols for ESXCLI Host Management Commands

Most ESXCLI commands are used to manage or retrieve information about one or more ESXi hosts. They can target an ESXi host or a vCenter Server system.

When you target a vCenter Server system, you can use
`--vihost`
to specify the ESXi host to run the command against. All commands support the HTTP and HTTPS protocols.

ESXCLI Commands Available on Different ESXi Hosts

The available ESXCLI commands depend on the ESXi host version.

When you run an ESXCLI command, you must know the commands supported on the target host. For example, if you run commands against ESXi 7.x hosts, ESXCLI 7.x commands are supported. If you run commands against ESXi 6.x hosts, ESXCLI 6.x commands are supported.

Some commands or command outputs are determined by the host type. In addition, VMware partners might develop custom ESXCLI commands that you can run on hosts where the partner VIB has been installed.

Run `esxcli --server <target> --help` for a list of namespaces supported on the target. You can drill down into the namespaces for additional help.

Trust Relationship Requirement for ESXCLI Commands

ESXCLI checks whether a trust relationship exists between the machine where you run the ESXCLI command and the ESXi host. An error results if the trust relationship does not exist.

Download and Install the vCenter Server Certificate

You can download the vCenter Server root certificate by using a Web browser and add it to the trusted certificates on the machine where you plan to run ESXCLI commands.

1. Enter the URL of the vCenter Server system into a Web browser.
2. Click the **Download trusted root CA certificates** link.
3. Verify that the extension of the downloaded file is `.zip`.

   The file is a ZIP file of all certificates in the TRUSTED_ROOTS store.

4. Extract the ZIP file.
   A certificates folder is extracted. The folder includes files with the extension `.0`, `.1`, and so on, which are certificates, and files with the extension `.r0`, `.r1`, and so on which are CRL files associated with the certificates.
5. Add the trusted root certificates to the list of trusted roots.

   The process differs depending on the platform that you are on.

You can now run ESXCLI commands against any host that is managed by the trusted vCenter Server system without supplying additional information if you specify the vCenter Server system in the `--server` option and the ESXi host in the `--vihost` option.

Using the --cacertsfile Option

Using a certificate to establish the trust relationship is the most secure option.

You can specify the certificate with the `--cacertsfile` parameter or the *`VI_CACERTFILE`* variable.

Using the --thumbprint Option

You can supply the thumbprint for the target ESXi host or vCenter Server system in the `--thumbprint` parameter or the *`VI_THUMBPRINT`* variable.

When you run a command, ESXCLI first checks whether a certificate file is available. If not, ESXCLI checks whether a thumbprint of the target server is available. If not, you receive an error of the following type.

```
Connect to sof-40583-srv failed. Server SHA-1 thumbprint:
5D:01:06:63:55:9D:DF:FE:38:81:6E:2C:FA:71:BC:Usin63:82:C5:16:51 (not trusted).
```

You can run the command with the thumbprint to establish the trust relationship, or add the thumbprint to the *`VI_THUMBPRINT`* variable. For example, using the thumbprint of the ESXi host above, you can run the following command.

```
esxcli --server myESXi --username user1 --password 'my_password' --thumbprint
5D:01:06:63:55:9D:DF:FE:38:81:6E:2C:FA:71:BC:63:82:C5:16:51 storage nfs list
```

Using ESXCLI Output

Many ESXCLI commands generate output you might want to use in your application. You can run `esxcli` with the `--formatter` dispatcher option and send the resulting output as input to a parser.

The `--formatter` option supports three values - `csv`, `xml`, and `keyvalue` and is used before any namespace.

The following example lists all file system information in CSV format.

```
esxcli --formatter=csv storage filesystem list
```

You can pipe the output to a file.

```
esxcli --formatter=keyvalue storage filesystem list > myfilesystemlist.txt
```

IMPORTANT
You should always use a formatter for consistent output.

Connection Options for ESXCLI Host Management Commands

You can run ESXCLI host management commands and other commands with several different connection options.

You can target hosts directly or target a vCenter Server system and specify the host you want to manage.

IMPORTANT
ESXCLI supports both the IPv4 protocol and the IPv6 protocol.

See the *Getting Started with ESXCLI* documentation for a complete list and examples.

ESXCLI Host Management Commands and Lockdown Mode

For additional security, an administrator can place one or more hosts managed by a vCenter Server system in lockdown mode. Lockdown mode affects login privileges for the ESXi host.

See the *vSphere Security* document for a detailed discussion of normal lockdown mode and strict lockdown mode, and of how to enable and disable them.

To make changes to ESXi systems in lockdown mode, you must go through a vCenter Server system that manages the ESXi system as the user `vpxuser` and include both the
`--server`
and
`--vihost`
options.

```
esxcli --server MyVC --vihost MyESXi storage filesystem list
```

The command prompts for the vCenter Server system user name and password.

If you have problems running a command on an ESXi host directly, without specifying a vCenter Server target, check whether lockdown mode is enabled on that host.

Managing Hosts

You can use host management commands to stop, reboot ESXi hosts, enter and exit maintenance mode, and manage modules. You can also manage components, base images, add-ons, and host updates.

vSphere 7.0 introduces components, base images, and add-ons. A component is an installation packaging element. One component might contain multiple VIBs. The component has a version number that is separate from the version numbers of any VIBs it contains, though it might be the same. Each VIB in a component can contain a device driver, a CIM module, or an application for communicating between the two. Components simplify the packaging and installation of installable items on ESXi. The base image is an ESXi image that VMware provides with every release of ESXi. The base image is a collection of components that is complete and can boot up a server. Base images have a user-readable name and a unique version that is updated with every major or minor release of ESXi. The add-on is a collection of components that

does not represent a complete, bootable image. You cannot use vendor add-ons on their own. To customize an ESXi release, you must add a vendor add-on to an ESXi base image.

For information on updating ESXi hosts with the **`esxcli software`** command and on changing the host acceptance level to match the level of a VIB that you might want to use for an update, see the *`VMware ESXi Upgrade`* document.

Stopping and Rebooting Hosts with ESXCLI

You can shut down or reboot an ESXi host by using the vSphere Client or ESXCLI.

Shutting down a managed host disconnects it from the vCenter Server system, but does not remove the host from the inventory. You can shut down a single host or all hosts in a data center or cluster.

To shut down a host, run **`esxcli system shutdown poweroff`**. You must specify the `--reason` option and supply a reason for the shutdown. A `--delay` option allows you to specify a delay interval, in seconds.

To reboot a host, run **`esxcli system shutdown reboot`**. You must specify the `--reason` option and supply a reason for the reboot. A `--delay` option allows you to specify a delay interval, in seconds.

Enter and Exit Maintenance Mode with ESXCLI

You place a host in maintenance mode to service it, for example, to install more memory. A host enters or leaves maintenance mode only as the result of a user request.

`esxcli system maintenanceMode set` allows you to enable or disable maintenance mode. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. To enter maintenance mode, run the following command.

   ```
   esxcli <conn_options> system maintenanceMode set --enable true
   ```

   After all virtual machines on the host have been suspended or migrated, the host enters maintenance mode.

   NOTE
   You cannot deploy or power on a virtual machine on hosts in maintenance mode.

2. To exit maintenance mode, run the following command.

   ```
   esxcli <conn_options> system maintenanceMode set --enable false
   ```

   NOTE
   If you attempt to exit maintenance mode when the host is no longer in maintenance mode, an error informs you that maintenance mode is already disabled.

Manage Modules with ESXCLI

The **`esxcli system module`** command supports setting and retrieving VMkernel module options. Not all VMkernel modules have settable module options.

The following example illustrates how to examine and enable a VMkernel module. Specify one of the connection options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. List information about the module.

   ```
   esxcli <conn_options> system module list --module=<module_name>
   ```

   The system returns the name, type, value, and description of the module.
2. List all enabled or loaded modules.

   ```
   esxcli <conn_options> system module list --enabled=true
   ```

   ```
   esxcli <conn_options> system module list --loaded=true
   ```

3. Enable the module.

   ```
   esxcli <conn_options> system module set --module=<module_name> --enabled=true
   ```

4. Set the parameter.

   ```
   esxcli system module parameters set --module=<module_name> --parameter-string=<parame
   ter_string>
   ```

5. Verify that the module is configured.

   ```
   esxcli <conn_options> system module parameters list --module=<module_name>
   ```

Retrieve Information about Components, Base Image, and Add-On on a Host with ESXCLI

You can use ESXCLI to list the components on an ESXi host and retrieve detailed information about each component. You can also retrieve information about base image and add-on on an ESXi host.

The following example illustrates how to list all components, retrieve details about an individual component, and retrieve information about base image and add-on. Specify one of the connection options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. List all components installed on the host.

   ```
   esxcli <conn_options> software component list
   ```

   The list contains information about the name, display name, version, display version, vendor, creation date, and acceptance level of each component.
2. Retrieve detailed information about a specific component from the list by specifying the component name.

   NOTE
   The component name can be either in the `<component_name>` or `<component_name>:<version>` format. If there are multiple components with the same name, you must specify the version. Component names are case-sensitive.

   ```
   esxcli <conn_options> software component get -n <component_name>
   ```

3. Retrieve detailed information about the base image installed on the host.

   ```
   esxcli <conn_options> software baseimage get
   ```

4. Retrieve detailed information about the add-on installed on the host.

   ```
   esxcli <conn_options> software addon get
   ```

Manage Components with ESXCLI

You can use ESXCLI to retrieve information about components in a depot, install or update components, and remove components.

The following example illustrates how to list all components in a depot, retrieve details about an individual component in the depot, install or update a component on an ESXi host, and remove a component from the host. Installing a component update might be useful for driver troubleshooting purposes. You can remove any unnecessary components from the host. Specify one of the connection options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. List all components in the depot by specifying the full remote URL to the `index.xml` file of the online depot or the local absolute datastore path to an offline bundle `.zip` file.

   ```
   esxcli <conn_options> software sources component list -d
   <depot_url_or_offline_bundle_path>
   ```

   The list contains information about the name, display name, version, display version, vendor, creation date, and acceptance level of each component.
2. Retrieve detailed information about a specific component in the depot by specifying the component name.

   NOTE
   The component name can be either in the *<component_name>* or *<component_name>*:*<version>* format. If there are multiple components with the same name, you must specify the version. Component names are case-sensitive.

   ```
   esxcli <conn_options> software sources component get -n <component_name> -d <depot_ur
   l_or_offline_bundle_path>
   ```

3. Install or update a component on the ESXi host by specifying the name and version of the component in the depot and the location of the depot.

   ```
   esxcli <conn_options> software component apply -n <component_name>:<version> -d <depo
   t_url_or_offline_bundle_path>
   ```

4. Remove a component from the host.

   NOTE
   The component name can be either in the *<component_name>* or *<component_name>*:*<version>* format. If there are multiple components with the same name, you must specify the version. Component names are case-sensitive.

   ```
   esxcli <conn_options> software component remove -n <component_name>
   ```

Manage Base Images and Add-Ons with ESXCLI

You can use ESXCLI to retrieve information about base images and add-ons in a depot. You can also apply a complete image by using a JSON software specification that specifies the base image, add-on, and components to install on the ESXi host.

The following example illustrates how to list all base images and add-ons in a depot, retrieve details about a specific base image or add-on, and apply a complete image to a host. You can also verify the signatures of installed components after applying the complete image. Specify one of the connection options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. List all base images in the depot by specifying the full remote URL to the `index.xml` file of the online depot or the local absolute datastore path to an offline bundle `.zip` file.

   ```
   esxcli <conn_options> software sources baseimage list -d
   <depot_url_or_offline_bundle_path>
   ```

   The list contains information about the release ID, version, vendor, release date, and acceptance level of each base image.
2. Retrieve detailed information about a specific base image in the depot by specifying the base image version.

   ```
   esxcli <conn_options> software sources baseimage get -b <base_image_version> -d <depo
   t_url_or_offline_bundle_path>
   ```

3. List all add-ons in the depot by specifying the full remote URL to the `index.xml` file of the online depot or the local absolute datastore path to an offline bundle `.zip` file.

   ```
   esxcli <conn_options> software sources addon list -d
   <depot_url_or_offline_bundle_path>
   ```

   The list contains information about the release ID, version, vendor, release date, and acceptance level of each add-on.
4. Retrieve detailed information about a specific add-on in the depot by specifying the add-on name.

   NOTE
   The add-on name can be either in the <*add-on_name*> or <*add-on_name*>:<*version*> format. If there are multiple add-ons with the same name, you must specify the version. Add-on names are case-sensitive.

   ```
   esxcli <conn_options> software sources addon get -a <add-on_name> -d <depot_url_or_of
   fline_bundle_path>
   ```

5. Prepare a JSON software specification.
   - Obtain a JSON software specification exported from a vSphere Lifecycle Manager managed cluster.
   - Create a custom JSON software specification.
     You can use the following JSON software specification syntax to apply a base image only.

     ```
     {
        "add_on": null,
        "base_image": {
            "version": "<base_image_version>"
        },
        "components": {}
     }
     ```

     You can use the following JSON software specification syntax to apply a base image, an add-on, and one or more components.

     ```
     {
        "base_image": {
            "version": "<base_image_version>"
     ```

```
      },
      "add_on": {
         "name": "<add-on_name>",
         "version": "<add-on_version>"
      },
      "components": {
         "<component_name>": "<component_version>"
      }
   }
```

6. Apply a complete image to the ESXi host by specifying the location of the JSON software specification and the location of the depot.

   NOTE
   The location of the JSON software specification can be either a remote URL or a local file path. You can use a software specification exported from a vSphere Lifecycle Manager managed cluster. You can specify multiple depots.

   ```
   esxcli <conn_options> software apply -s <location_of_software_spec>.json -d <depot_url_or_offline_bundle_path>
   ```

7. Verify the signatures of components installed on the host.

   NOTE
   If you have not rebooted the ESXi host after applying the complete image, you can verify the newly applied image by using the `--rebooting-image` option.

   ```
   esxcli <conn_options> software component signature verify
   ```

   A list of all installed components appears. The list contains the name, version, vendor, acceptance level, and signature verification result of each component.

Updating Hosts

When you add custom drivers or patches to a host, the process is called an update.

- Update ESXi 6.0 hosts with **`esxcli software vib`** commands discussed in the *vSphere Upgrade* documentation included in the vSphere 6.0 documentation set.
- Update ESXi 6.5 hosts with **`esxcli software vib`** commands discussed in the *vSphere Upgrade* documentation included in the vSphere 6.5 documentation set.
- Update ESXi 6.7 hosts with **`esxcli software vib`** commands discussed in the *VMware ESXi Upgrade* documentation included in the vSphere 6.7 documentation set.
- Update ESXi 7.0 hosts with **`esxcli software vib`** commands discussed in the *VMware ESXi Upgrade* documentation included in the vSphere 7.0 documentation set.

Configuring the GuestStore Repository with ESXCLI

You can set the URL to the GuestStore repository, clear the URL setting, and retrieve information about the URL configuration by using ESXCLI.

The GuestStore feature provides an easy and flexible mechanism to maintain and distribute various content types across multiple virtual machines on multiple ESXi hosts at the same time. By using the GuestStore framework, you can make sure that the distributed content is always consistent and you can improve the content management in your environment. For more information about the GuestStore feature, see *vSphere Virtual Machine Administration*.

Set the GuestStore Repository with ESXCLI

You can use ESXCLI commands to set the URL to the GuestStore repository and verify the configuration by retrieving the currently set URL.

1. Set the GuestStore repository URL.

   NOTE
   The URL must point to a datastore path where GuestStore content is stored. The datastore path must be accessible to the ESXi host. If you want to set up a common GuestStore repository across multiple ESXi hosts, you should use a shared datastore path.

   ```
   esxcli system settings gueststore repository set --url "<datastore_path>"
   ```

   The following example contains a possible syntax for the datastore path.

   ```
   esxcli system settings gueststore repository set --url "ds:///vmfs/volumes/
   <datastore_uuid>/GuestStore"
   ```

2. Retrieve the GuestStore repository URL.

   ```
   esxcli system settings gueststore repository get
   ```

   You receive the currently set URL in the output.

   ```
   URL: <datastore_path>
   ```

Clear the GuestStore Repository Setting with ESXCLI

You can use ESXCLI commands to clear the GuestStore repository URL setting and verify that the URL is not set.

1. Clear the GuestStore repository URL setting.

   ```
   esxcli system settings gueststore repository set --url ""
   ```

2. Retrieve the GuestStore repository URL.

   ```
   esxcli system settings gueststore repository get
   ```

   You receive the URL information in the output.

   ```
   URL: <not set>
   ```

Managing Files

You can use ESXCLI to manage VMFS (Virtual Machine File System) volumes.

NOTE
See Managing Storage for information about storage manipulation commands.

Introduction to Virtual Machine File Management

You can use the vSphere Client or ESXCLI commands to access different types of storage devices that your ESXi host discovers and to deploy datastores on those devices.

> **NOTE**
> Datastores are logical containers, analogous to file systems, that hide specifics of each storage device and provide a uniform model for storing virtual machine files. Datastores can be used for storing ISO images, virtual machine templates, and floppy images. The vSphere Client uses the term datastore exclusively. In ESXCLI, the term datastore, as well as VMFS or NFS volume, refer to the same logical container on the physical device.

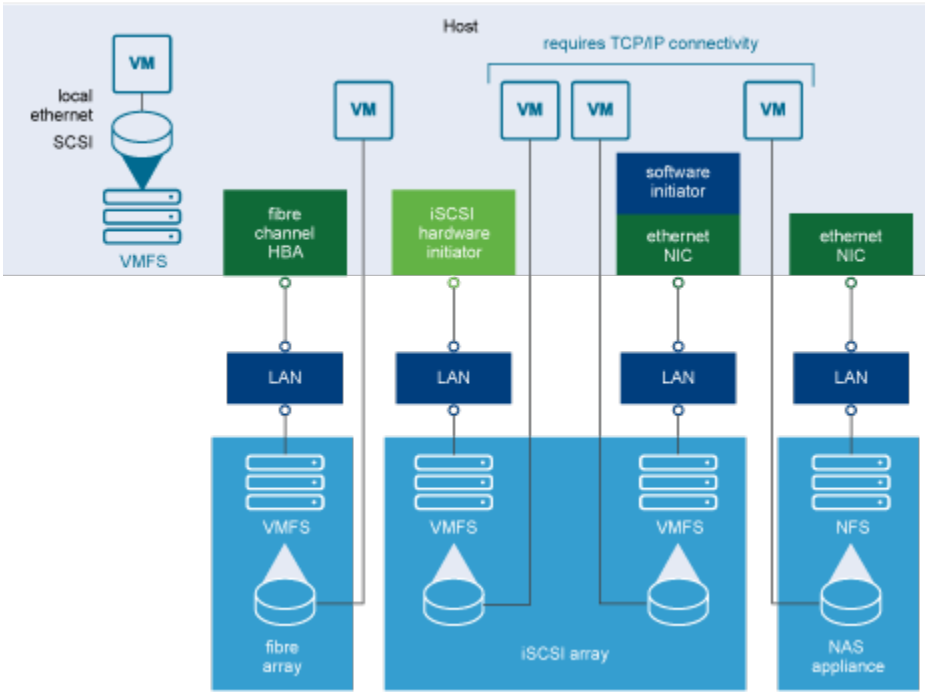
Depending on the type of storage you use, datastores can be backed by the VMFS and NFS file system formats.

- Virtual Machine File System (VMFS) - High-performance file system that is optimized for storing virtual machines. Your host can deploy a VMFS datastore on any SCSI-based local or networked storage device, including Fibre Channel and iSCSI SAN equipment. As an alternative to using the VMFS datastore, your virtual machine can have direct access to raw devices and use a mapping file (RDM) as a proxy.

  You can manage VMFS and RDMs with the vSphere Client.
- Network File System (NFS) - The NFS client built into ESXi uses the NFS protocol over TCP/IP to access a designated NFS volume that is located on a NAS server. The ESXi host can mount the volume and use it for its storage needs. vSphere supports versions 3 and 4.1 of the NFS protocol. Typically, the NFS volume or directory is created by a storage administrator and is exported form the NFS server. The NFS volumes do not need to be formatted with a local file system, such as VMFS. You can mount the volumes directly and use them to store and boot virtual machines in the same way that you use VMFS datastores. The host can access a designated NFS volume located on an NFS server, mount the volume, and use it for any storage needs.

  You manage NAS storage devices from the vSphere Client or with the **`esxcli storage nfs`** command. The diagram below illustrates different types of storage, but it is for conceptual purposes only. It is not a recommended configuration.

Figure 180: Virtual Machines Accessing Different Types of Storage



Managing VMFS Volumes

Different commands are available for listing, mounting, and unmounting VMFS volumes and for listing, mounting, and unmounting VMFS snapshot volumes.

- Managing VMFS volumes
  `esxcli storage filesystem list` shows all volumes, mounted and unmounted, that are resolved, that is, that are not snapshot volumes.

  `esxcli storage filesystem unmount` unmounts a currently mounted filesystem. Use this command for snapshot volumes or resolved volumes.
- Managing snapshot volumes
  `esxcli storage vmfs snapshot` commands can be used for listing, mounting, and resignaturing snapshot volumes. See Mounting Datastores with Existing Signatures and Resignaturing VMFS Copies.

Managing Duplicate VMFS Datastores

In some cases VMFS datastores can have duplicate UUIDs.

Each VMFS datastore created in a LUN has a unique UUID that is stored in the file system superblock. When the LUN is replicated or when a snapshot is made, the resulting LUN copy is identical, byte-for-byte, to the original LUN. As a result, if the original LUN contains a VMFS datastore with UUID X, the LUN copy appears to contain an identical VMFS datastore, or a VMFS datastore copy, with the same UUID X.

ESXi hosts can determine whether a LUN contains the VMFS datastore copy, and either mount the datastore copy with its original UUID or change the UUID to resignature the datastore.

When a LUN contains a VMFS datastore copy, you can mount the datastore with the existing signature or assign a new signature. The *`vSphere Storage`* documentation discusses volume resignaturing in detail.

Mounting Datastores with Existing Signatures

You can mount a VMFS datastore copy without changing its signature if the original is not mounted.

For example, you can maintain synchronized copies of virtual machines at a secondary site as part of a disaster recovery plan. In the event of a disaster at the primary site, you can mount the datastore copy and power on the virtual machines at the secondary site.

> **IMPORTANT**
> You can mount a VMFS datastore only if it does not conflict with an already mounted VMFS datastore that has the same UUID.

When you mount the VMFS datastore, ESXi allows both read and write operations to the datastore that resides on the LUN copy. The LUN copy must be writable. The datastore mounts are persistent and valid across system reboots.

Mount a Datastore with ESXCLI

The **`esxcli storage filesystem`** commands support mounting and unmounting volumes. You can also specify whether to persist the mounted volumes across reboots by using the `--no-persist` option.

Use the **`esxcli storage filesystem`** command to list mounted volumes, mount new volumes, and unmount a volume. Specify one of the connection options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. List all volumes that have been detected as snapshots.

   ```
   esxcli <conn_options> storage filesystem list
   ```

2. Run **`esxcli storage filesystem mount`** with the volume label or volume UUID.

   ```
   esxcli <conn_options> storage filesystem volume mount --volume-label=<label>|--
   volume-uuid=<VMFS-UUID>
   ```

   NOTE
   This command fails if the original copy is online.

You can later run **`esxcli storage filesystem volume unmount`** to unmount the snapshot volume.

```
esxcli <conn_options> storage filesystem volume unmount --volume-label=<label>|--volume-
uuid=<VMFS-UUID>
```

Resignaturing VMFS Copies

You can use datastore resignaturing to retain the data stored on the VMFS datastore copy.

When resignaturing a VMFS copy, the ESXi host assigns a new UUID and a new label to the copy, and mounts the copy as a datastore distinct from the original. Because ESXi prevents you from resignaturing the mounted datastore, unmount the datastore before resignaturing.

The default format of the new label assigned to the datastore is `snap-<snapID>-<oldLabel>`, where `<snapID>` is an integer and `<oldLabel>` is the label of the original datastore.

When you perform datastore resignaturing, consider the following points.

- Datastore resignaturing is irreversible.
- The LUN copy that contains the VMFS datastore that you resignature is no longer treated as a LUN copy.
- A spanned datastore can be resignatured only if all its extents are online.
- The resignaturing process is crash and fault tolerant. If the process is interrupted, you can resume it later.
- You can mount the new VMFS datastore without a risk of its UUID conflicting with UUIDs of any other datastore, such as an ancestor or child in a hierarchy of LUN snapshots.

Resignature a VMFS Copy with ESXCLI

The **`esxcli storage vmfs snapshot`** commands support resignaturing a snapshot volume.

Specify one of the connection options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. List unresolved snapshots or replica volumes.

   ```
   esxcli <conn_options> storage vmfs snapshot list
   ```

2. Unmount the copy.

   ```
   esxcli <conn_options> storage filesystem unmount
   ```

3. Run the resignature command.

   ```
   esxcli <conn_options> storage vmfs snapshot resignature --volume-label=<label>|--
   volume-uuid=<id>
   ```

   The command returns to the prompt or signals an error.

After resignaturing, you might have to perform the following operations.

- If the resignatured datastore contains virtual machines, update references to the original VMFS datastore in the virtual machine files, including `.vmx`, `.vmdk`, `.vmsd`, and `.vmsn`.
- To power on virtual machines, register them with the vCenter Server system.

Reclaiming Unused Storage Space

When VMFS datastores reside on thin-provisioned LUNs, you can use ESXCLI commands to reclaim the unused logical blocks of a thin-provisioned LUN formatted with VMFS.

When you run the commands, you must specify the volume label `--volume-label` or the volume ID `--volume-uuid` but you cannot specify both.

In each iteration, the command issues unmap commands to the number of file system blocks that are specified by the optional `reclaim-unit` argument, which defaults to 200.

The following examples illustrate how to use the command.

```
# esxcli storage vmfs unmap --volume-label datastore1 --reclaim-unit 100

# esxcli storage vmfs unmap -l datastore1 -n 100

# esxcli storage vmfs unmap --volume-uuid 515615fb-1e65c01d-b40f-001d096dbf97 --reclaim-
unit 500

# esxcli storage vmfs unmap -u 515615fb-1e65c01d-b40f-001d096dbf97 -n 500

# esxcli storage vmfs unmap -l datastore1

# esxcli storage vmfs unmap -u 515615fb-1e65c01d-b40f-001d096dbf97
```

Managing Storage

A virtual machine uses a virtual disk to store its operating system, program files, and other data associated with its activities. A virtual disk is a large physical file, or a set of files, that can be copied, moved, archived, and backed up.

To store virtual disk files and manipulate the files, a host requires dedicated storage space. ESXi storage is storage space on a variety of physical storage systems, local or networked, that a host uses to store virtual machine disks.

Managing iSCSI Storage discusses iSCSI storage management. Managing Third-Party Storage Arrays explains how to manage the Pluggable Storage Architecture, including Path Selection Plugin (PSP) and Storage Array Type Plug-in (SATP) configuration.

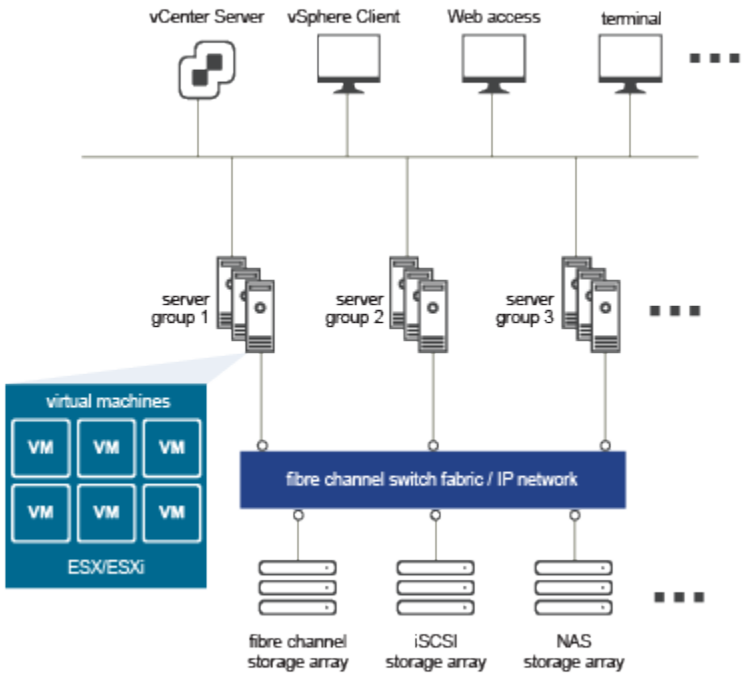
For information on masking and unmasking paths with ESXCLI, see the *vSphere Storage* documentation.

Introduction to Storage

Fibre Channel SAN arrays, iSCSI SAN arrays, and NAS arrays are widely used storage technologies supported by VMware vSphere to meet different data center storage needs.

The storage arrays are connected to and shared between groups of servers through storage area networks. This arrangement allows aggregation of the storage resources and provides more flexibility in provisioning them to virtual machines.

Figure 181: vSphere Data Center Physical Topology



How Virtual Machines Access Storage

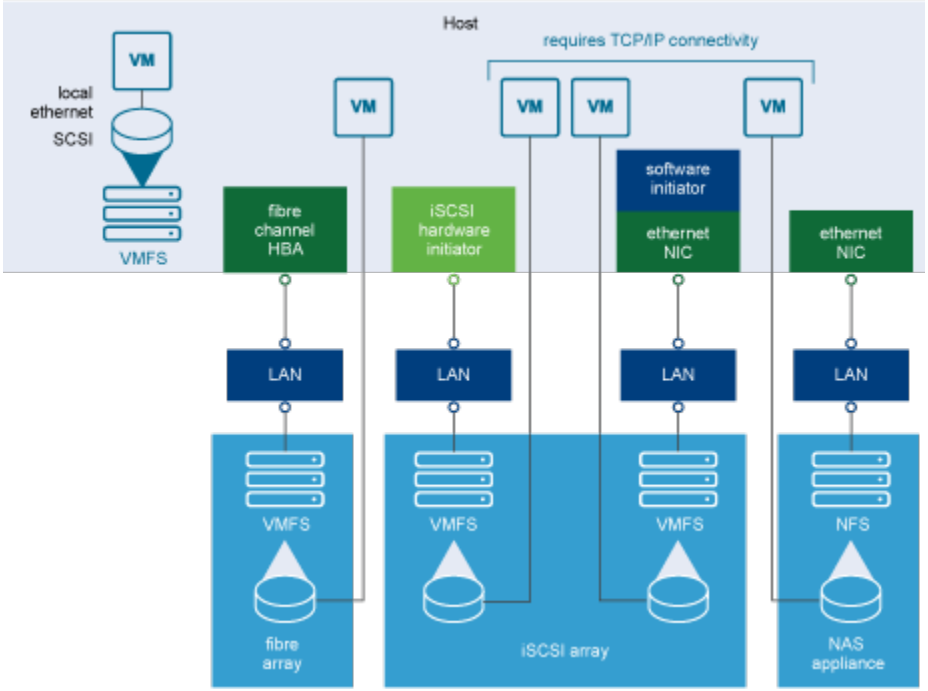
A virtual disk hides the physical storage layer from the virtual machine's operating system.

Regardless of the type of storage device that your host uses, the virtual disk always appears to the virtual machine as a mounted virtual storage device. As a result, you can run operating systems that are not certified for specific storage equipment, such as SAN, in the virtual machine.

When a virtual machine communicates with its virtual disk stored on a datastore, it issues virtual device interface commands appropriate for the device. Because datastores can exist on various types of physical storage, these commands are encapsulated into other forms, depending on the protocol that the ESXi host uses to connect to a storage device.

Figure: Virtual Machines Accessing Different Types of Storage depicts five virtual machines that use different types of storage to illustrate the differences between each type.

Figure 182: Virtual Machines Accessing Different Types of Storage



You can use ESXCLI commands to manage the virtual machine file system and storage devices.

- Datastores - Several commands allow you to manage datastores and are useful for multiple protocols.
  - LUNs - Use **`esxcli storage core`** to display available LUNs and mappings for each VMFS volume to its corresponding partition. See Examining LUNs.
  - Path management - Use **`esxcli storage core`** to list information about Fibre Channel or iSCSI LUNs and to change a path's state. See Managing Paths. Use the ESXCLI command to view and modify path policies. See Managing Path Policies.
  - Rescan - Use **`esxcli storage core`** to perform a rescan operation each time you reconfigure your storage setup. See Scanning Storage Adapters.
- Storage devices - Several commands manage only specific storage devices.
  - NFS storage - Use **`esxcli storage nfs`** to manage NAS storage devices. See Managing NFS/NAS Datastores.
  - iSCSI storage - Use **`esxcli iscsi`** to manage both hardware and software iSCSI. See Managing iSCSI Storage.
- Software-defined storage - vSphere supports several types of software-defined storage.
  - vSAN storage - Use commands in the **`esxcli vsan`** namespace to manage vSAN. See Monitoring and Managing Storage.
  - Virtual Flash storage - Use commands in the **`esxcli storage vflash`** namespace to manage VMware vSphere Flash Read Cache.
  - Virtual volumes - Virtual volumes offer a different layer of abstraction than datastores. As a result, finer-grained management is possible. Use commands in the **`esxcli storage vvol`** namespace.

Datastores

ESXi hosts use storage space on a variety of physical storage systems, including internal and external devices and networked storage.

A host can discover storage devices to which it has access and format them as datastores. Each datastore is a special logical container, analogous to a file system on a logical volume, where the host places virtual disk files and other virtual machine files. Datastores hide specifics of each storage product and provide a uniform model for storing virtual machine files.

Depending on the type of storage you use, datastores can be backed by the following file system formats.

- Virtual Machine File System (VMFS) - High-performance file system optimized for storing virtual machines. Your host can deploy a VMFS datastore on any SCSI-based local or networked storage device, including Fibre Channel and iSCSI SAN equipment.
- Network File System (NFS) - File system on a NAS storage device. ESXi supports NFS version 3 and 4.1. The host can access a designated NFS volume located on an NFS server, mount the volume, and use it for any storage needs.

Storage Device Naming

Each storage device, or LUN, is identified by several device identifier names.

Device Identifiers

Depending on the type of storage, the ESXi host uses different algorithms and conventions to generate an identifier for each storage device.

- SCSI INQUIRY identifiers - The host uses the SCSI INQUIRY command to query a storage device and uses the resulting data, in particular the Page 83 information, to generate a unique identifier. SCSI INQUIRY device identifiers are unique across all hosts, persistent, and have one of the following formats.
  - `naa.<number>`
  - `t10.<number>`
  - `eui.<number>`

  These formats follow the T10 committee standards. See the SCSI-3 documentation on the T10 committe Web site for information on Page 83.
- Path-based identifier. If the device does not provide the information on Page 83 of the T10 committee SCSI-3 documentation, the host generates an `mpx.<path>` name, where `<path>` represents the first path to the device, for example, `mpx.vmhba1:C0:T1:L3`. This identifier can be used in the same way as the SCSI inquiry identifiers.

  The `mpx.` identifier is created for local devices on the assumption that their path names are unique. However, this identifier is neither unique nor persistent and could change after every boot.

  Typically, the path to the device has the following format.

  `vmhba<adapter>:C<channel>:T<target>:L<LUN>`
  - `vmbh<adapter>` is the name of the storage adapter. The name refers to the physical adapter on the host, not the SCSI controller used by the virtual machines.
  - `C<channel>` is the storage channel number. Software iSCSI adapters and dependent hardware adapters use the channel number to show multiple paths to the same target.
  - `T<target>` is the target number. Target numbering is determined by the host and might change if the mappings of targets that are visible to the host change. Targets that are shared by different hosts might not have the same target number.
  - `L<LUN>` is the LUN number that shows the position of the LUN within the target. The number is provided by the storage system. If a target has only one LUN, the LUN number is always zero (0).

Legacy Identifiers

In addition to the SCSI INQUIRY or mpx identifiers, ESXi generates an alternative legacy name, called VML name, for each device. Use the device UID instead.

Examining LUNs

A LUN (Logical Unit Number) is an identifier for a disk volume in a storage array target.

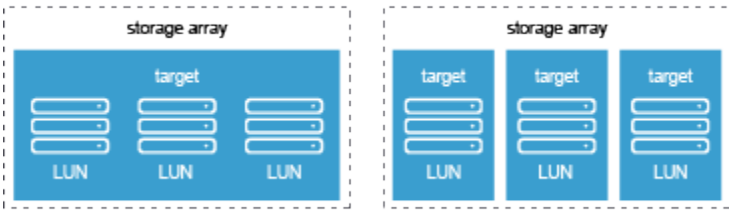
Target and Device Representation

In the ESXi context, the term target identifies a single storage unit that a host can access. The terms device and LUN describe a logical volume that represents storage space on a target.

The terms device and LUN mean a SCSI volume presented to the host from a storage target.

Different storage vendors present their storage systems to ESXi hosts in different ways. Some vendors present a single target with multiple LUNs on it. Other vendors, especially iSCSI vendors, present multiple targets with one LUN each.

Figure 183: Target and LUN Representations



In Figure: Target and LUN Representations, three LUNs are available in each configuration. On the left, the host sees one target, but that target has three LUNs that can be used. Each LUN represents an individual storage volume. On the right, the host sees three different targets, each having one LUN.

Examining LUNs with ESXCLI

You can use **`esxcli storage core`** to display information about available LUNs on ESXi.

You can run one of the following commands to examine LUNs. Specify one of the connection options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

- List all logical devices known on this system with detailed information.
  `esxcli <conn_options> storage core device list`

  The command lists device information for all logical devices on this system. The information includes the name (UUID), device type, display name, and multipathing plugin. Specify the `--device` option to only list information about a specific device. See Storage Device Naming for background information.

```
naa.5000c50037b3967e

Display Name: <name> (naa.5000c50037b3967e)

Has Settable Display Name: true

Size: 953869

Device Type: Direct-Access

...

naa.500000e014e7a4e0

Display Name: <name> (naa.500000e014e7a4e0)

Has Settable Display Name: true
```

```
Size: 70007

Device Type: Direct-Access

...

mpx.vmhba0:C0:T0:L0

Display Name: Local <name> CD-ROM (mpx.vmhba0:C0:T0:L0)

Has Settable Display Name: false

Size: 0

Device Type: CD-ROM
```

- List a specific logical device with its detailed information.
  `esxcli <conn_options> storage core device list -d mpx.vmhba32:C0:T1:L0`
- List all device unique identifiers.
  `esxcli <conn_options> storage core device list`

  The command lists the primary UID for each device, such as `naa.xxx` or other primary name, and any other UIDs for each UID (VML name). You can specify `--device` to only list information for a specific device.
- Print mappings for VMFS volumes to the corresponding partition, path to that partition, VMFS UUID, extent number, and volume names.
  `esxcli <conn_option> storage filesystem list`
- Print HBA devices with identifying information.
  `esxcli <conn_options> storage core adapter list`

  The return value includes adapter and UID information.
- Print a mapping between HBAs and the devices it provides paths to.
  `esxcli <conn_options> storage core path list`

Detach a Device and Remove a LUN

Before you can remove a LUN, you must detach the corresponding device by using the vSphere Client, or the **`esxcli storage core device set`** command.

- Make sure you are familiar with virtual machine migration. See the *vCenter Server and Host Management* documentation.
- Make sure you are familiar with datastore mounting and unmounting. See Mount a Datastore with ESXCLI.

Detaching a device brings a device offline. Detaching a device does not impact path states. If the LUN is still visible, the path state is not set to dead.

1. Migrate virtual machines from the device you plan to detach.
2. Unmount the datastore deployed on the device.

   If the unmount fails, ESXCLI returns an error. If you ignore that error, you will get an error when you attempt to detach a device with a VMFS partition still in use.
3. If the unmount failed, check whether the device is in use.

   ```
   esxcli storage core device world list -d <device>
   ```

   If a VMFS volume is using the device indirectly, the world name includes the string `idle0`. If a virtual machine uses the device as an RDM, the virtual machine process name is displayed. If any other process is using the raw device, the information is displayed.

4. Detach the storage device.

   ```
   esxcli storage core device set -d naa.xxx... --state=off
   ```

   Detach is persistent across reboots and device unregistration. Any device that is detached remains detached until a manual attach operation. Rescan does not bring persistently detached devices back online. A persistently detached device comes back in the off state.

   ESXi maintains the persistent information about the device's offline state even if the device is unregistered. You can remove the device information by running `esxcli storage core device detached remove -d naa.12`.

5. To troubleshoot the detach operation, list all devices that were detached manually.

   ```
   esxcli storage core device detached list
   ```

6. Perform a rescan.

   ```
   esxcli <conn_options> storage core adapter rescan
   ```

Reattach a Device

When you have completed storage reconfiguration, you can reattach the storage device, mount the datastore, and restart the virtual machines.

Make sure you are familiar with datastore mounting. See Mounting Datastores with Existing Signatures.

1. Check whether the device is detached.

   ```
   esxcli storage core device detached list
   ```

2. Attach the device.

   ```
   esxcli storage core device set -d naa.XXX --state=on
   ```

3. Mount the datastore and restart virtual machines.

Working with Permanent Device Loss

In some cases a permanent device loss (PDL) might occur.

With earlier ESXi releases, an APD (All Paths Down) event results when the LUN becomes unavailable. The event is difficult for administrators because they do not have enough information about the state of the LUN to know which corrective action is appropriate.

The ESXi host can determine whether the cause of an APD event is temporary, or whether the cause is PDL. A PDL status occurs when the storage array returns SCSI sense codes indicating that the LUN is no longer available or that a severe, unrecoverable hardware problem exist with it. ESXi has an improved infrastructure that can speed up operations of upper-layer applications in a device loss scenario.

> **IMPORTANT**
> Do not plan for APD or PDL events, for example, when you want to upgrade your hardware. Instead, perform an orderly removal of LUNs from your ESXi server, which is described in Detach a Device and Remove a LUN, perform the operation, and add the LUN back.

Removing a PDL LUN

How you remove a PDL LUN depends on whether it was in use.

- If the LUN that goes into PDL is not in use by any user process or by the VMkernel, the LUN disappears by itself after a PDL.
- If the LUN was in use when it entered PLD, delete the LUN manually by following the process described in Detach a Device and Remove a LUN.

Reattach a PDL LUN

You can reattach a PDL LUN after it has been removed.

1. Return the LUN to working order.
2. Remove any users of the device.

   You cannot bring a device back without removing active users. The ESXi host cannot know whether the device that was added back has changed. ESXi must be able to treat the device similarly to a new device being discovered.

3. Perform a rescan to get the device back in working order.

Managing Paths

To maintain a constant connection between an ESXi host and its storage, ESXi supports multipathing. With multipathing you can use more than one physical path for transferring data between the ESXi host and the external storage device.

In case of failure of an element in the SAN network, such as an HBA, switch, or cable, the ESXi host can fail over to another physical path. On some devices, multipathing also offers load balancing, which redistributes I/O loads between multiple paths to reduce or eliminate potential bottlenecks.

The storage architecture supports a special VMkernel layer, Pluggable Storage Architecture (PSA). The PSA is an open modular framework that coordinates the simultaneous operation of multiple multipathing plug-ins (MPPs). You can manage PSA by using ESXCLI commands. See Managing Third-Party Storage Arrays. This section assumes you are using only PSA plug-ins included in vSphere by default.

Multipathing with Local Storage and FC SANs

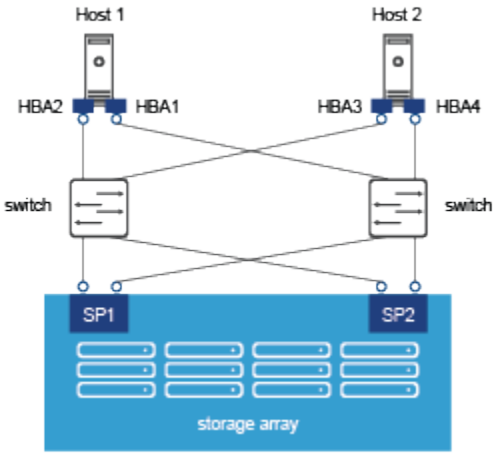
Multipathing is a technique that lets you use more than one physical path that transfers data between the host and an external storage device.

In a simple multipathing local storage topology, you can use one ESXi host with two HBAs. The ESXi host connects to a dual-port local storage system through two cables. This configuration ensures fault tolerance if one of the connection elements between the ESXi host and the local storage system fails.

To support path switching with FC SAN, the ESXi host typically has two HBAs available from which the storage array can be reached through one or more switches. Alternatively, the setup can include one HBA and two storage processors so that the HBA can use a different path to reach the disk array.

In FC Multipathing, multiple paths connect each host with the storage device. For example, if HBA1 or the link between HBA1 and the switch fails, HBA2 takes over and provides the connection between the server and the switch. The process of one HBA taking over for another is called HBA failover.

Figure 184: FC Multipathing



If SP1 or the link between SP1 and the switch breaks, SP2 takes over and provides the connection between the switch and the storage device. This process is called SP failover. ESXi multipathing supports HBA and SP failover.

After you have set up your hardware to support multipathing, you can use the vSphere Client or ESXCLI commands to list and manage paths. You can perform the following tasks.

- List path information. See Listing Path Information with ESXCLI.
- Change path state. See Disable a Path with ESXCLI.
- Change path policies. See Set Policy Details for Devices that Use Round Robin.
- Mask paths. See the *vSphere Storage* documentation.
- Manipulate the rules that match paths to multipathing plugins to newly discovered devices. See Managing Claim Rules.
- Run or rerun claim rules or unclaim paths. See Managing Claim Rules.
- Rescan storage adapters. See Scanning Storage Adapters.

Listing Path Information with ESXCLI

You can run **`esxcli storage core path`** to display information about Fibre Channel or iSCSI LUNs.

> **IMPORTANT**
> Use industry-standard device names, with format `eui.xxx` or `naa.xxx` to ensure consistency. Do not use VML LUN names unless device names are not available.
> Names of virtual machine HBAs are not guaranteed to be valid across reboots.

You can display information about paths by running **`esxcli storage core path`**. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

- List all devices with their corresponding paths, state of the path, adapter type, and other information.
  ```
  esxcli <conn_options> storage core path list
  ```
- Limit the display to only a specified path or device.
  ```
  esxcli <conn_options> storage core path list --path <path>

  esxcli <conn_options> storage core path list --device <device>
  ```
- List the statistics for the SCSI paths in the system. You can list all paths or limit the display to a specific path.

```
esxcli <conn_options> storage core path stats get

esxcli <conn_options> storage core path stats get --path <path>
```

- List detailed information for the paths for the device specified with `--device`.
  ```
  esxcli <conn_options> storage core path list -d <naa.xxxxxx>
  ```
- List all adapters.
  ```
  esxcli <conn_options> storage core adapter list
  ```
- Rescan all adapters.
  ```
  esxcli <conn_options> storage core adapter rescan
  ```

Disable a Path with ESXCLI

You can temporarily disable a path with ESXCLI for maintenance or other reasons, and enable the path when you need it again.

If you are changing a path's state, the change operation fails if I/O is active when the path setting is changed. Reissue the command. You must issue at least one I/O operation before the change takes effect.
Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. List all devices and corresponding paths.

   ```
   esxcli <conn_options> storage core path list
   ```

   The display includes information about each path's state.
2. Set the state of a LUN path to off.

   ```
   esxcli <conn_options> storage core path set --state off --path vmhba32:C0:T1:L0
   ```

When you are ready, set the path state to active again.

```
esxcli <conn_options> storage core path set --state active --path vmhba32:C0:T1:L0
```

Managing Path Policies

For each storage device managed by NMP, and not PowerPath, an ESXi host uses a path selection policy. If you have a third-party PSP installed on your host, its policy also appears on the list.

Supported Path Policies

The following path policies are supported by default.

Policy	Description
`VMW_PSP_FIXED`	The host uses the designated preferred path, if it has been configured. Otherwise, the host selects the first working path discovered at system boot time. If you want the host to use a particular preferred path, specify it through the vSphere Client, or by using **`esxcli storage nmp psp`**

Table continued on next page

Continued from previous page

Policy	Description
	`fixed deviceconfig set`. See Change the Path Policy with ESXCLI. The default policy for active-active storage devices is `VMW_PSP_FIXED`. **NOTE** If the host uses a default preferred path and the path's status turns to Dead, a new path is selected as preferred. However, if you explicitly designate the preferred path, it will remain preferred even when it becomes inaccessible.
`VMW_PSP_MRU`	The host selects the path that it used most recently. When the path becomes unavailable, the host selects an alternative path. The host does not revert back to the original path when that path becomes available again. There is no preferred path setting with the MRU policy. MRU is the default policy for active-passive storage devices. The `VMW_PSP_MRU` ranking capability allows you to assign ranks to individual paths. To set ranks to individual paths, use the **`esxcli storage nmp psp generic pathconfig set`** command. For details, see the VMware knowledge base article 2003468.
`VMW_PSP_RR`	The host uses an automatic path selection algorithm that rotates through all active paths when connecting to active-passive arrays, or through all available paths when connecting to active-active arrays. Automatic path selection implements load balancing across the physical paths available to your host. Load balancing is the process of spreading I/O requests across the paths. The goal is to optimize throughput performance such as I/O per second, megabytes per second, or response times. `VMW_PSP_RR` is the default for a number of arrays and can be used with both active-active and active-passive arrays to implement load balancing across paths for different LUNs.

Path Policy Effects

The type of array and the path policy determine the behavior of the host.

Policy	Active/Active Array	Active/Passive Array
Most Recently Used	Administrator action is required to fail back after path failure.	Administrator action is required to fail back after path failure.

Table continued on next page

Continued from previous page

Policy	Active/Active Array	Active/Passive Array
Fixed	VMkernel resumes using the preferred path when connectivity is restored.	VMkernel attempts to resume by using the preferred path. This action can cause path thrashing or failure when another SP now owns the LUN.
Round Robin	No fail back.	Next path in round robin scheduling is selected.

Multipathing Considerations

You should consider a number of key points when working with multipathing.

The following considerations help you with multipathing.

- If no SATP is assigned to the device by the claim rules, the default SATP for iSCSI or FC devices is `VMW_SATP_DEFAULT_AA`. The default PSP is `VMW_PSP_FIXED`.
- When the system searches the SATP rules to locate a SATP for a given device, it searches the driver rules first. If there is no match, the vendor/model rules are searched, and finally the transport rules are searched. If no match occurs, NMP selects a default SATP for the device.
- If `VMW_SATP_ALUA` is assigned to a specific storage device, but the device is not ALUA-aware, no claim rule match occurs for this device. The device is claimed by the default SATP based on the device's transport type.
- The default PSP for all devices claimed by `VMW_SATP_ALUA` is `VMW_PSP_MRU`. The `VMW_PSP_MRU` selects an active/optimized path as reported by the `VMW_SATP_ALUA`, or an active/unoptimized path if there is no active/optimized path. This path is used until a better path is available (MRU). For example, if the `VMW_PSP_MRU` is currently using an active/unoptimized path and an active/optimized path becomes available, the `VMW_PSP_MRU` will switch the current path to the active/optimized one.
- While `VMW_PSP_MRU` is typically selected for ALUA arrays by default, certain ALUA storage arrays need to use `VMW_PSP_FIXED`. To check whether your storage array requires `VMW_PSP_FIXED`, see the *VMware Compatibility Guide* or contact your storage vendor. When using `VMW_PSP_FIXED` with ALUA arrays, unless you explicitly specify a preferred path, the ESXi host selects the most optimal working path and designates it as the default preferred path. If the host selected path becomes unavailable, the host selects an alternative available path. However, if you explicitly designate the preferred path, it remains preferred no matter what its status is.
- By default, the PSA claim rule 101 masks Dell array pseudo devices. Do not delete this rule, unless you want to unmask these devices.

Change the Path Policy with ESXCLI

You can change the path policy with ESXCLI.

Verify that you are familiar with the supported path policies. See Managing Path Policies.

Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. Ensure your device is claimed by the NMP plug-in.

   Only NMP devices allow you to change the path policy.

   ```
   esxcli <conn_options> storage nmp device list
   ```

2. Retrieve the list of path selection policies on the system to see which values are valid for the `--psp` option when you set the path policy.

   ```
   esxcli storage core plugin registration list --plugin-class="PSP"
   ```

3. Set the path policy by using ESXCLI.

   ```
   esxcli <conn_options> storage nmp device set --device naa.xxx --psp VMW_PSP_RR
   ```

4. If you specified the `VMW_PSP_FIXED` policy, you must make sure the preferred path is set correctly.
   a) Check which path is the preferred path for a device.

      ```
      esxcli <conn_options> storage nmp psp fixed deviceconfig get --device naa.xxx
      ```

   b) If necessary, change the preferred path.

      ```
      esxcli <conn_options> storage nmp psp fixed deviceconfig set --device naa.xxx
      --path vmhba3:C0:T5:L3
      ```

      The command sets the preferred path to `vmhba3:C0:T5:L3`. Run the command with `--default` to clear the preferred path selection.

Set Policy Details for Devices that Use Round Robin

ESXi hosts can use multipathing for failover. With some storage devices, ESXi hosts can also use multipathing for load balancing.

To achieve better load balancing across paths, administrators can specify that the ESXi host should switch paths under specific circumstances. Different options determine when the ESXi host switches paths and what paths are chosen. Only a limited number of storage arrays support round robin.
You can use **`esxcli storage nmp psp roundrobin`** to retrieve and set round robin path options on a device controlled by the `roundrobin PSP`. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. Retrieve path selection settings for a device that is using the `roundrobin PSP`.

   ```
   esxcli <conn_options> storage nmp psp roundrobin deviceconfig get --device na.xxx
   ```

2. Set the path selection. You can specify when the path should change, and whether unoptimized paths should be included.
   – Use `--bytes` or `--iops` to specify when the path should change, as in the following examples.

     ```
     esxcli <conn_options> storage nmp psp roundrobin deviceconfig set --type "bytes"
     -B 12345 --device naa.xxx
     ```

     Sets the device specified by `--device` to switch to the next path each time 12345 bytes have been sent along the current path.

     ```
     esxcli <conn_options> storage nmp psp roundrobin deviceconfig set --type=iops --
     iops 4200 --device naa.xxx
     ```

     Sets the device specified by `--device` to switch after 4200 I/O operations have been performed on a path.
   – Use `useano` to specify that the round robin PSP should include paths in the active, unoptimized state in the round robin set (1) or that the PSP should use active, unoptimized paths only if no active optimized paths are available (0). If you do not include this option, the PSP includes only active optimized paths in the round robin path set.

Scheduling Queues for Virtual Machine I/O

You can use ESXCLI to enable or disable per file I/O scheduling.

By default, vSphere provides a mechanism that creates scheduling queues for each virtual machine file. Each file has individual bandwidth controls. This mechanism ensures that the I/O for a particular virtual machine goes into its own separate queue and does not interfere with the I/O of other virtual machines.

This capability is enabled by default. You can turn it off by using the `esxcli system settings kernel set -s isPerFileSchedModelActive` option.

- Run `esxcli system settings kernel set -s isPerFileSchedModelActive -v FALSE` to disable per file scheduling.
- Run `esxcli system settings kernel set -s isPerFileSchedModelActive -v TRUE` to enable per file scheduling.

Managing NFS/NAS Datastores

ESXi hosts can access a designated NFS volume located on a NAS (Network Attached Storage) server, can mount the volume, and can use it for its storage needs. You can use NFS volumes to store and boot virtual machines in the same way that you use VMFS datastores.

Capabilities Supported by NFS/NAS

An NFS client built into the ESXi hypervisor uses the Network File System (NFS) protocol over TCP/IP to access a designated NFS volume that is located on a NAS server. The ESXi host can mount the volume and use it for its storage needs.

vSphere supports versions 3 and 4.1 of the NFS protocol.

Typically, the NFS volume or directory is created by a storage administrator and is exported from the NFS server. The NFS volume does not need to be formatted with a local file system, such as VMFS. You can mount the volume directly on ESXi hosts, and use it to store and boot virtual machines in the same way that you use VMFS datastores.

In addition to storing virtual disks on NFS datastores, you can also use NFS as a central repository for ISO images, virtual machine templates, and so on. If you use the datastore for ISO images, you can connect the virtual machine's CD-ROM device to an ISO file on the datastore and install a guest operating system from the ISO file.

ESXi hosts support the following shared storage capabilities on NFS volumes.

- VMware vMotion and Storage vMotion
- High Availability (HA), Fault Tolerance, and Distributed Resource Scheduler (DRS)
- ISO images, which are presented as CD-ROMs to virtual machines
- Virtual machine snapshots
- Host profiles
- Virtual machines with large capacity virtual disks, or disks greater than 2 TB. Virtual disks created on NFS datastores are thin-provisioned by default, unless you use hardware acceleration that supports the Reserve Space operation. See *Hardware Acceleration on NAS Devices* in the vSphere Storage documentation.

In addition to storing virtual disks on NFS datastores, you can also use NFS as a central repository for ISO images, virtual machine templates, and so on.

To use NFS as a shared repository, you create a directory on the NFS server and then mount the directory as a datastore on all hosts. If you use the datastore for ISO images, you can connect the virtual machine's CD-ROM device to an ISO file on the datastore and install a guest operating system from the ISO file.

Manage a NAS File System with ESXCLI

You can list, add, and delete a NAS file system with ESXCLI.

For more information on connection options, see Connection Options for ESXCLI Host Management Commands.

1. List all known NAS file systems.

   ```
   esxcli <conn_options> storage nfs list
   ```

For each NAS file system, the command lists the mount name, share name, and host name and whether the file system is mounted. If no NAS file systems are available, the system does not return a NAS filesystem and returns to the command prompt.

2. Add a new NAS file system to the ESXi host.

   Specify the NAS server with `--host`, the volume to use for the mount with `--volume-name`, and the share name on the remote system to use for this NAS mount point with `--share`.

   ```
   esxcli <conn_options> storage nfs add --host=dir42.eng.vmware.com --share=/
   <mount_dir> --volume-name=nfsstore-dir42
   ```

   This command adds an entry to the known NAS file system list and supplies the share name of the new NAS file system. You must supply the host name, share name, and volume name for the new NAS file system.

3. Add a second NAS file system with read-only access.

   ```
   esxcli <conn_options> storage nfs add --host=dir42.eng.vmware.com --share=/home --
   volume-name=FileServerHome2 --readonly
   ```

4. Delete one of the NAS file systems.

   ```
   esxcli <conn_options> storage nfs remove --volume-name=FileServerHome2
   ```

   This command unmounts the NAS file system and removes it from the list of known file systems.

Monitor and Manage FibreChannel SAN Storage

The **`esxcli storage san commands`** help administrators troubleshoot issues with I/O devices and fabric, and include Fibre Channel, FCoE, iSCSI, SAS protocol statistics.

The commands allow you to retrieve device information and I/O statistics from those device. You can also issue Loop Initialization Primitives (LIP) to FC/FCoE devices and you can reset SAS devices.
For FC and FCoE devices, you can retrieve FC events such as `RSCN`, `LINKUP`, `LINKDOWN`, Frame Drop and `FCoE CVL`. The commands log a warning in the VMkernel log if it encounters too many Link Toggling or frame drops.
The following example examines and resets SAN storage through a FibreChannel adapter. Instead of fc, the information retrieval commands can also use `iscsi`, `fcoe`, and `sas`.

1. List adapter attributes.

   ```
   esxcli storage san fc list
   ```

2. Retrieve all events for a Fibre Channel I/O device.

   ```
   esxcli storage san fc events get
   ```

3. Clear all I/O Device Management events for the specified adapter.

   ```
   esxcli storage san fc events clear --adapter adapter
   ```

4. Reset the adapter.

   ```
   esxcli storage san fc reset
   ```

Monitoring and Managing vSAN Storage

vSAN is a distributed layer of software that runs natively as a part of the ESXi hypervisor. vSAN aggregates local or direct-attached storage disks of a host cluster and creates a single storage pool shared across all hosts of the cluster.

While supporting VMware features that require shared storage, such as HA, vMotion, and DRS, vSAN eliminates the need for an external shared storage and simplifies storage configuration and virtual machine provisioning activities.

You can use ESXCLI commands to retrieve vSAN information, manage vSAN clusters, perform network management, add storage, set the policy, and perform other monitoring and management tasks. Type `esxcli vsan --help` for a complete list of commands.

Retrieve vSAN Information

You can use ESXCLI commands to retrieve vSAN information.

1. Verify which VMkernel adapters are used for vSAN communication.

   ```
   esxcli vsan network list
   ```

2. List storage disks that were claimed by vSAN.

   ```
   esxcli vsan storage list
   ```

3. Get vSAN cluster information.

   ```
   esxcli vsan cluster get
   ```

Manage a vSAN Cluster

You can activate vSAN when you create host clusters or enable vSAN on existing clusters. When enabled, vSAN aggregates all local storage disks available on the hosts into a single datastore shared by all hosts.

You can run these commands in the ESXi Shell for a host, or the command affects the target host that you specify as part of the ESXCLI connection options.

1. Join the target host to a given vSAN cluster.

   ```
   esxcli vsan cluster join --cluster-uuid <uuid>
   ```

   NOTE
   The UUID of the cluster is required.

2. Verify that the target host is joined to a vSAN cluster.

   ```
   esxcli vsan cluster get
   ```

3. Remove the target host from the vSAN cluster.

   ```
   esxcli vsan cluster leave
   ```

Add and Remove vSAN Storage

You can use ESXCLI commands to add and remove vSAN storage. You can also retrieve information about the vSAN storage configuration.

1. Add an HDD or data disk for use by vSAN.

   ```
   esxcli vsan storage add --disks <device_name>
   ```

NOTE
The command expects an empty disk, which is partitioned or formatted. Specify a device name, for example, `mpx.vmhba2:C0:T1:L0`.

2. Add an SSD disk for use by vSAN.

   ```
   esxcli vsan storage add --ssd <device_name>
   ```

   NOTE
   The command expects an empty disk, which is partitioned or formatted. Specify a device name, for example, `mpx.vmhba2:C0:T1:L0`.

3. List the vSAN storage configuration. You can display the complete list, or filter to show only a single device.

   ```
   esxcli vsan storage list --device <device>
   ```

4. Remove disks or disk groups.

   NOTE
   You can remove disks or disk groups only when vSAN is in manual mode. For the automatic disk claim mode, the remove action is not supported.

   – Remove an individual vSAN disk.
   ```
   esxcli vsan storage remove --disk <device_name>
   ```
   Instead of specifying the device name, you can specify the UUID if you include the `--uuid` option.
   – Remove a disk group's SSD and each of its backing HDD drives from vSAN usage.
   ```
   esxcli vsan storage remove --ssd <device_name>
   ```
   Instead of specifying the device name, you can specify the UUID if you include the `--uuid` option. Any SSD that you remove from vSAN becomes available for such features as Flash Read Cache.

Monitoring vSphere Flash Read Cache

Flash Read Cache™ lets you accelerate virtual machine performance through the use of host resident flash devices as a cache.

The *vSphere Storage* documentation discusses vSphere Flash Read Cache in some detail.

You can reserve a Flash Read Cache for any individual virtual disk. The Flash Read Cache is created only when a virtual machine is powered on, and it is discarded when a virtual machine is suspended or powered off. When you migrate a virtual machine you have the option to migrate the cache. By default the cache is migrated if the virtual flash module on the source and destination hosts are compatible. If you do not migrate the cache, the cache is rewarmed on the destination host. You can change the size of the cache while a virtual machine is powered on. In this instance, the existing cache is discarded and a new write-through cache is created, which results in a cache warm up period. The advantage of creating a new cache is that the cache size can better match the application's active data.

Flash Read Cache supports write-through or read caching. Write-back or write caching are not supported. Data reads are satisfied from the cache, if present. Data writes are dispatched to the backing storage, such as a SAN or NAS. All data that is read from or written to the backing storage is unconditionally stored in the cache.

NOTE
Not all workloads benefit with a Flash Read Cache. The performance boost depends on your workload pattern and working set size. Read-intensive workloads with working sets that fit into the cache can benefit from a Flash Read Cache configuration. By configuring Flash Read Cache for your read-intensive workloads additional I/O resources become available on your shared storage, which can result in a performance increase for other workloads even though they are not configured to use Flash Read Cache.

You can manage vSphere Flash Read Cache from the vSphere Client. You can monitor Flash Read Cache by using commands in the `esxcli storage vflash` namespace. The following table lists available commands. See the *`ESXCLI Reference`* for a list of options to each command.

Table 114: Commands for Monitoring vSphere Flash Read Cache

Command	Description
`esxcli storage vflash cache get`	Gets individual `vflash` cache info.
`esxcli storage vflash cache list`	Lists individual `vflash` caches.
`esxcli storage vflash cache stats get`	Gets `vflash` cache statistics.
`esxcli storage vflash cache stats reset`	Resets `vflash` cache statistics.
`esxcli storage vflash device list`	Lists `vflash` SSD devices.
`esxcli storage vflash module get`	Gets `vflash` module info.
`esxcli storage vflash module list`	Lists `vflash` modules.
`esxcli storage vflash module stats get`	Gets `vflash` module statistics.

Monitoring and Managing Virtual Volumes

The Virtual Volumes functionality changes the storage management paradigm from managing space inside datastores to managing abstract storage objects handled by storage arrays.

With Virtual Volumes, an individual virtual machine, not the datastore, becomes a unit of storage management, while storage hardware gains complete control over virtual disk content, layout, and management. The *`vSphere Storage`* documentation discusses Virtual Volumes in some detail and explains how to manage them by using the vSphere Client.

The following ESXCLI commands are available for managing display information about virtual volumes and for unbinding all Virtual Volumes from all vendor providers. See the *`vSphere Storage`* documentation for information on creating Virtual Volumes and configuring multipathing and SCSI-based endpoints.

Table 115: VVol Commands

Command	Description
`esxcli storage vvol daemon unbindall`	Unbinds all Virtual Volume instances from all storage providers that are known to the ESXi host.
`esxcli storage vvol protocolendpoint list`	Lists the VVol protocol endpoints currently known to the ESXi host.
`esxcli storage vvol storagecontainer list`	Lists the VVol storage containers currently known to the ESXi host.
`esxcli storage vvol storagecontainer restore`	Restores storage containers of vendor providers that are registered on the host.
`esxcli storage vvol vasacontext get`	Gets the VASA context (VC UUID).
`esxcli storage vvol vendorprovider list`	Lists the vendor providers registered on the host.

Table continued on next page

Continued from previous page

Command	Description
esxcli storage vvol vendorprovider restore	Restores the vendor providers that are registered on the host.

Configuring FCoE Adapters

ESXi can use Fibre Channel over Ethernet (FCoE) adapters to access Fibre Channel storage.

The FCoE protocol encapsulates Fibre Channel frames into Ethernet frames. As a result, your host does not need special Fibre Channel links to connect to Fibre Channel storage, but can use 10 Gbit lossless Ethernet to deliver Fibre Channel traffic.

To use FCoE, you need to install FCoE adapters. The adapters that VMware supports generally fall into two categories, hardware FCoE adapters and software FCoE adapters.

- Hardware FCoE adapters include completely offloaded specialized Converged Network Adapters (CNAs) that contain network and Fibre Channel functionalities on the same card. When such an adapter is installed, your host detects and can use both CNA components. In the vSphere Client, the networking component appears as a standard network adapter (vmnic) and the Fibre Channel component as a FCoE adapter (vmhba). You do not have to configure a hardware FCoE adapter to be able to use it.
- A software FCoE adapter is a software code that performs some of the FCoE processing. The adapter can be used with a number of NICs that support partial FCoE offload. Unlike the hardware FCoE adapter, the software adapter must be activated.

Scanning Storage Adapters

You must perform a rescan operation each time you reconfigure your storage setup.

You can scan by using the vSphere Client or the **esxcli storage core adapter rescan** command.

esxcli storage core adapter rescan supports the following additional options.

- `-a|--all or -A|--adapter=<string>` – Scan all adapters or a specified adapter.
- `-S|--skip-claim` – Skip claiming of new devices by the appropriate multipath plug-in.
- `-F|--skip-fs-scan` – Skip filesystem scan.
- `-t|--type` – Specify the type of scan to perform. The command either scans for all changes (`all`) or for added, deleted, or updated adapters (`add, delete, update`).

The following command scans a specific adapter and skips the filesystem scan that is performed by default.

```
esxcli <conn_options> storage core adapter rescan --adapter=vmhba33 --skip-claim
```

The command returns an indication of success or failure, but no detailed information.

Retrieving SMART Information

You can use ESXCLI to retrieve information related to SMART. SMART is a monitoring system for computer hard disks that reports information about the disks.

You can use the following example syntax to retrieve SMART information.

```
esxcli storage core device smart get -d device
```

What the command returns depends on the level of SMART information that the device supports. If no information is available for a parameter, the output displays N/A, as in the following sample output.

```
Parameter                     Value  Threshold  Worst
-------------------------------------------------
Health Status                 OK     N/A        N/A
Media Wearout Indicator       N/A    N/A        N/A
Write Error Count             N/A    N/A        N/A
Read Error Count              119    6          74
Power-on Hours                57     0          57
Power Cycle Count             100    20         100
Reallocated Sector Count      100    36         100
Raw Read Error Rate           119    6          74
Drive Temperature             38     0          49
Driver Rated Max Temperature  62     45         51
Write Sectors TOT Count       200    0          200
Read Sectors TOT Count        100    0          253
Initial Bad Block Count       N/A    N/A        N/A
```

Managing iSCSI Storage

ESXi systems include iSCSI technology to access remote storage using an IP network. You can use the vSphere Client or commands in the `esxcli iscsi` namespace to configure both hardware and software iSCSI storage for your ESXi system.

See the *vSphere Storage* documentation for additional information.

iSCSI Storage Overview

With iSCSI, SCSI storage commands that your virtual machine issues to its virtual disk are converted into TCP/IP protocol packets and transmitted to a remote device, or target, on which the virtual disk is located. To the virtual machine, the device appears as a locally attached SCSI drive.

To access remote targets, the ESXi host uses iSCSI initiators. Initiators transport SCSI requests and responses between ESXi and the target storage device on the IP network. ESXi supports the following types of initiators.

- Software iSCSI adapter - VMware code built into the VMkernel. Allows an ESXi host to connect to the iSCSI storage device through standard network adapters. The software initiator handles iSCSI processing while communicating with the network adapter.
- Hardware iSCSI adapter - Offloads all iSCSI and network processing from your host. Hardware iSCSI adapters are broken into two types.
  - Dependent hardware iSCSI adapter - Leverages the VMware iSCSI management and configuration interfaces.
  - Independent hardware iSCSI adapter - Leverages its own iSCSI management and configuration interfaces.

See the *vSphere Storage* documentation for details on setup and failover scenarios.

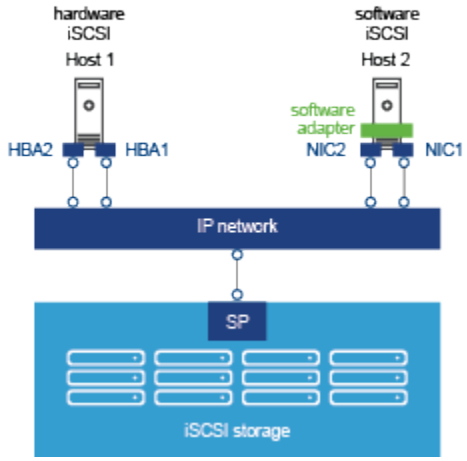
You must configure iSCSI initiators for the host to access and display iSCSI storage devices.

Figure: iSCSI Storage depicts hosts that use different types of iSCSI initiators.

- The host on the left uses an independent hardware iSCSI adapter to connect to the iSCSI storage system.
- The host on the right uses software iSCSI.

Dependent hardware iSCSI can be implemented in different ways and is not shown. iSCSI storage devices from the storage system become available to the host. You can access the storage devices and create VMFS datastores for your storage needs.

Figure 185: iSCSI Storage



Discovery Sessions

A discovery session is part of the iSCSI protocol. The discovery session returns the set of targets that you can access on an iSCSI storage system.

ESXi systems support dynamic and static discovery.

- Dynamic discovery - Also known as Send Targets discovery. Each time the ESXi host contacts a specified iSCSI storage server, it sends a Send Targets request to the server. In response, the iSCSI storage server supplies a list of available targets to the ESXi host. Monitor and manage with **`esxcli iscsi adapter discovery sendtarget`**.
- Static discovery - The ESXi host does not have to perform discovery. Instead, the ESXi host uses the IP addresses or domain names and iSCSI target names, IQN or EUI format names, to communicate with the iSCSI target. Monitor and manage with **`esxcli iscsi adapter discovery statictarget`**.

For either case, you set up target discovery addresses so that the initiator can determine which storage resource on the network is available for access. You can do this setup with dynamic discovery or static discovery. With dynamic discovery, all targets associated with an IP address or host name and the iSCSI name are discovered. With static discovery, you must specify the IP address or host name and the iSCSI name of the target you want to access. The iSCSI HBA must be in the same VLAN as both ports of the iSCSI array.

Discovery Target Names

The target name is either an IQN name or an EUI name.

The IQN and EUI names use specific formats.

- The IQN name uses the following format.
  `iqn.yyyy-mm.{reversed domain name}:id_string`

  The following IQN name contains example values.
  `iqn.2007-05.com.mydomain:storage.tape.sys3.abc`

  The ESXi host generates an IQN name for software iSCSI and dependent hardware iSCSI adapters. You can change that default IQN name.

- The EUI name is described in IETF rfc3720 as follows.
  The IEEE Registration Authority provides a service for assigning globally unique identifiers [EUI]. The EUI-64 format is used to build a global identifier in other network protocols. For example, Fibre Channel defines a method of encoding it into a `WorldWideName`.

  The format is `eui.` followed by an EUI-64 identifier (16 ASCII-encoded hexadecimal digits).

  The following EUI name contains example values.

  ```
  Type EUI-64 identifier (ASCII-encoded hexadecimal)

  +- -++---------------+

  |   ||               |

  eui.02004567A425678D
  ```

  The IEEE EUI-64 iSCSI name format can be used when a manufacturer is registered with the IEEE Registration Authority and uses EUI-64 formatted worldwide unique names for its products.

You can check in the UI of the storage array whether an array uses an IQN name or an EUI name.

Protecting an iSCSI SAN

Your iSCSI configuration is only as secure as your IP network. By enforcing good security standards when you set up your network, you help safeguard your iSCSI storage.

Protecting Transmitted Data

A primary security risk in iSCSI SANs is that an attacker might sniff transmitted storage data.

Neither the iSCSI adapter nor the ESXi host iSCSI initiator encrypts the data that it transmits to and from the targets, making the data vulnerable to sniffing attacks. You must therefore take additional measures to prevent attackers from easily seeing iSCSI data.

Allowing your virtual machines to share virtual switches and VLANs with your iSCSI configuration potentially exposes iSCSI traffic to misuse by a virtual machine attacker. To help ensure that intruders cannot listen to iSCSI transmissions, make sure that none of your virtual machines can see the iSCSI storage network.

Protect your system by giving the iSCSI SAN a dedicated virtual switch.

- If you use an independent hardware iSCSI adapter, make sure that the iSCSI adapter and ESXi physical network adapter are not inadvertently connected outside the host. Such a connection might result from sharing a switch.
- If you use dependent hardware or software iscsi adapter, which uses ESXi networking, configure iSCSI storage through a different virtual switch than the one used by your virtual machines.

You can also configure your iSCSI SAN on its own VLAN to improve performance and security. Placing your iSCSI configuration on a separate VLAN ensures that no devices other than the iSCSI adapter can see transmissions within the iSCSI SAN. With a dedicated VLAN, network congestion from other sources cannot interfere with iSCSI traffic.

Securing iSCSI Ports

You can improve the security of iSCSI ports by installing security patches and limiting the devices connected to the iSCSI network.

When you run iSCSI devices, the ESXi host does not open ports that listen for network connections. This measure reduces the chances that an intruder can break into the ESXi host through spare ports and gain control over the host. Therefore, running iSCSI does not present an additional security risks at the ESXi host end of the connection.

An iSCSI target device must have one or more open TCP ports to listen for iSCSI connections. If security vulnerabilities exist in the iSCSI device software, your data can be at risk through no fault of the ESXi system. To lower this risk, install all security patches that your storage equipment manufacturer provides and limit the devices connected to the iSCSI network.

Setting iSCSI CHAP

iSCSI storage systems authenticate an initiator using a name and key pair. ESXi systems support Challenge Handshake Authentication Protocol (CHAP).

Using CHAP for your SAN implementation is a best practice. The ESXi host and the iSCSI storage system must have CHAP enabled and must have common credentials. During iSCSI login, the iSCSI storage system exchanges its credentials with the ESXi system and checks them.

You can set up iSCSI authentication by using the vSphere Client, as discussed in the *vSphere Storage* documentation or by using the **esxcli** command, discussed in Enabling iSCSI Authentication. To use CHAP authentication, you must enable CHAP on both the initiator side and the storage system side. After authentication is enabled, it applies for targets to which no connection has been established, but does not apply to targets to which a connection is established. After the discovery address is set, the new volumes to which you add a connection are exposed and can be used.

For software iSCSI and dependent hardware iSCSI, ESXi hosts support per-discovery and per-target CHAP credentials. For independent hardware iSCSI, ESXi hosts support only one set of CHAP credentials per initiator. You cannot assign different CHAP credentials for different targets.

When you configure independent hardware iSCSI initiators, ensure that the CHAP configuration matches your iSCSI storage. If CHAP is enabled on the storage array, it must be enabled on the initiator. If CHAP is enabled, you must set up the CHAP authentication credentials on the ESXi host to match the credentials on the iSCSI storage.

Supported CHAP Levels

To set CHAP levels with **esxcli iscsi adapter setauth**, specify one of the values in Table : Supported Levels for CHAP for `<level>`. Only two levels are supported for independent hardware iSCSI.

Mutual CHAP is supported for software iSCSI and for dependent hardware iSCSI, but not for independent hardware iSCSI.

> **IMPORTANT**
> Ensure that CHAP is set to `chapRequired` before you set mutual CHAP, and use compatible levels for CHAP and mutual CHAP. Use different passwords for CHAP and mutual CHAP to avoid security risks.

Table 116: Supported Levels for CHAP

Level	Description	Supported
`chapProhibited`	Host does not use CHAP authentication. If authentication is enabled, specify `chapProhibited` to disable it.	Software iSCSI Dependent hardware iSCSI Independent hardware iSCSI
`chapDiscouraged`	Host uses a non-CHAP connection, but allows a CHAP connection as fallback.	Software iSCSI Dependent hardware iSCSI

Table continued on next page

Continued from previous page

Level	Description	Supported
`chapPreferred`	Host uses CHAP if the CHAP connection succeeds, but uses non-CHAP connections as fallback.	Software iSCSI Dependent hardware iSCSI Independent hardware iSCSI
`chapRequired`	Host requires successful CHAP authentication. The connection fails if CHAP negotiation fails.	Software iSCSI Dependent hardware iSCSI

Returning Authentication to Default Inheritance

The values of iSCSI authentication settings associated with a dynamic discovery address or a static discovery target are inherited from the corresponding settings of the parent. For the dynamic discovery address, the parent is the adapter. For the static target, the parent is the adapter or discovery address.

- If you use the vSphere Client to modify authentication settings, you must deselect the **Inherit from Parent** check box before you can make a change to the discovery address or discovery target.
- If you use **`esxcli iscsi`** commands, the value you set overrides the inherited value. You can set CHAP at the following levels.
  - `esxcli iscsi adapter auth chap [get|set]`
  - `esxcli iscsi adapter discovery sendtarget auth chap [get|set]`
  - `esxcli iscsi adapter target portal auth chap [get|set]`

Inheritance is relevant only if you want to return a dynamic discovery address or a static discovery target to its inherited value. In that case, use one of the following commands.

- Dynamic discovery
  `esxcli iscsi adapter discovery sendtarget auth chap set --inherit`
- Static discovery
  `esxcli iscsi adapter target portal auth chap set --inherit`

> **NOTE**
> You can set target-level CHAP authentication properties to be inherited from the send target level and set send target level CHAP authentication properties to be inherited from the adapter level. Resetting adapter-level properties is not supported.

Command Syntax for esxcli iscsi

You can manage iSCSI storage by using **`esxcli iscsi`** commands.

For details, see the *`ESXCLI Reference`* and esxcli iscsi Command Syntax.

esxcli iscsi Command Syntax

The **`esxcli iscsi`** command includes a number of nested namespaces.

The following table illustrates the namespace hierarchy. Commands at each level are included in bold. Many namespaces include both commands and namespaces.

Table 117: esxcli iscsi Command Overview

adapter [**get**|**list**|**set**]	auth	chap [**set**|**get**]		
	discovery [**rediscover**]	sendtarget [**add**|**list**|**remove**]		
			auth	chap [**get**|**set**]
			param [**get**|**set**]	
		statictarget [**add**|**list**|**remove**]		
		status **get**		
	target [**list**]	portal [**list**]	auth	chap [**get**|**set**]
			param [get|set]	
	capabilities **get**			
	firmware [**get**|**set**]			
	param [**get**|**set**]			
networkportal [**add**|**list**|**remove**]	ipconfig [**get**|**set**]			
physicalnetworkportal [**list**]	param [**get**|**set**]			
session [**add**|**list**|**remove**]	connection **list**			
ibftboot [**get**|**import**]				
logicalnetworkportal **list**				
plugin **list**				
software [**get**|**set**]				

Key to esxcli iscsi Short Options

ESXCLI commands for iSCSI management consistently use the same short options. For several options, the associated full option depends on the command.

Table 118: Short Options for iSCSI ESXCLI Command Options

Lower-case Option	Option	Upper-case Option	Option	Number	Option
a	--address, alias	A	--adapter	1	--dns1
c	--cid			2	--dns2

Table continued on next page

Continued from previous page

Lower-case Option	Option	Upper-case Option	Option	Number	Option
d	--direction	D	--default		
f	--file, force				
g	--gateway				
i	--ip	I	--inherit		
k	--key				
l	--level				
m	--method	M	--module		
n	--nic	N	--authname, --name		
o	--option				
p	--plugin				
s	--isid, subnet, switch	S	--state, secret		
v	--value				

iSCSI Storage Setup with ESXCLI

You can set up iSCSI storage by using commands in the **`esxcli iscsi`** namespace.

You can also set up iSCSI storage by using the vSphere Client.

Set Up Software iSCSI with ESXCLI

Software iSCSI setup requires a number of high-level tasks.

- Verify that you are familiar with iSCSI authentication. See Enabling iSCSI Authentication.
- Verify that you are familiar with CHAP. See Setting iSCSI CHAP.
- Verify that you are familiar with iSCSI parameters. See Listing and Setting iSCSI Parameters.

You should be familiar with the corresponding command for each task. You can refer to the relevant documentation for each command or run `esxcli iscsi --help` in the console. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. Enable software iSCSI.

   ```
   esxcli <conn_options> iscsi software set --enabled=true
   ```

2. Check whether a network portal, that is, a bound port, exists for iSCSI traffic.

   ```
   esxcli <conn_options> iscsi adapter list
   ```

3. If no adapter exists, add one.

   Software iSCSI does not require port binding, but requires that at least one VMkernel NIC is available and can be used as an iSCSI NIC. You can name the adapter as you add it.

   ```
   esxcli <conn_options> iscsi networkportal add -n <portal_name> -A <vmhba>
   ```

4. Check the status.

```
esxcli <conn_options> iscsi software get
```

The system prints `true` if software iSCSI is enabled, or `false` if it is not enabled.

5. Set the iSCSI name and alias.

```
esxcli <conn_options> iscsi adapter set --adapter=<iscsi adapter> --name=<name>

esxcli <conn_options> iscsi adapter set --adapter=<iscsi adapter> --alias=<alias>
```

6. Add a dynamic discovery address or a static discovery address.
   - With dynamic discovery, all storage targets associated with a host name or IP address are discovered. You can run the following command.
     ```
     esxcli <conn_options> iscsi adapter discovery sendtarget add --address=<ip/
     dns[:port]> --adapter=<adapter_name>
     ```
   - With static discovery, you must specify the host name or IP address and the iSCSI name of the storage target. You can run the following command.
     ```
     esxcli <conn_options> iscsi adapter discovery statictarget add --address=<ip/
     dns[:port]> --adapter=<adapter_name> --name=<target_name>
     ```

   When you later remove a discovery address, it might still be displayed as the parent of a static target. You can add the discovery address and rescan to display the correct parent for the static targets.

7. Set the authentication information for CHAP.

   You can set per-target CHAP for static targets, per-adapter CHAP, or apply the command to the discovery address.

Option	Command
Adapter-level CHAP	`esxcli iscsi adapter auth chap set --direction=uni --chap_username=<name> --chap_password=<pwd> --level=[prohibited, discouraged, preferred, required] --secret=<string> --adapter=<vmhba>`
Discovery-level CHAP	`esxcli iscsi adapter discovery sendtarget auth chap set --direction=uni --chap_username=<name> --chap_password=<pwd> --level=[prohibited, discouraged, preferred, required] --secret=<string> --adapter=<vmhba> --address<sendtarget_address>`
Target-level CHAP	`esxcli iscsi adapter target portal auth chap set --direction=uni --chap_username=<name> --chap_password=<pwd> --level=[prohibited, discouraged, preferred, required] --secret=<string> --adapter=<vmhba> --name<iscsi_iqn_name>`

   The following example sets adapter-level CHAP.

```
esxcli <conn_options> iscsi adapter auth chap set --direction=uni --
chap_username=<name> --chap_password=<pwd> --level=preferred --secret=uni_secret --
adapter=vmhba33
```

8. Set the authentication information for mutual CHAP by running `esxcli iscsi adapter auth chap set` again with `--direction` set to `mutual` and a different authentication user name and secret.

Option	Command
Adapter-level CHAP	`esxcli iscsi adapter auth chap set --direction=mutual --mchap_username=<name2> --mchap_password=<pwd2> --level=[prohibited required] --secret=<string2> --adapter=<vmhba>`
Discovery-level CHAP	`esxcli iscsi adapter discovery sendtarget auth chap set --direction=mutual --mchap_username=<name2> --mchap_password=<pwd2> --level=[prohibited, required] --secret=<string2> --adapter=<vmhba> --address=<sendtarget_address>`
Target-level CHAP	`esxcli iscsi adapter target portal auth chap set --direction=mutual --mchap_username=<nam2e> --mchap_password=<pwd2> --level=[prohibited required] --secret=<string2> --adapter=<vmhba> --name=<iscsi_iqn_name>`

IMPORTANT
You are responsible for making sure that CHAP is set before you set mutual CHAP, and for using compatible levels for CHAP and mutual CHAP.

9. Set iSCSI parameters.

Option	Command
Adapter-level CHAP	`esxcli iscsi adapter param set --adapter=<vmhba> --key=<key> --value=<value>`
Discovery-level CHAP	`esxcli iscsi adapter discovery sendtarget param set --adapter=<vmhba> --key=<key> --value=<value> --address=<sendtarget_address>`
Target-level CHAP	`esxcli iscsi adapter target portal param set --adapter=<vmhba> --key=<key> --value=<value> --address=<address> --name=<iqn.name>`

10. After setup is complete, perform rediscovery and rescan all storage devices.
    The following example performs the rediscovery and rescan operations.
    ```
    esxcli <conn_options> iscsi adapter discovery rediscover

    esxcli <conn_options> storage core adapter rescan --adapter=vmhba36
    ```
11. If you want to make additional iSCSI login parameter changes, you must log out of the corresponding iSCSI session and log back in.
    a) Run `esxcli iscsi session remove` to log out.
    b) Run `esxcli iscsi session add` or rescan the adapter to add the session back.

Set Up Dependent Hardware iSCSI with ESXCLI

Dependent hardware iSCSI setup requires several high-level tasks.

- Verify that you are familiar with iSCSI authentication. See Enabling iSCSI Authentication.
- Verify that you are familiar with CHAP. See Setting iSCSI CHAP.
- Verify that you are familiar with iSCSI parameters. See Listing and Setting iSCSI Parameters.

You should be familiar with the corresponding command for each task. You can refer to the relevant documentation for each command or run `esxcli iscsi --help` in the console. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. Determine the iSCSI adapter type and retrieve the iSCSI adapter ID.

   ```
   esxcli <conn_options> iscsi adapter list
   ```

2. Set the iSCSI name and alias.

   ```
   esxcli <conn_options> iscsi adapter set --adapter <adapter_name> --name=<name>
   ```

   ```
   esxcli <conn_options> iscsi adapter set --adapter <adapter_name> --alias=<alias>
   ```

3. Set up port binding.
   a) Identify the VMkernel port of the dependent hardware iSCSI adapter.

      ```
      esxcli <conn_options> iscsi logicalnetworkportal list --adapter=<adapter_name>
      ```

   b) Connect the dependent hardware iSCSI initiator to the iSCSI VMkernel ports by running the following command for each port.

      ```
      esxcli <conn_options> iscsi networkportal add --nic=<bound_vmknic> --
      adapter=<iscsi_adapter>
      ```

   c) Verify that the ports were added to the dependent hardware iSCSI initiator.

      ```
      esxcli <conn_options> iscsi physicalnetworkportal list --adapter=<adapter_name>
      ```

4. Add a dynamic discovery address or a static discovery address.
   – With dynamic discovery, all storage targets associated with a host name or IP address are discovered. You can run the following command.

     ```
     esxcli <conn_options> iscsi adapter discovery sendtarget add --address=<ip/
     dns[:port]> --adapter=<adapter_name>
     ```

   – With static discovery, you must specify the host name or IP address and the iSCSI name of the storage target. You can run the following command.

     ```
     esxcli <conn_options> iscsi adapter discovery statictarget add --address=<ip/
     dns[:port]> --adapter=<adapter_name> --name=<target_name>
     ```

   When you later remove a discovery address, it might still be displayed as the parent of a static target. You can add the discovery address and rescan to display the correct parent for the static targets.

5. Set the authentication information for CHAP.

   You can set per-target CHAP for static targets, per-adapter CHAP, or apply the command to the discovery address.

Option	Command
Adapter-level CHAP	`esxcli iscsi adapter auth chap set --direction=uni --chap_username=<name> --chap_password=<pwd> --level=[prohibited,`

Table continued on next page

Continued from previous page

Option	Command
	`discouraged, preferred, required] --secret=<string> --adapter=<vmhba>`
Discovery-level CHAP	`esxcli iscsi adapter discovery sendtarget auth chap set --direction=uni --chap_username=<name> --chap_password=<pwd> --level=[prohibited, discouraged, preferred, required] --secret=<string> --adapter=<vmhba> --address<sendtarget_address>`
Target-level CHAP	`esxcli iscsi adapter target portal auth chap set --direction=uni --chap_username=<name> --chap_password=<pwd> --level=[prohibited, discouraged, preferred, required] --secret=<string> --adapter=<vmhba> --name<iscsi_iqn_name>`

The following example sets adapter-level CHAP.

```
esxcli <conn_options> iscsi adapter auth chap set --direction=uni --
chap_username=<name> --chap_password=<pwd> --level=preferred --secret=uni_secret --
adapter=vmhba33
```

6. Set the authentication information for mutual CHAP by running `esxcli iscsi adapter auth chap set` again with `--direction` set to `mutual` and a different authentication user name and secret.

Option	Command
Adapter-level CHAP	`esxcli iscsi adapter auth chap set --direction=mutual --mchap_username=<name2> --mchap_password=<pwd2> --level=[prohibited required] --secret=<string2> --adapter=<vmhba>`
Discovery-level CHAP	`esxcli iscsi adapter discovery sendtarget auth chap set --direction=mutual --mchap_username=<name2> --mchap_password=<pwd2> --level=[prohibited, required] --secret=<string2> --adapter=<vmhba> --address=<sendtarget_address>`
Target-level CHAP	`esxcli iscsi adapter target portal auth chap set --direction=mutual --mchap_username=<nam2e> --mchap_password=<pwd2> --level=[prohibited required] --secret=<string2> --adapter=<vmhba> --name=<iscsi_iqn_name>`

IMPORTANT
You are responsible for making sure that CHAP is set before you set mutual CHAP, and for using compatible levels for CHAP and mutual CHAP.

7. Set iSCSI parameters.

Option	Command
Adapter-level CHAP	`esxcli iscsi adapter param set --adapter=<vmhba> --key=<key> --value=<value>`
Discovery-level CHAP	`esxcli iscsi adapter discovery sendtarget param set --adapter=<vmhba> --key=<key> --value=<value> --address=<sendtarget_address>`
Target-level CHAP	`esxcli iscsi adapter target portal param set --adapter=<vmhba> --key=<key> --value=<value> --address=<address> --name=<iqn.name>`

8. After setup is complete, perform rediscovery and rescan all storage devices.
   The following example performs the rediscovery and rescan operations.
   ```
   esxcli <conn_options> iscsi adapter discovery rediscover
   ```
   ```
   esxcli <conn_options> storage core adapter rescan --adapter=vmhba36
   ```
9. If you want to make additional iSCSI login parameter changes, you must log out of the corresponding iSCSI session and log back in.
   a) Run `esxcli iscsi session remove` to log out.
   b) Run `esxcli iscsi session add` or rescan the adapter to add the session back.

Set Up Independent Hardware iSCSI with ESXCLI

With independent hardware-based iSCSI storage, you use a specialized third-party adapter capable of accessing iSCSI storage over TCP/IP. This iSCSI initiator handles all iSCSI and network processing and management for your ESXi system.

- Verify that you are familiar with iSCSI authentication. See Enabling iSCSI Authentication.
- Verify that you are familiar with CHAP. See Setting iSCSI CHAP.
- Verify that you are familiar with iSCSI parameters. See Listing and Setting iSCSI Parameters.

You must install and configure the independent hardware iSCSI adapter for your host before you can access the iSCSI storage device. For installation information, see vendor documentation.
Hardware iSCSI setup requires a number of high-level tasks. You should be familiar with the corresponding command for each task. You can refer to the relevant documentation for each command or run `esxcli iscsi --help` in the console. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. Determine the iSCSI adapter type and retrieve the iSCSI adapter ID.
   ```
   esxcli <conn_options> iscsi adapter list
   ```
2. Configure the hardware initiator (HBA) by running `esxcli iscsi networkportal ipconfig` with one or more of the following options.

Option	Description
`-A|--adapter=<str>`	iSCSI adapter name (required)
`-1|--dns1=<str>`	iSCSI network portal primary DNS address
`-2|--dns2=<str>`	iSCSI network portal secondary DNS address

Table continued on next page

Continued from previous page

Option	Description
`-g|--gateway=<str>`	iSCSI network portal gateway address
`-i|--ip=<str>`	iSCSI network portal IP address (required)
`-n|--nic=<str>`	iSCSI network portal (vmknic)
`-s|--subnet=<str>`	iSCSI network portal subnet mask (required)

3. Set the iSCSI name and alias.

```
esxcli <conn_options> iscsi adapter set --adapter <adapter_name> --name=<name>

esxcli <conn_options> iscsi adapter set --adapter <adapter_name> --alias=<alias>
```

4. Add a dynamic discovery address or a static discovery address.
   - With dynamic discovery, all storage targets associated with a host name or IP address are discovered. You can run the following command.
     ```
     esxcli <conn_options> iscsi adapter discovery sendtarget add --address=<ip/
     dns[:port]> --adapter=<adapter_name>
     ```
   - With static discovery, you must specify the host name or IP address and the iSCSI name of the storage target. You can run the following command.
     ```
     esxcli <conn_options> iscsi adapter discovery statictarget add --address=<ip/
     dns[:port]> --adapter=<adapter_name> --name=<target_name>
     ```
5. Set the authentication information for CHAP.

   You can set per-target CHAP for static targets, per-adapter CHAP, or apply the command to the discovery address.

Option	Command
Adapter-level CHAP	`esxcli iscsi adapter auth chap set --direction=uni --chap_username=<name> --chap_password=<pwd> --level=[prohibited, discouraged, preferred, required] --secret=<string> --adapter=<vmhba>`
Discovery-level CHAP	`esxcli iscsi adapter discovery sendtarget auth chap set --direction=uni --chap_username=<name> --chap_password=<pwd> --level=[prohibited, discouraged, preferred, required] --secret=<string> --adapter=<vmhba> --address<sendtarget_address>`
Target-level CHAP	`esxcli iscsi adapter target portal auth chap set --direction=uni --chap_username=<name> --chap_password=<pwd> --level=[prohibited, discouraged, preferred, required] --secret=<string> --adapter=<vmhba> --name<iscsi_iqn_name>`

   The following example sets adapter-level CHAP.

```
esxcli <conn_options> iscsi adapter auth chap set --direction=uni --
chap_username=<name> --chap_password=<pwd> --level=preferred --secret=uni_secret --
adapter=vmhba33
```

NOTE
Mutual CHAP is not supported for independent hardware iSCSI storage.

6. Set iSCSI parameters.

Option	Command
Adapter-level CHAP	`esxcli iscsi adapter param set --adapter=<vmhba> --key=<key> --value=<value>`
Discovery-level CHAP	`esxcli iscsi adapter discovery sendtarget param set --adapter=<vmhba> --key=<key> --value=<value> --address=<sendtarget_address>`
Target-level CHAP	`esxcli iscsi adapter target portal param set --adapter=<vmhba> --key=<key> --value=<value> --address=<address> --name=<iqn.name>`

7. After setup is complete, run `esxcli storage core adapter rescan --adapter=<iscsi_adapter>` to rescan all storage devices.
8. After setup is complete, perform rediscovery and rescan all storage devices.
   The following example performs the rediscovery and rescan operations.
   ```
   esxcli <conn_options> iscsi adapter discovery rediscover

   esxcli <conn_options> storage core adapter rescan --adapter=vmhba36
   ```

Listing and Setting iSCSI Options

You can list and set iSCSI options with ESXCLI.

You can also manage parameters. See Listing and Setting iSCSI Parameters.

Listing iSCSI Options with ESXCLI

You can use **`esxcli iscsi`** information retrieval commands to list external HBA properties, information about targets, and LUNs.

You can use the following **`esxcli iscsi`** options to list iSCSI parameters. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

- Run **`esxcli iscsi adapter firmware`** to list or upload the firmware for the iSCSI adapter.
  ```
  esxcli <conn_options> iscsi adapter firmware get --adapter=<adapter_name>

  esxcli <conn_options> iscsi adapter firmware set --file=<firmware_file_path>
  ```
  The system returns information about the vendor, model, description, and serial number of the HBA.
- Run commands in the **`esxcli iscsi adapter target`** name space.
  - **`esxcli iscsi adapter target portal`** lists and sets authentication and portal parameters.
  - **`esxcli iscsi adapter target list`** lists LUN information.

Setting MTU with ESXCLI

You can change MTU settings by using ESXCLI.

If you want to change the MTU used for your iSCSI storage, you must make the change in two places.

- Run **`esxcli network vswitch standard set`** to change the MTU of the virtual switch.
- Run **`esxcli network ip interface set`** to change the MTU of the network interface.

Listing and Setting iSCSI Parameters

You can list and set iSCSI parameters for software iSCSI and for dependent hardware iSCSI by using ESXCLI. You can also return parameters to default inheritance.

Listing and Setting iSCSI Parameters with ESXCLI

You can list and set iSCSI parameters for software iSCSI and for dependent hardware iSCSI by using ESXCLI.

You can retrieve and set iSCSI parameters by running one of the following commands.

Parameter Type	Command
Adapter-level parameters	`esxcli iscsi adapter param set --adapter=<vmhba> --key=<key> --value=<value>`
Target-level parameters	`esxcli iscsi adapter target portal param set --adapter=<vmhba> --key=<key> --value=<value> --address=<address> --name=<iqn.name>`
Discovery-level parameters	`esxcli iscsi adapter discovery sendtarget param set --adapter=<vmhba> --key=<key> --value=<value> --address=<address>`

The following table lists all settable parameters. These parameters are also described in the IETF rfc 3720. You can run **`esxcli iscsi adapter param get`** to determine whether a parameter is settable or not.

The parameters in the table apply to software iSCSI and dependent hardware iSCSI.

Table 119: Settable iSCSI Parameters

Parameter	Description
`DataDigestType`	Increases data integrity. When data digest is enabled, the system performs a checksum over each PDUs data part and verifies using the CRC32C algorithm. **NOTE** Systems that use Intel Nehalem processors offload the iSCSI digest calculations for software iSCSI, thus reducing the impact on performance. Valid values are `digestProhibited`, `digestDiscouraged`, `digestPreferred`, or `digestRequired`.
`HeaderDigest`	Increases data integrity. When header digest is enabled, the system performs a checksum over the header part of each iSCSI Protocol Data Unit (PDU) and verifies using the CRC32C algorithm.
`MaxOutstandingR2T`	Max Outstanding R2T defines the Ready to Transfer (R2T) PDUs that can be in transition before an acknowledgement PDU is received.

Table continued on next page

Continued from previous page

Parameter	Description
`FirstBurstLength`	Maximum amount of unsolicited data an iSCSI initiator can send to the target during the execution of a single SCSI command, in bytes.
`MaxBurstLength`	Maximum SCSI data payload in a Data-In or a solicited Data-Out iSCSI sequence, in bytes.
`MaxRecvDataSegLen`	Maximum data segment length, in bytes, that can be received in an iSCSI PDU.
`NoopOutInterval`	Time interval, in seconds, between NOP-Out requests sent from your iSCSI initiator to an iSCSI target. The NOP-Out requests serve as the ping mechanism to verify that a connection between the iSCSI initiator and the iSCSI target is active. Supported only at the initiator level.
`NoopOutTimeout`	Amount of time, in seconds, that can lapse before your host receives a NOP-In message. The message is sent by the iSCSI target in response to the NOP-Out request. When the `NoopTimeout` limit is exceeded, the initiator terminates the current session and starts a new one. Supported only at the initiator level.
`RecoveryTimeout`	Amount of time, in seconds, that can lapse while a session recovery is performed. If the timeout exceeds its limit, the iSCSI initiator terminates the session.
`DelayedAck`	Allows systems to delay acknowledgment of received data packets.

You can use the following ESXCLI commands to list parameter options.

- Run **`esxcli iscsi adapter param get`** to list parameter options for the iSCSI adapter.
- Run **`esxcli iscsi adapter discovery sendtarget param get`** or **`esxcli iscsi adapter target portal param set`** to retrieve information about iSCSI parameters and whether they are settable.
- Run **`esxcli iscsi adapter discovery sendtarget param get`** or **`esxcli iscsi adapter target portal param set`** to set iSCSI parameter options.

If special characters are in the `<name>=<value>` sequence, for example, if you add a space, you must surround the sequence with double quotes (`"<name> = <value>"`).

Returning Parameters to Default Inheritance with ESXCLI

The values of iSCSI parameters associated with a dynamic discovery address or a static discovery target are inherited from the corresponding settings of the parent.

For the dynamic discovery address, the parent is the adapter. For the static target, the parent is the adapter or discovery address.

- If you use the vSphere Client to modify authentication settings, you must deselect the **Inherit from Parent** check box before you can make a change to the discovery address or discovery target.
- If you use **`esxcli iscsi`**, the value you set overrides the inherited value.

Inheritance is relevant only if you want to return a dynamic discovery address or a static discovery target to its inherited value. In that case, use the following command, which requires the `--name` option for static discovery addresses, but not for dynamic discovery targets.

Target Type	Command
Dynamic target	`esxcli iscsi adapter discovery sendtarget param set`
Static target	`esxcli iscsi adapter target portal param set`

Enabling iSCSI Authentication

You can enable iSCSI authentication by using ESXCLI.

Enable iSCSI Authentication with ESXCLI

You can use the **`esxcli iscsi adapter auth`** commands to enable iSCSI authentication.

For information on iSCSI CHAP, see Setting iSCSI CHAP.

1. Set the authentication information for CHAP.

   ```
   esxcli <conn_options> iscsi adapter auth chap set --direction=uni --
   chap_username=<name> --chap_password=<pwd> --level=[prohibited, discouraged,
   preferred, required] --secret=<string> --adapter=<adapter_name>
   ```

   You can set per-target CHAP for static targets, per-adapter CHAP, or apply the command to the discovery address.

Option	Command
Per-adapter CHAP	`esxcli iscsi adapter auth chap set`
Per-discovery CHAP	`esxcli iscsi adapter discovery sendtarget auth chap set`
Per-target CHAP	`esxcli iscsi adapter target portal auth chap set`

   The following example sets adapter-level CHAP.

   ```
   esxcli <conn_options> iscsi adapter auth chap set --direction=uni --
   chap_username=User1 --chap_password=MySpecialPwd --level=preferred --
   secret=uni_secret --adapter=vmhba33
   ```

2. Set the authentication information for mutual CHAP by running **`esxcli iscsi adapter auth chap`** set again with the `-d` option set to `mutual option` and a different authentication user name and secret.

   ```
   esxcli <conn_options> iscsi adapter auth chap set --direction=mutual --
   mchap_username=<m_name> --mchap_password=<m_pwd> --level=[prohibited, required] --
   secret=<string> --adapter=<adapter_name>
   ```

   For `<level>`, specify `prohibited` or `required`.

Option	Description
`prohibited`	The host does not use CHAP authentication. If authentication is enabled, specify `chapProhibited` to disable it.

Table continued on next page

Continued from previous page

Option	Description
`required`	The host requires successful CHAP authentication. The connection fails if CHAP negotiation fails. You can set this value for mutual CHAP only if CHAP is set to `chapRequired`.

For direction, specify `mutual`.

IMPORTANT
You are responsible for making sure that CHAP is set before you set mutual CHAP, and for using compatible levels for CHAP and mutual CHAP. Use a different secret in CHAP and mutual CHAP.

Enable Mutual iSCSI Authentication with ESXCLI

Mutual authentication is supported for software iSCSI and dependent hardware iSCSI, but not for independent hardware iSCSI.

- Verify that CHAP authentication is already set up when you start setting up mutual CHAP.
- Verify that CHAP and mutual CHAP use different user names and passwords. The second user name and password are supported for mutual authentication on the storage side.
- Verify that CHAP and mutual CHAP use compatible CHAP levels.

For information on iSCSI CHAP, see Setting iSCSI CHAP.

1. Enable authentication.

   ```
   esxcli <conn_options> iscsi adapter auth chap set --direction=uni --
   chap_username=<name> --chap_password=<pw> --level=[prohibited, discouraged,
   preferred, required] --secret=<string> --adapter=<adapter_name>
   ```

   The specified `chap_username` and `secret` must be supported on the storage side.

2. List possible VMkernel NICs to bind.

   ```
   esxcli <conn_options> iscsi logicalnetworkportal list
   ```

3. Enable mutual authentication.

   ```
   esxcli <conn_options> iscsi adapter auth chap set --direction=mutual --
   mchap_username=<m_name> --mchap_password=<m_pwd> --level=[prohibited, required] --
   secret=<string> --adapter=<adapter_name>
   ```

   The specified `mchap_username` and `secret` must be supported on the storage side.

4. After setup is complete, perform rediscovery and rescan all storage devices.
   The following example performs the rediscovery and rescan operations.

   ```
   esxcli <conn_options> iscsi adapter discovery rediscover

   esxcli <conn_options> storage core adapter rescan --adapter=vmhba36
   ```

Set Up Ports for iSCSI Multipathing

With port binding, you create a separate VMkernel port for each physical NIC using 1:1 mapping.

Verify that you are familiar with iSCSI session removal. See Removing iSCSI Sessions.

You can add all network adapter and VMkernel port pairs to a single vSwitch. The *vSphere Storage* documentation explains in detail how to specify port binding.
In the examples below, specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. Find out which uplinks are available for use with iSCSI adapters.

   ```
   esxcli <conn_options> iscsi physicalnetworkportal list --adapter=<adapter_name>
   ```

2. Connect the software iSCSI or dependent hardware iSCSI initiator to the iSCSI VMkernel ports by running the following command for each port.

   ```
   esxcli <conn_options> iscsi networkportal nic add --adapter=<adapter_name> --
   nic=<bound_nic>
   ```

3. Verify that the ports were added to the iSCSI initiator by running the following command.

   ```
   esxcli <conn_options> iscsi networkportal list --adapter=<adapter_name>
   ```

4. If there are active iSCSI sessions between your host and targets, discontinue them. See Removing iSCSI Sessions.
5. Rescan the iSCSI initiator.

   ```
   esxcli <conn_options> storage core adapter rescan --adapter <iscsi adapter>
   ```

6. To disconnect the iSCSI initiator from the ports, run the following command.

   ```
   esxcli <conn_options> iscsi networkportal remove --adapter=<adapter_name> --
   nic=<bound_nic>
   ```

Managing iSCSI Sessions

To communicate with each other, iSCSI initiators and targets establish iSCSI sessions. You can use **`esxcli iscsi session`** to list and manage iSCSI sessions for software iSCSI and dependent hardware iSCSI.

Introduction to iSCSI Session Management

By default, software iSCSI and dependent hardware iSCSI initiators start one iSCSI session between each initiator port and each target port.

If your iSCSI initiator or target has more than one port, your host can establish multiple sessions. The default number of sessions for each target equals the number of ports on the iSCSI adapter times the number of target ports. You can display all current sessions to analyze and debug them. You might add sessions to the default for several reasons.

- Cloning sessions - Some iSCSI arrays support multiple sessions between the iSCSI adapter and target ports. If you clone an existing session on one of these arrays, the array presents more data paths for your adapter. Duplicate sessions do not persist across reboot. Additional sessions to the target might have performance benefits, but the result of cloning depends entirely on the array. You must log out from an iSCSI session if you want to clone a session. You can use the **`esxcli iscsi session add`** command to clone a session.
- Enabling Header and Data Digest - If you are logged in to a session and want to enable the `Header` and `Data Digest` parameters, you must set the parameter, remove the session, and add the session back for the parameter change to take effect. You must log out from an iSCSI session if you want to clone a session.
- Establishing target-specific sessions - You can establish a session to a specific target port. This can be useful if your host connects to a single-port storage system that, by default, presents only one target port to your initiator, but can redirect additional sessions to a different target port. Establishing a new session between your iSCSI initiator and another target port creates an additional path to the storage system.

CAUTION
Some storage systems do not support multiple sessions from the same initiator name or endpoint. Attempts to create multiple sessions to such targets can result in unpredictable behavior of your iSCSI environment.

Listing iSCSI Sessions

You can use **`esxcli iscsi session`** to list sessions.

The following example scenario uses the available commands. Run `esxcli iscsi session --help` and each command with `--help` for reference information. The example uses a configuration file to log in to the host. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

- List a software iSCSI session at the adapter level.
  ```
  esxcli <conn_options> iscsi session list --adapter=<iscsi_adapter>
  ```
- List a software iSCSI session at the target level.
  ```
  esxcli <conn_options> iscsi session list --name=<target> --adapter=<iscsi_adapter>
  ```

Logging in to iSCSI Sessions

You can use **`esxcli iscsi session`** to log in to a session.

Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

- Log in to a session on the current software iSCSI or dependent hardware iSCSI configuration at the adapter level.
  ```
  esxcli <conn_options> iscsi session add --adapter=<adapter_name>
  ```
  The following example applies custom values.
  ```
  esxcli --config /host-config-file iscsi session add --adapter=vmhba36
  ```
- Log in to a session on the current software iSCSI or dependent hardware iSCSI configuration at the target level.
  ```
  esxcli <conn_options> iscsi session add --name=<target> --adapter=<adapter_name>
  ```
  The following example applies custom values.
  ```
  esxcli --config /host-config-file iscsi session add -name=iqn.xxx --adapter=vmhba36
  ```
- Add duplicate sessions with target and session IDs in current software iSCSI or dependent hardware iSCSI configuration.
  ```
  esxcli <conn_options> iscsi session add --name=<iqn.xxxx> --isid=<session_id> --
  adapter=<iscsi_adapter>
  ```
  `iqn.xxxx` is the target IQN, which you can determine by listing all sessions. `session_id` is the session's iSCSI ID. The following example applies custom values.
  ```
  esxcli --config /host-config-file iscsi session add -name=iqn.xxx --
  isid='00:02:3d:00:00:01' --adapter=vmhba36
  ```

Removing iSCSI Sessions

You can use **`esxcli iscsi session`** to remove iSCSI sessions.

Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

- Remove sessions from the current software iSCSI or dependent hardware iSCSI configuration at the adapter level.
  ```
  esxcli <conn_options> iscsi session remove --adapter=<iscsi_adapter>
  ```
  The following example applies custom values.

```
esxcli iscsi session remove --adapter=vmhba36
```

- Remove sessions from the current software iSCSI or dependent hardware iSCSI configuration at the target level.
  ```
  esxcli <conn_options> iscsi session remove --name=<iqn> --adapter=<iscsi_adapter>
  ```
  The following example applies custom values.
  ```
  esxcli <conn_options> iscsi session remove --name=iqn.xxx --adapter=vmhba38
  ```
- Remove sessions from the current software iSCSI or dependent hardware iSCSI configuration with target and session ID.
  ```
  esxcli <conn_options> iscsi session remove --name=<iqn.xxxx> --isid=<session id> --
  adapter=<iscsi_adapter>
  ```
  `iqn.xxxx` is the target IQN, which you can determine by listing all sessions. `session_id` is the session's iSCSI ID. The following example applies custom values.
  ```
  esxcli --config /host-config-file iscsi session remove --name=iqn.xxx --
  session='00:02:3d:01:00:01' --adapter=vmhba36
  ```

Managing Third-Party Storage Arrays

VMware partners and customers can optimize performance of their storage arrays in conjunction with VMware vSphere by using VMware PSA (pluggable storage architecture). The **`esxcli storage core`** namespace manages VMware PSA and the **`esxcli storage nmp`** namespace manages the VMware NMP plug-in.

The *`vSphere Storage`* documentation discusses PSA functionality in detail and explains how to use the vSphere Client to manage the PSA, the associated native multipathing plug-in (NMP), and third-party plug-ins.

This chapter uses the following acronyms.

Acronym	Meaning
PSA	Pluggable Storage Architecture
NMP	Native Multipathing Plug-in. Generic VMware multipathing module.
PSP	Path Selection Plug-in. Handles path selection for a given device.
SATP	Storage Array Type Plug-in. Handles path failover for a given storage array.

Managing NMP with esxcli storage nmp

The NMP is an extensible multipathing module that ESXi supports by default. You can use **`esxcli storage nmp`** to manage devices associated with NMP and to set path policies.

The NMP supports all storage arrays listed on the VMware storage Hardware Compatibility List (HCL) and provides a path selection algorithm based on the array type. The NMP associates a set of physical paths with a storage device (LUN). An SATP determines how path failover is handled for a specific storage array. A PSP determines which physical path is used to issue an I/O request to a storage device. SATPs and PSPs are plug-ins within the NMP.

Device Management with esxcli storage nmp device

The device option performs operations on devices currently claimed by the VMware NMP.

esxcli storage nmp device list

The **list** command lists the devices controlled by VMware NMP and shows the SATP and PSP information associated with each device. To show the paths claimed by NMP, run **esxcli storage nmp path list** to list information for all devices, or for just one device with the `--device` option.

Options	Description

`--device <device>`
 	
`-d <device>`
	Filters the output of the command to show information about a single device. Default is all devices.

esxcli storage nmp device set

The **set** command sets the PSP for a device to one of the policies loaded on the system.

Any device can use the PSP assigned to the SATP handling that device, or you can run `esxcli storage nmp device set --device naa.xxx --psp <psp>` to specifically override the PSP assigned to the device.

- If a device does not have a specific PSP set, it always uses the PSP assigned to the SATP. If the default PSP for the SATP changes, the PSP assigned to the device changes only after reboot or after a device is reclaimed. A device is reclaimed when you unclaim all paths for the device and reclaim the paths.
- If you use **esxcli storage nmp device set** to override the SATPs default PSP with a specific PSP, the PSP changes immediately and remains the user-defined PSP across reboots. A change in the SATP's PSP has no effect.

Use the `--default` option to return the device to using the SATP's PSP.

Options	Description

`--default`
 	
`-E`
	Sets the PSP back to the default for the SATP assigned to this device.

`--device <device>`
 	
`-d <device>`
	Device to set the PSP for.

`--psp <PSP>`
 	
`-P <PSP>`
	PSP to assign to the specified device. Call **esxcli storage nmp psp list** to display all currently available PSPs. See Managing Path Policies.
	
	See *vSphere Storage* for a discussion of path policies.

To set the path policy for the specified device to `VMW_PSP_FIXED`, run the following command.

```
esxcli <conn_options> storage nmp device set --device naa.xxx --psp VMW_PSP_FIXED
```

Listing Paths with esxcli storage nmp path

You can use the `path` option to list paths claimed by NMP.

By default, the command displays information about all paths on all devices. You can filter in the following ways.

- Only show paths to a singe device.
  `esxcli storage nmp path list --device <device>`
- Only show information for a single path.

```
esxcli storage nmp path list --path=<path>
```

To list devices, call **`esxcli storage nmp device list`**.

Managing Path Selection Policy Plug-Ins with esxcli storage nmp psp

You can use **`esxcli storage nmp psp`** to manage VMware path selection policy plug-ins included with the VMware NMP and to manage third-party PSPs.

> **IMPORTANT**
> When used with third-party PSPs, the syntax depends on the third-party PSP implementation.

Retrieving PSP Information

The **`esxcli storage nmp psp generic deviceconfig get`** and **`esxcli storage nmp psp generic pathconfig get`** commands retrieve PSP configuration parameters. The type of PSP determines which command to use.

- Use **`nmp psp generic deviceconfig get`** for PSPs that are set to `VMW_PSP_RR`, `VMW_PSP_FIXED` or `VMW_PSP_MRU`.
- Use **`nmp psp generic pathconfig get`** for PSPs that are set to `VMW_PSP_FIXED` or `VMW_PSP_MRU`. No path configuration information is available for `VMW_PSP_RR`.

To retrieve PSP configuration parameters, use the appropriate command for the PSP.

- Device configuration information.
  ```
  esxcli <conn_options> storage nmp psp generic deviceconfig get --device=<device>
  ```
  ```
  esxcli <conn_options> storage nmp psp fixed deviceconfig get --device=<device>
  ```
  ```
  esxcli <conn_options> storage nmp psp roundrobin deviceconfig get --device=<device>
  ```
- Path configuration information.
  ```
  esxcli <conn_options> storage nmp psp generic pathconfig get --path=<path>
  ```
- Retrieve the PSP configuration for the specified path.
  ```
  esxcli <conn_options> nmp psp pathconfig generic get --path vmhba4:C1:T2:L23
  ```

The **`esxcli storage nmp psp list`** command shows the list of PSPs on the system and a brief description of each plug-in.

Setting Configuration Parameters for Third-Party Extensions

The **`esxcli storage nmp psp generic deviceconfig set`** and **`esxcli storage nmp psp generic pathconfig set`** commands support future third-party PSA expansion. The **`setconfig`** command sets PSP configuration parameters for those third-party extensions.

> **NOTE**
> The precise results of these commands depend on the third-party extension. See the extension documentation for information.

Use **`esxcli storage nmp roundrobin setconfig`** for other path policy configuration. See Customizing Round Robin Setup.

You can run `esxcli storage nmp psp generic deviceconfig set --device=<device>` to specify PSP information for a device, and `esxcli storage nmp psp generic pathconfig set --path=<path>` to specify PSP information for a path. For each command, use `--config` to set the specified configuration string.

Options	Description
--config <config_string> -c <config_string>	Configuration string to set for the device or path specified by --device or --path. See Managing Path Policies.
--device <device> -d <device>	Device for which you want to customize the path policy.
--path <path> -p <path>	Path for which you want to customize the path policy.

Fixed Path Selection Policy Operations

The `fixed` option gets and sets the preferred path policy for NMP devices configured to use `VMW_PSP_FIXED`.

Retrieving the Preferred Path

The **`esxcli storage nmp fixed deviceconfig get`** command retrieves the preferred path on a specified device that is using NMP and the `VMW_PSP_FIXED` path policy.

Options	Description
- -device <device> -d <device>	Device for which you want to get the preferred path. This device must be controlled by the `VMW_PSP_FIXED` PSP.

To return the path configured as the preferred path for the specified device, run the following command. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

```
esxcli <conn_options> storage nmp fixed deviceconfig get --device naa.xxx
```

Setting the Preferred Path

The **`esxcli storage nmp fixed deviceconfig set command`** sets the preferred path on a specified device that is using NMP and the `VMW_PSP_FIXED` path policy.

Options	Description
--device <device> -d <device>	Device for which you want to set the preferred path. This device must be controlled by the `VMW_PSP_FIXED` PSP. Use `esxcli storage nmp device --list` to list the policies for all devices.
--path <path> -p <path>	Path to set as the preferred path for the specified device.

To set the preferred path for the specified device to `vmhba3:C0:T5:L3`, run the following command. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

```
esxcli <conn_options> storage nmp fixed deviceconfig set --device naa.xxx --path
vmhba3:C0:T5:L3
```

Customizing Round Robin Setup

You can use the **`esxcli storage nmp psp roundrobin`** commands to set round robin path options on a device controlled by the `VMW_PSP_RR` PSP.

Specifying and Customizing Round Robin Path Policies

You can use **`esxcli storage nmp`** commands to set path policies. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. Set the path policy to round robin.
   ```
   esxcli <conn_options> storage nmp device set --device naa.xxx --psp VMW_PSP_RR
   ```
2. Specify when to switch paths.
   You can choose the number of I/O operations, number of bytes, and so on. The following example sets the device specified by `--device` to switch to the next path each time 12345 bytes have been sent along the current path.

   ```
   esxcli <conn_options> storage nmp psp roundrobin deviceconfig set --type "bytes" -B
   12345 --device naa.xxx
   ```

   The following example sets the device specified by `--device` to switch after 4200 I/O operations have been performed on a path.

   ```
   esxcli <conn_options> storage nmp psp roundrobin deviceconfig set --type=iops --iops
   4200 --device naa.xxx
   ```

Retrieving Path Selection Settings

The **`esxcli storage nmp psp roundrobin deviceconfig get`** command retrieves path selection settings for a device that is using the `roundrobin` PSP. You can specify the device to retrieve the information for.

Options	Description

`-d <device>`
 	
`--device <device>`
	Device to get round robin properties for.

Specifying Conditions for Path Changes

The **`esxcli storage nmp psp roundrobin deviceconfig set`** command specifies under which conditions a device that is using the `VMW_PSP_RR` PSP changes to a different path. You can use `--bytes` or `--iops` to specify when the path should change.

Options	Description

`--bytes`
 	
`-B`
	Number of bytes to send along one path for this device before the PSP switches to the next path. You can use this option only when `--type` is set to bytes.

`--device`
 	
`-d`
	Device to set round robin properties for. This device must be controlled by the round robin (`VMW_PSP_RR`) PSP.

`--iops`
 	
`-I`
	Number of I/O operations to send along one path for this device before the PSP switches to the next path. You can use this option only when `--type` is set to `iops`.

Table continued on next page

Continued from previous page

Options	Description

`--type`
 	
`-t`
	Type of round robin path switching to enable for this device. The following values for `type` are supported.
	
	• `bytes`: Sets the trigger for path switching based on the number of bytes sent down a path.
	• `default`: Sets the trigger for path switching back to default values.
	• `iops`: Sets the trigger for path switching based on the number of I/O operations on a path.
	An equal sign (=) before the type or double quotes around the type are optional.

`--useANO`
 	
`-U`
	If set to 1, the round robin PSP includes paths in the active, unoptimized state in the round robin set. If set to 0, the PSP uses active, unoptimized paths only if no active optimized paths are available. Otherwise, the PSP includes only active optimized paths in the round robin path set.

Managing SATPs

The **`esxcli storage nmp satp`** commands manage SATPs.

You can use these commands to perform the following tasks.

- Retrieve and set configuration parameters.
- Add and remove rules from the list of claim rules for a specified SATP.
- Set the default PSP for a specified SATP.
- List SATPs that are currently loaded into NMP and the associated claim rules.

The default SATP for an active-active FC array with a vendor and model not listed in the SATP rules is `VMW_SATP_DEFAULT_AA`.

Retrieving Information About SATPs

The **`esxcli storage nmp satp list`** command lists the SATPs that are currently available to the NMP system and displays information about those SATPs. This command supports no options and displays information about these SATPs.

```
esxcli <conn_options> storage nmp satp list
```

The rule list command lists the claim rules for SATPs.

```
esxcli <conn_options> storage nmp satp rule list
```

Adding SATP Rules

Claim rules specify that a storage device that uses a certain driver or transport or has a certain vendor or model should use a certain SATP. The **`esxcli storage nmp satp rule add`** command adds a rule that performs such a mapping to the list of claim rules. The options you specify define the rule. For example, the following command specifies that if a path has vendor `VMWARE` and model `Virtual`, the PSA assigns it to the `VMW_SATP_LOCAL` SATP.

```
esxcli <conn_options> storage nmp satp rule add --satp="VMW_SATP_LOCAL" --vendor="VMWARE"
--model="Virtual" --description="VMware virtual disk"
```

Option	Description

`--driver`
 	
`-D`
	Driver string to set when adding the SATP claim rule.

`--device`
 	
`-d`
	Device to set when adding SATP claim rules. Device rules are mutually exclusive with vendor/model and driver rules.

`--force`
 	
`-f`
	Force claim rules to ignore validity checks and install the rule even if checks fail.

`--model`
 	
`-M`
	Model string to set when adding the SATP claim rule. Can be the model name or a pattern `^mod*`, which matches all devices that start with `mod`. That is, the pattern successfully matches `mod1` and `modz`, but not `mymod1`.
	
	The command supports the start/end (^) and wildcard (*) functionality but no other regular expressions.

`--transport`
 	
`-R`
	Transport string to set when adding the SATP claim rule. Describes the type of storage HBA, for example, `iscsi` or `fc`.

`--vendor`
 	
`-V`
	Vendor string to set when adding the SATP claim rule.

`--satp`
 	
`-s`
	SATP for which the rule is added.

`--claim-option`
 	
`-c`
	Claim option string to set when adding the SATP claim rule.

`--description`
 	
`-e`
	Description string to set when adding the SATP claim rule.

`--option`
 	
`-o`
	Option string to set when adding the SATP claim rule. Surround the option string in double quotes, and use a space, not a comma, when specifying more than one option.
	`"enable_local enable_ssd"`

`--psp`
 	
`-P`
	Default PSP for the SATP claim rule.

`--psp-option`
 	
`-O`
	PSP options for the SATP claim rule.

`--type`
 	
`-t`
	Set the claim type when adding a SATP claim rule.

The following examples illustrate adding SATP rules. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

- Add an SATP rule that specifies that disks with vendor string `VMWARE` and model string `Virtual` should be added to `VMW_SATP_LOCAL`.
  ```
  esxcli <conn_options> storage nmp satp rule add --satp="VMW_SATP_LOCAL" --
  vendor="VMWARE" --model="Virtual" --description="VMware virtual disk"
  ```
- Add an SATP rule that specifies that disks with the driver string `somedriver` should be added to `VMW_SATP_LOCAL`.
  ```
  esxcli <conn_options> storage nmp satp rule add --satp="VMW_SATP_LOCAL" --
  driver="somedriver"
  ```
- Add a rule that specifies that all storage devices with vendor string `ABC` and a model name that starts with `120` should use `VMW_SATP_DEFAULT_AA`.
  ```
  esxcli <conn_options> storage nmp satp rule add --satp VMW_SATP_DEFAULT_AA --
  vendor="ABC" --model="^120*
  ```

Removing SATP Rules

The **`esxcli storage nmp satp rule remove`** command removes an existing SATP rule. The options you specify define the rule to remove. The options listed for Adding SATP Rules are supported.

The following example removes the rule that assigns devices with vendor string `VMWARE` and model string `Virtual` to `VMW_SATP_LOCAL`.

```
esxcli <conn_options> storage nmp satp rule remove --satp="VMW_SATP_LOCAL" --
vendor="VMWARE" --model="Virtual"
```

Retrieving and Setting SATP Configuration Parameters

The **`esxcli storage nmp satp generic deviceconfig get`** and **`esxcli storage nmp satp generic pathconfig get`** commands retrieve per-device or per-path SATP configuration parameters. You cannot retrieve paths or devices for all SATPs, you must retrieve the information one path or one device at a time.

Use the following command to retrieve per device or per path SATP configuration parameters, and to see whether you can set specific configuration parameters for a device or path.

For example, `esxcli storage nmp satp generic deviceconfig get --device naa.xxx` might return `SATP VMW_SATP_LSI does not support device configuration.`

`esxcli storage nmp satp generic pathconfig get -path vmhba1:C0:T0:L8 might return INIT,AVT OFF,v5.4,DUAL ACTIVE,ESX FAILOVER`

The **`esxcli storage nmp satp generic deviceconfig set`** and **`esxcli storage nmp satp generic pathconfig set`** commands set configuration parameters for SATPs that are loaded into the system, if they support device configuration. You can set per-path or per-device SATP configuration parameters.

> **IMPORTANT**
> The command passes the configuration string to the SATP associated with that device or path.

The configuration strings might vary by SATP. VMware supports a fixed set of configuration strings for a subset of its SATPs. The strings might change in future releases.

Options	Description

`--config`
 	
`-c`
	Configuration string to set for the path specified by `--path` or the device specified by `--device`.

Table continued on next page

Continued from previous page

Options	Description
	You can set the configuration for the following SATPs. • `VMW_SATP_ALUA_CX` • `VMW_SATP_ALUA` • `VMW_SATP_CX` • `VMW_SATP_INV` You can specify one of the following device configuration strings. • `navireg_on` – starts automatic registration of the device with Navisphere. • `navireg_off` – stops the automatic registration of the device. • `ipfilter_on` – stops the sending of the host name for Navisphere registration. Used if host is known as `localhost`. • `ipfilter_off` – enables the sending of the host name during Navisphere registration.
`--device` `-d`	Device to set SATP configuration for. Not all SATPs support the setconfig option on devices.
`--path` `-p`	Path to set SATP configuration for. Not all SATPs support the setconfig option on paths.

Run `esxcli storage nmp device set --default --device=<device>` to set the PSP for the specified device back to the default for the assigned SATP for this device.

Path Claiming with esxcli storage core claiming

The **`esxcli storage core claiming`** namespace includes a number of troubleshooting commands.

These commands are not persistent and are useful only to developers who are writing PSA plug-ins or troubleshooting a system. If I/O is active on the path, unclaim and reclaim actions fail.

> **IMPORTANT**
> The help for **`esxcli storage core claiming`** includes the **`autoclaim`** command. Do not use this command unless instructed to do so by VMware support staff.

Using the Reclaim Troubleshooting Command

The **`esxcli storage core claiming reclaim`** troubleshooting command is intended for PSA plug-in developers or administrators who troubleshoot PSA plug-ins.

The command performs the following tasks.

- Attempts to unclaim all paths to a device.
- Runs the loaded claim rules on each of the unclaimed paths to reclaim those paths.

It is normal for this command to fail if a device is in use.

IMPORTANT
The reclaim command unclaims paths associated with a device.
You cannot use the command to reclaim paths currently associated with the `MASK_PATH` plug-in because `--device` is the only option for reclaim and `MASK_PATH` paths are not associated with a device.

You can use the command to unclaim paths for a device and have those paths reclaimed by the `MASK_PATH` plug-in.

Options	Description

`--device <device>`
 	
`-d <device>`
	Name of the device on which all paths are reclaimed.

`--help`	Displays the help message.

Unclaiming Paths or Sets of Paths

The **`esxcli storage core claiming unclaim`** command unclaims a path or set of paths, disassociating those paths from a PSA plug-in. The command fails if the device is in use.

You can unclaim only active paths with no outstanding requests. You cannot unclaim the ESXi USB partition or devices with VMFS volumes on them. It is therefore normal for this command to fail, especially when you specify a plug-in or adapter to unclaim.

Unclaiming does not persist. Periodic path claiming reclaims unclaimed paths unless claim rules are configured to mask a path. See the *`vSphere Storage`* documentation for details.

IMPORTANT
The **`unclaim`** command unclaims paths associated with a device. You can use this command to unclaim paths associated with the `MASK_PATH` plugin but cannot use the `--device` option to unclaim those paths.

Options	Description

`--adapter <adapter>`
 	
`-A <adapter>`
	If `--type` is set to `location`, specifies the name of the HBA for the paths that you want to unclaim. If you do not specify this option, unclaiming runs on paths from all adapters.

`--channel <channel>`
 	
`-C <channel>`
	If `--type` is set to `location`, specifies the SCSI channel number for the paths that you want to unclaim. If you do not specify this option, unclaiming runs on paths from all channels.

`--claimrule-class <cl>`
 	
`-c <cl>`
	Claim rule class to use in this operation. You can specify `MP` (Multipathing), `Filter`, or `VAAI`. `Multipathing` is the default. `Filter` is used only for `VAAI`. Specify claim rules for both `VAAI_FILTER` and `VAAI` plug-in to use it.

`--device <device>`
 	
`-d <device>`
	If `--type` is set to `device`, attempts to unclaim all paths to the specified device. If there are active I/O operations on the specified device, at least one path cannot be unclaimed.

`--driver <driver>`	If `--type` is `driver`, unclaims all paths specified by this HBA driver.

Table continued on next page

Continued from previous page

Options	Description
-D <driver>	
--lun <lun_number> -L <lun_number>	If --type is location, specifies the SCSI LUN for the paths to unclaim. If you do not specify --lun, unclaiming runs on paths with any LUN number.
--model <model> -m <model>	If --type is vendor, attempts to unclaim all paths to devices with specific model information (for multipathing plug-ins) or unclaim the device itself (for filter plug-ins). If there are active I/O operations on this device, at least one path fails to unclaim.
--path <path> -p <path>	If --type is path, unclaims a path specified by its path UID or runtime name.
--plugin <plugin> -P	If --type is plugin, unclaims all paths for a specified multipath plug-in. <plugin> can be any valid PSA plug-in on the system. By default, only NMP and MASK_PATH are available, but additional plugi-ns might be installed.
--target <target> -T <target>	If --type is location, unclaims the paths with the SCSI target number specified by target. If you do not specify --target, unclaiming runs on paths from all targets.
--type <type> -t <type>	Type of unclaim operation to perform. Valid values are location, path, driver, device, plugin, and vendor.
--vendor <vendor> -v <vendor>	If --type is vendor, attempts to unclaim all paths to devices with specific vendor info for multipathing plug-ins or unclaim the device itself for filter plug-ins. If there are any active I/O operations on this device, at least one path fails to unclaim

The following troubleshooting command tries to unclaim all paths on vmhba1.

```
esxcli <conn_options> storage core claiming unclaim --type location -A vmhba1
```

If a path is the last path to a device that was in use, or a if a path was very recently in use, the unclaim operation might fail. An error is logged that not all paths could be unclaimed. You can stop processes that might use the device and wait 15 seconds to let the device be quiesced, then retry the command.

Managing Claim Rules

The PSA uses claim rules to determine which multipathing module should claim the paths to a particular device and to manage the device. **esxcli storage core claimrule** manages claim rules.

Change the Current Claim Rules in the VMkernel

Claim rule modification commands do not operate on the VMkernel directly. Instead, they operate on the configuration file by adding and removing rules.

Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of <conn_options>.

1. Run one or more of the **`esxcli storage core claimrule`** modification commands.
   For example, **`add`**, **`remove`**, or **`move`**.
2. Run **`esxcli storage core claimrule load`** to replace the current rules in the VMkernel with the modified rules from the configuration file.

You can also run **`esxcli storage core plugin list`** to list all loaded plug-ins.

Adding Claim Rules

The **`esxcli storage core claimrule add`** command adds a claim rule to the set of claim rules on the system.

You can use this command to add new claim rules or to mask a path using the `MASK_PATH` claim rule. You must load the rules after you add them.

Options	Description
`--adapter <adapter>` -A <adapter>	Adapter of the paths to use. Valid only if `--type` is `location`.

`--autoassign`
 	
`-u`
	Adds a claim rule based on its characteristics. The rule number is not required.

`--channel <channel>`
 	
`-C <channel>`
	Channel of the paths to use. Valid only if `--type` is `location`.

`--claimrule-class <cl>`
 	
`-c <cl>`
	Claim rule class to use in this operation. You can specify `MP` (default), `Filter`, or `VAAI`.
	
	To configure hardware acceleration for a new array, add two claim rules, one for the VAAI filter and another for the VAAI plug-in. See *`vSphere Storage`* for detailed instructions.

`--driver <driver>`
 	
`-D <driver>`
	Driver for the HBA of the paths to use. Valid only if `--type` is `driver`.

`--force`
 	
`-f`
	Force claim rules to ignore validity checks and install the rule.

`--lun <lun_number>`
 	
`-L <lun_number>`
	LUN of the paths to use. Valid only if `--type` is `location`.

`--model <model>`
 	
`-M <model>`
	Model of the paths to use. Valid only if `--type` is `vendor`.
	
	Valid values are values of the Model string from the SCSI inquiry string.

`--plugin`
 	
`-P`
	PSA plug-in to use. Currently, the values are `NMP` or `MASK_PATH`, but third parties can ship their own PSA plug-ins in the future.

Table continued on next page

Continued from previous page

Options	Description
	`MASK_PATH` refers to the plug-in `MASK_PATH_PLUGIN`. The command adds claim rules for this plug-in if the user wants to mask the path. You can add a claim rule that causes the `MASK_PATH_PLUGIN` to claim the path to mask a path or LUN from the host. See the *vSphere Storage* documentation for details.

`--rule <rule_ID>`
 	
`-r <rule_ID>`
	Rule ID to use. Run **`esxcli storage core claimrule list`** to see the rule ID. The rule ID indicates the order in which the claim rule is to be evaluated. User-defined claim rules are evaluated in numeric order starting with 101.

`--target <target>`
 	
`-T <target>`
	Target of the paths to use. Valid only if `--type` is `location`.

`--transport <transport>`
 	
`-R <transport>`
	Transport of the paths to use. Valid only if `--type` is `transport`. The following values are supported.
	
	• `block` – block storage
	• `fc` – FibreChannel
	• `iscsivendor` — iSCSI
	• `iscsi` – not currently used
	• `ide` — IDE storage
	• `sas` — SAS storage
	• `sata` — SATA storage
	• `usb` – USB storage
	• `parallel` – parallel
	• `unknown`

`--type <type>`
 	
`-t <type>`
	Type of matching to use for the operation. Valid values are `vendor`, `location`, `driver`, and `transport`.

`--vendor`
 	
`-V`
	Vendor of the paths to use. Valid only if `--type` is `vendor`.
	
	Valid values are values of the vendor string from the SCSI inquiry string.

`--wwnn`	World-Wide Node Number for the target to use in this operation.
`--wwpn`	World-Wide Port Number for the target to use in this operation.

`--xcopy-max-transfer-size`
 	
`-m`
	Maximum data transfer size when using XCOPY. Valid only if `--xcopy-use-array-values` is specified.

`--xcopy-use-array-values`
 	
`-a`
	Use the array reported values to construct the XCOPY command to be sent to the storage array. This applies to VAAI claim rules only.

Table continued on next page

Continued from previous page

Options	Description

`--xcopy-use-multi-segs`
 	
`-s`
	Use multiple segments when issuing an XCOPY request. Valid only if `--xcopy-use-array-values` is specified.

Claim rules are numbered as follows.

- Rules 0–100 are reserved for internal use by VMware.
- Rules 101–65435 are available for general use. Any third-party multipathing plug-ins installed on your system use claim rules in this range. By default, the PSA claim rule 101 masks Dell array pseudo devices. Do not remove this rule, unless you want to unmask these devices.
- Rules 65436–65535 are reserved for internal use by VMware.

When claiming a path, the PSA runs through the rules starting from the lowest number and determines is a path matches the claim rule specification. If the PSA finds a match, it gives the path to the corresponding plug-in. This is worth noticing because a given path might match several claim rules.

The following examples illustrate adding claim rules. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

- Add rule 321, which claims the path on adapter vmhba0, channel 0, target 0, LUN 0 for the NMP.
  ```
  esxcli <conn_options> storage core claimrule add -r 321 -t location -A vmhba0 -C 0 -T
  0 -L 0 -P NMP
  ```
- Add rule 429, which claims all paths provided by an adapter with the mptscsi driver for the `MASK_PATH` plug-in.
  ```
  esxcli <conn_options> storage core claimrule add -r 429 -t driver -D mptscsi -P
  MASK_PATH
  ```
- Add rule 914, which claims all paths with vendor string `VMWARE` and model string `Virtual` for the NMP.
  ```
  esxcli <conn_options> storage core claimrule add -r 914 -t vendor -V VMWARE -M Virtual
  -P NMP
  ```
- Add rule 1015, which claims all paths provided by FC adapters for the NMP.
  ```
  esxcli <conn_options> storage core claimrule add -r 1015 -t transport -R fc -P NMP
  ```

Removing Claim Rules

The **`esxcli storage core claimrule remove`** command removes a claim rule from the set of claim rules on the system.

> **IMPORTANT**
> By default, the PSA claim rule 101 masks Dell array pseudo devices. Do not remove this rule, unless you want to unmask these devices.

Option	Description

`--rule <rule_ID>`
 	
`-r <rule_ID>`
	ID of the rule to be removed. Run **`esxcli storage core claimrule list`** to see the rule ID.

The following example removes rule 1015. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

```
esxcli <conn_options> storage core claimrule remove -r 1015
```

Listing Claim Rules

The **`list`** command lists all claim rules on the system.

You can specify the claim rule class as an argument.

Option	Description

`--claimrule-class <cl>`
 	
`-c <cl>`
	Claim rule class to use in this operation. You can specify `MP` (Multipathing), `Filter`, or `VAAI`. Multipathing is the default. Filter is used only for VAAI. Specify claim rules for both `VAAI_FILTER` and `VAAI` plug-in to use it. See *`vSphere Storage`* for information about VAAI.

You can run the command as follows. The equal sign is optional, so both forms of the command have the same result. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

```
esxcli <conn_options> storage core claimrule list -c Filter
```

```
esxcli <conn_options> storage core claimrule list --claimrule-class=Filter
```

Loading Claim Rules

The **`esxcli storage core claimrule load`** command loads claim rules from the `esx.conf` configuration file into the VMkernel. Developers and experienced storage administrators might use this command for boot time configuration.

`esxcli storage core claimrule load` has no options. The command always loads all claim rules from `esx.conf`.

Moving Claim Rules

The **`esxcli storage core claimrule move`** command moves a claim rule from one rule ID to another.

Options	Description

`--claimrule-class <cl>`
 	
`-c <cl>`
	Claim rule class to use in this operation.

`--new-rule <rule_ID>`
 	
`-n <rule_ID>`
	New rule ID you want to give to the rule specified by the `--rule` option.

`--rule <rule_ID>`
 	
`-r <rule_ID>`
	ID of the rule to be removed. Run **`esxcli storage core claimrule list`** to display the rule ID.

The following example renames rule 1016 to rule 1015 and removes rule 1016. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

```
esxcli <conn_options> storage core claimrule move -r 1015 -n 1016
```

Load and Apply Path Claim Rules

You can run the **`esxcli storage core claimrule run`** command to apply claim rules that are loaded.

If you do not call run, the system checks for claim rule updates every five minutes and applies them. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. Modify rules and load them.

   ```
   esxcli <conn_options> storage core claimrule load
   ```

2. Quiesce the devices that use paths for which you want to change the rule and unclaim those paths.

   ```
   esxcli <conn_options> storage core claiming unclaim --device=<device>
   ```

3. Run path claiming rules.

   ```
   esxcli <conn_options> storage core claimrule run
   ```

Running Path Claim Rules

The **`esxcli storage core claimrule run`** command runs path claiming rules.

You can run this command to apply claim rules that are loaded. See Load and Apply Path Claim Rules.

You can also use the **`esxcli storage core claimrule run`** command for troubleshooting and boot time configuration.

Options	Description

`--adapter <adapter>`
 	
`-A <adapter>`
	If `--type` is `location`, name of the HBA for the paths to run the claim rules on. To run claim rules on paths from all adapters, omit this option.

`--channel <channel>`
 	
`-C <channel>`
	If `--type` is `location`, value of the SCSI channel number for the paths to run the claim rules on. To run claim rules on paths with any channel number, omit this option.

`--claimrule-class`
 	
`-c`
	Claim rule class to use in this operation.

`--device`
 	
`-d`
	Device UID to use for this operation.

`--lun <lun>`
 	
`-L <lun>`
	If `--type` is `location`, value of the SCSI LUN for the paths to run claim rules on. To run claim rules on paths with any LUN, omit this option.

`--path <path_UID>`
 	
`-p <path_UID>`
	If `--type` is `path`, this option indicates the unique path identifier (UID) or the runtime name of a path to run claim rules on.

`--target <target>`
 	
`-T <target>`
	If `--type` is `location`, value of the SCSI target number for the paths to run claim rules on. To run claim rules on paths with any target number, omit this option.

`--type <location|path|all>`
 	
`-t <location|path|all>`
	Type of claim to perform. By default, uses `all`, which means claim rules run without restriction to specific paths or SCSI addresses. Valid values are `location`, `path`, and `all`.

`--wait`
 	
`-w`
	You can use this option only if you also use `--type all`.

Table continued on next page

Continued from previous page

Options	Description
	If the option is included, the claim waits for paths to settle before running the claim operation. In that case, the system does not start the claiming process until it is likely that all paths on the system have appeared before starting the claim process. After the claiming process has started, the command does not return until device registration has completed. If you add or remove paths during the claiming or the discovery process, this option might not work correctly.

Managing Users

An ESXi system grants access to its resources when a known user with appropriate permissions logs on to the system with a password that matches the one stored for that user.

You can use the vSphere SDK for all user management tasks. You cannot create ESXi users by using the vSphere Client.

Users in the vSphere Environment

Users and roles control who has access to vSphere components and what actions each user can perform.

User management is discussed in detail in the *vSphere Security* documentation.

vCenter Server and ESXi systems authenticate a user with a combination of user name, password, and permissions. Servers and hosts maintain lists of authorized users and the permissions assigned to each user.

Privileges define basic individual rights that are required to perform actions and retrieve information. ESXi and vCenter Server use sets of privileges, or roles, to control which users can access particular vSphere objects. ESXi and vCenter Server provide a set of pre-established roles.

The privileges and roles assigned on an ESXi host are separate from the privileges and roles assigned on a vCenter Server system. When you manage a host by using a vCenter Server system, only the privileges and roles assigned through the vCenter Server system are available. You cannot create ESXi users by using the vSphere Client.

Assigning Permissions with ESXCLI

You can use ESXCLI commands to manage permissions.

A set of ESXCLI commands allows you to perform the following operations.

- Give permissions to local users and groups by assigning them one of the predefined roles.
- Give permissions to Active Directory users and groups if your ESXi host has been joined to an Active Directory domain by assigning them one of the predefined roles.

  IMPORTANT
  When you manage local users on your ESXi host, you are not affecting the vCenter Server users.

Manage Permissions

You can list, remove, and set permissions for a user or group, as shown in the following example.

1. List permissions.
   ```
   esxcli system permission list
   ```
   The system displays permission information. The second column indicates whether the information is for a user or group.
   ```
   Principal            Is Group  Role

   ---------------------------------

   ABCDEFGH\esx^admins  true      Admin

   dcui                 false     Admin

   root                 false     Admin

   vpxuser              false     Admin

   test1                false     ReadOnly
   ```
2. Set permissions for a user or group. Specify the ID of the user or group, and set the `--group` option to `true` to indicate a group. Specify one of three roles, `Admin`, `ReadOnly` or `NoAccess`.
   ```
   esxcli system permission set --id test1 -r ReadOnly
   ```
3. Remove permissions for a user or group.
   ```
   esxcli system permission unset --id test1
   ```

Account Management

You can manage accounts by using the following commands.

```
esxcli system account add

esxcli system account set

esxcli system account list

esxcli system account remove
```

Managing Virtual Machines

You can manage virtual machines by using the vSphere Client. You can use ESXCLI to list all running virtual machines and to stop a specific virtual machine.

Forcibly Stop a Virtual Machine with ESXCLI

You can use ESXCLI to stop a virtual machine forcibly.

In some cases, virtual machines do not respond to the normal shutdown or stop commands. In these cases, it might be necessary to forcibly shut down the virtual machines. Forcibly shutting down a virtual machine might result in guest operating system data loss and is similar to pulling the power cable on a physical machine.
You can forcibly stop virtual machines that are not responding to normal stop operation with the **`esxcli vm process kill`** command. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. List all running virtual machines on the system to see the World ID of the virtual machine that you want to stop.
   ```
   esxcli <conn_options> vm process list
   ```
2. Stop the virtual machine by running the following command.

```
esxcli <conn_options> vm process kill --type <kill_type> --world-id <ID>
```

The command supports three
`--type`
options. Try the types sequentially - `soft` before `hard`, `hard` before `force`. The following types are supported through the
`--type`
option.

Type	Description
`soft`	Gives the VMX process a chance to shut down cleanly, like `kill` or `kill -SIGTERM`.
`hard`	Stops the VMX process immediately, like `kill -9` or `kill -SIGKILL`.
`force`	Stops the VMX process when other options do not work.

If all three options do not work, reboot your ESXi host to resolve the issue.

Managing vSphere Networking

The ESXCLI networking commands allow you to manage the vSphere network services.

You can connect virtual machines to the physical network and to each other and configure vSphere standard switches. Limited configuration of vSphere distributed switches is also supported. You can also set up your vSphere environment to work with external networks such as SNMP or NTP.

Introduction to vSphere Networking

At the core of vSphere Networking are virtual switches.

vSphere supports standard switches (VSS) and distributed switches (VDS). Each virtual switch has a preset number of ports and one or more port groups.

Virtual switches allow your virtual machines to connect to each other and to connect to the outside world.

- When two or more virtual machines are connected to the same virtual switch, and those virtual machines are also on the same port group or VLAN, network traffic between them is routed locally.
- When virtual machines are connected to a virtual switch that is connected to an uplink adapter, each virtual machine can access the external network through that uplink. The adapter can be an uplink connected to a standard switch or a distributed uplink port connected to a distributed switch.

Virtual switches allow your ESXi host to migrate virtual machines with VMware vMotion and to use IP storage through VMkernel network interfaces.

- Using vMotion, you can migrate running virtual machines with no downtime. You cannot enable vMotion with ESXCLI.
- IP storage refers to any form of storage that uses TCP/IP network communication as its foundation and includes iSCSI and NFS for ESXi. Because these storage types are network based, they can use the same VMkernel interface and port group.

The network services that the VMkernel provides (iSCSI, NFS, and vMotion) use a TCP/IP stack in the VMkernel. The VMkernel TCP/IP stack is also separate from the guest operating system's network stack. Each of these stacks accesses various networks by attaching to one or more port groups on one or more virtual switches.

Networking Using vSphere Standard Switches

vSphere standard switches allow you to connect virtual machines to the outside world.

Figure 186: Networking with vSphere Standard Switches

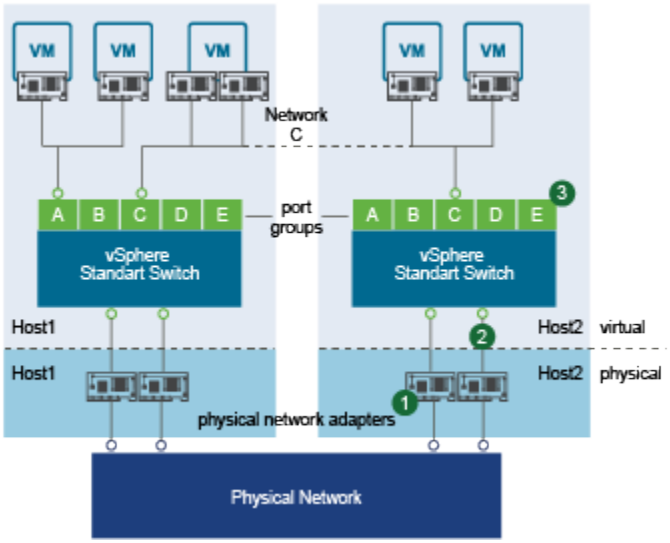


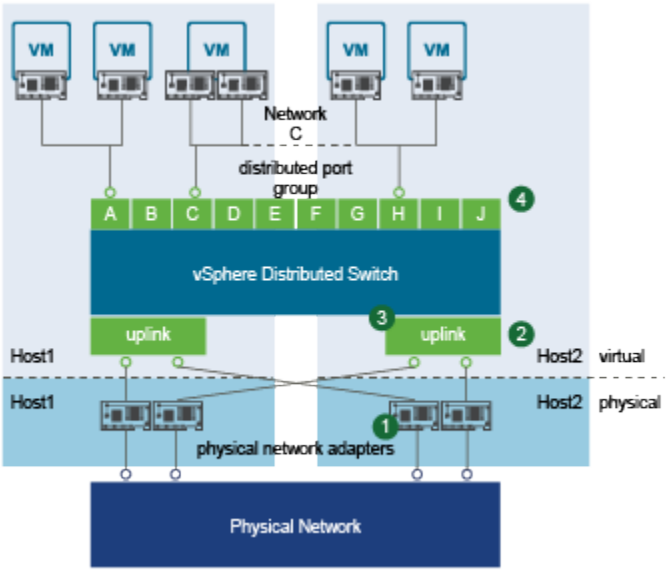
Figure: Networking with vSphere Standard Switches shows the relationship between the physical and virtual network elements. The numbers match those in the figure.

- Associated with each ESXi host are one or more uplink adapters (1). Uplink adapters represent the physical switches the ESXi host uses to connect to the network. You can manage uplink adapters by using the **`esxcli network nic`** command. See Managing Uplink Adapters.
- Each uplink adapter is connected to a standard switch (2). You can manage a standard switch and associate it with uplink adapters by using the **`esxcli network vswitch`** command. See Setting Up Virtual Switches and Associating a Switch with a Network Interface.
- Associated with the standard switch are port groups (3). Port group is a unique concept in the virtual environment. You can configure port groups to enforce policies that provide enhanced networking security, network segmentation, better performance, high availability, and traffic management. You can use the **`esxcli network vswitch standard portgroup`** command to associate a standard switch with a port group, and the **`esxcli network ip interface`** command to associate a port group with a VMkernel network interface.
- The VMkernel TCP/IP networking stack supports iSCSI, NFS, and vMotion and has an associated VMkernel network interface. You configure VMkernel network interfaces by using the **`esxcli network ip interface`** command. See Adding and Modifying VMkernel Network Interfaces. Separate VMkernel network interfaces are often used for separate tasks, for example, you might devote one VMkernel network interface card to vMotion only. Virtual machines run their own systems' TCP/IP stacks and connect to the VMkernel at the Ethernet level through virtual switches.

Networking Using vSphere Distributed Switches

When you want to connect a virtual machine to the outside world, you can use a standard switch or a distributed switch. With a distributed switch, the virtual machine can maintain its network settings even if the virtual machine is migrated to a different host.

Figure 187: Networking with vSphere Distributed Switches



- Each physical network adapter (1) on the host is paired with a distributed uplink port (2), which represents the uplink to the virtual machine. With distributed switches, the virtual machine no longer depends on the host's physical uplink but on the (virtual) uplink port. You manage a uplink ports primarily using the vSphere Client or vSphere APIs.
- The distributed switch itself (3) functions as a single virtual switch across all associated hosts. Because the switch is not associated with a single host, virtual machines can maintain consistent network configuration as they migrate from one host to another.

  Like a standard switch, each distributed switch is a network hub that virtual machines can use. A distributed switch can route traffic internally between virtual machines or link to an external network by connecting to physical network adapters. You create a distributed switch by using the vSphere Client UI. You can list distributed virtual switches by using the **`esxcli network vswitch`** command. See Setting Up Virtual Switches and Associating a Switch with a Network Interface.

Retrieving Basic Networking Information

Service console commands for retrieving networking information are not included in the ESXi Shell. You can instead use ESXCLI commands directly in the shell.

For information corresponding to the Linux **`ifconfig`** command, use the following ESXCLI commands.

```
esxcli <conn_options> network ip interface list
esxcli <conn_options> network ip interface ipv4 get -n vmk<X>
esxcli <conn_options> network ip interface ipv6 get -n vmk<X>
esxcli <conn_options> network ip interface ipv6 address list
```

For information corresponding to the Linux **`netstat`** command, use the following ESXCLI command.

```
esxcli <conn_options> network ip connection list
```

You can also ping individual hosts with the **`esxcli network diag ping`** command. The command includes options for using ICMPv4 or ICMPv6 packet requests, specifying an interface to use, specifying the interval, and so on.

Troubleshoot a Networking Setup

You can use ESXCLI network commands to view network statistics and troubleshoot your networking setup. The nested hierarchy of commands allows you to drill down to potential trouble spots.

1. List all virtual machine networks on a host.

   ```
   esxcli network vm list
   ```

   The command returns for each virtual machine the World ID, name, number of ports, and networks, as in the following example.

   ```
   World ID  Name                    Num Ports  Networks

   ------------------------------------------------

      10374  ubuntu-server-11.04-1          2  VM Network, dvportgroup-19

      10375  ubuntu-server-11.04-2          2  VM Network, dvportgroup-19

      10376  ubuntu-server-11.04-3          2  VM Network, dvportgroup-19

      10408  ubuntu-server-11.04-4          3  VM Network, VM Network 10Gbps,
   dvportgroup-19
   ```

2. List the ports for one of the virtual machines by specifying its World ID.

   ```
   esxcli network vm port list -w 10408
   ```

   The command returns port information, as in the following example.

   ```
   Port:

     Port ID: XXXXXXXX

     vSwitch: vSwitch0

     Portgroup: VM Network

     DVPort ID:

     MAC Address: 00:XX:XX:aa:XX:XX

     IP Address: 10.XXX.XXX.XXX

     Team Uplink: vmnic0

     Uplink Port ID: 12345678

     Active Filters:
   ```

3. Retrieve the switch statistics for a port.

   ```
   esxcli network port stats get -p 12345678
   ```

   The command returns detailed statistics, as in the following example.

   ```
   Packet statistics for port 12345678:

      Packets received: 517631

      Packets sent: 18937

      Bytes received: 100471874
   ```

```
   Bytes sent: 1527233
   Broadcast packets received: 474160
   Broadcast packets sent: 107
   Multicast packets received: 8020
   Multicast packets sent: 8
   Unicast packets received: 35451
   Unicast packets sent: 18822
   Receive packets dropped: 45
   Transmit packets dropped: 0
```

4. Retrieve the filter information for the port.

   ```
   esxcli network port filter stats get -p 12345678
   ```

   The command returns detailed statistics, as in the following example.

   ```
   Filter statistics for dvfilter-test:
      Filter direction: Receive
      Packets in: 202080
      Packets out: 202080
      Packets dropped: 0
      Packets filtered: 0
      Packets faulted: 0
      Packets queued: 0
      Packets injected: 0
      Packet errors: 0
   ```

5. Retrieve complete statistics for a NIC.

   ```
   esxcli network nic stats get -n vmnic0
   ```

6. Get a per-VLAN packed breakdown on a NIC.

   ```
   esxcli network nic vlan stats get -n vmnic0
   ```

   The command returns the number of packets sent and received for the VLAN you specified.

Setting Up vSphere Networking with vSphere Standard Switches

You can use ESXCLI to set up the vSphere networking.

You can set up your virtual network by performing a set of tasks.

1. Create or manipulate virtual switches by using **`esxcli network vswitch`**. By default, eachESXi host has one virtual switch, `vSwitch0`. You can create additional virtual switches or manage existing switches. See Setting Up Virtual Switches and Associating a Switch with a Network Interface.
2. Make changes to the uplink adapter by using **`esxcli network vswitch standard uplink`**. See Managing Uplink Adapters.
3. Use **`esxcli network ip interface`** to configure the VMkernel network interfaces. See Adding and Modifying VMkernel Network Interfaces.

Setting Up Virtual Switches and Associating a Switch with a Network Interface

A virtual switch models a physical Ethernet switch. You can manage virtual switches and port groups by using the vSphere Client or ESXCLI commands.

By default, each ESXi host has a single virtual switch called `vSwitch0`. Each virtual switch has logical ports. For information about maximum allowed virtual switches and ports, see the *`VMware Configuration Maximums`* tool. Ports connect to the virtual machines and the ESXi physical network adapters.

- You can connect one virtual machine network adapter to each port by using the vSphere Client UI.
- You can connect the uplink adapter to the virtual switches by using **`esxcli network vswitch standard uplink`**. See Linking and Unlinking Uplink Adapters with ESXCLI.

When two or more virtual machines are connected to the same virtual switch, network traffic between them is routed locally. If an uplink adapter is attached to the virtual switch, each virtual machine can access the external network that the adapter is connected to.

This section discusses working in a standard switch environment. See Networking Using vSphere Distributed Switches for information about distributed switch environments.

When working with virtual switches and port groups, perform the following tasks.

1. Find out which virtual switches are available and, optionally, what the associated MTU and CDP (Cisco Discovery Protocol) settings are. See Retrieving Information About Virtual Switches with ESXCLI.
2. Add a virtual switch. See Adding and Deleting Virtual Switches with ESXCLI.
3. For a newly added switch, perform these tasks.

    a. Add a port group. See Managing Port Groups with ESXCLI.
    b. Set the port group VLAN ID. See Setting the Port Group VLAN ID with ESXCLI.
    c. Add an uplink adapter. See Linking and Unlinking Uplink Adapters with ESXCLI.
    d. Change the MTU or CDP settings. See Setting Switch Attributes with ESXCLI.

Retrieving Information About Virtual Switches with ESXCLI

You can retrieve information about virtual switches by using **`esxcli network vswitch`** commands.

Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

- List all virtual switches and associated port groups.
  ```
  esxcli <conn_options> network vswitch standard list
  ```
  The command prints information about the virtual switch, which might include its name, number of ports, MTU, port groups, and other information. The output includes information about CDP settings for the virtual switch. The precise information depends on the target system. The default port groups are `Management Network and VM Network`.
- List the network policy settings, such as security policy, traffic shaping policy, and failover policy, for the virtual switch. The following commands are supported.

```
esxcli <conn_options> network vswitch standard policy failover get
esxcli <conn_options> network vswitch standard policy security get
esxcli <conn_options> network vswitch standard policy shaping get
```

Adding and Deleting Virtual Switches

You can add and delete virtual switches with ESXCLI.

Adding and Deleting Virtual Switches with ESXCLI

You can add and delete virtual switches by using the **`esxcli network vswitch standard`** namespace.

Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

- Add a virtual switch.
  ```
  esxcli <conn_options> network vswitch standard add --vswitch-name=vSwitch42
  ```
  You can specify the number of port groups while adding the virtual switch. If you do not specify a value, the default value is used. The system-wide port count cannot be greater than 4096.
  ```
  esxcli <conn_options> network vswitch standard add --vswitch-name=vSwitch42 --ports=8
  ```
  After you have added a virtual switch, you can set switch attributes. See Setting Switch Attributes with ESXCLI. You can also add one or more uplink adapters. See Linking and Unlinking Uplink Adapters with ESXCLI.
- Delete a virtual switch.
  ```
  esxcli <conn_options> network vswitch standard remove --vswitch-name=vSwitch42
  ```
  You cannot delete a virtual switch if any ports on the switch are still in use by VMkernel networks or virtual machines. Run **`esxcli network vswitch standard list`** to determine whether a virtual switch is in use.

Setting Switch Attributes with ESXCLI

You can set the maximum transmission unit (MTU) and CDP status for a virtual switch. The CDP status shows which Cisco switch port is connected to which uplink.

Specify one of the options listed in *`Connection Options for vCLI Host Management Commands`* in place of `<conn_options>`.

- Set the MTU for a vSwitch.
  ```
  esxcli <conn_options> network vswitch standard set --mtu=9000 --vswitch-name=vSwitch1
  ```
  The MTU is the size, in bytes, of the largest protocol data unit the switch can process. When you set this option, it affects all uplinks assigned to the virtual switch.
- Set the CDP value for a vSwitch. You can set status to `down`, `listen`, `advertise`, or `both`.
  ```
  esxcli <conn_options> network vswitch standard set --cdp-status=listen --vswitch-
  name=vSwitch1
  ```

Managing Port Groups with ESXCLI

You can use **`esxcli network vswitch standard portgroup`** to check, add, and remove port groups.

Network services connect to vSwitches through port groups. A port group allows you to group traffic and specify configuration options such as bandwidth limitations and VLAN tagging policies for each port in the port group. A virtual switch must have one port group assigned to it. You can assign additional port groups.

Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

- List port groups currently associated with a virtual switch.
  ```
  esxcli <conn_options> network vswitch standard portgroup list
  ```
  The command lists the port group name, associated virtual switch, active clients, and VLAN ID.
- Add a port group.
  ```
  esxcli <conn_options> network vswitch standard portgroup add --portgroup-name=<name>
  --vswitch-name=vSwitch1
  ```
- Delete one of the existing port groups.
  ```
  esxcli <conn_options> network vswitch standard portgroup remove --portgroup-
  name=<name> --vswitch-name=vSwitch1
  ```

Connecting and Disconnecting Uplink Adapters and Port Groups with ESXCLI

You can use **`esxcli network vswitch standard portgroup policy failover set`** to connect and disconnect uplink adapters and port groups.

If your setup includes one or more port groups, you can associate each port group with one or more uplink adapters and remove the association. This functionality allows you to filter traffic from a port group to a specific uplink, even if the virtual switch is connected with multiple uplinks.

Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

- Connect a port group with an uplink adapter.
  ```
  esxcli <conn_options> network vswitch standard portgroup policy failover set --active-
  uplinks=vmnic1,vmnic6,vmnic7
  ```
  This command fails silently if the uplink adapter does not exist.
- Make some of the adapters standby instead of active.
  ```
  esxcli <conn_options> network vswitch standard portgroup policy failover set --
  standby-uplinks=vmnic1,vmnic6,vmnic7
  ```

Setting the Port Group VLAN ID with ESXCLI

You can use **`esxcli network vswitch standard portgroup set`** to manage VLANs.

VLANs allow you to further segment a single physical LAN segment so that groups of ports are isolated as if they were on physically different segments. The standard is IEEE 802.1Q.

A VLAN ID restricts port group traffic to a logical Ethernet segment within the physical network.

- Set the VLAN ID to 4095 to allow a port group to reach port groups located on other VLAN.
- Set the VLAN ID to 0 to disable the VLAN for this port group.

If you use VLAN IDs, you must change the port group labels and VLAN IDs together so that the labels properly represent connectivity. VLAN IDs are optional.

You can use the following commands for VLAN management.

- Allow port groups to reach port groups located on other VLANs.
  ```
  esxcli <conn_options> network vswitch standard portgroup set -p <pg_name> --vlan-id
  4095
  ```
  Run the command multiple times to allow all ports to reach port groups located on other VLANs.
- Disable VLAN for port group g42.

```
esxcli <conn_options> network vswitch standard portgroup set --vlan-id 0 -p g42
```

Run **`esxcli network vswitch standard portgroup list`** to list all port groups and associated VLAN IDs.

Managing Uplink Adapters

You can manage uplink adapters, which represent the physical NICs that connect the ESXi host to the network by using the **`esxcli network nic`** command. You can also use **`esxcli network vswitch`** to link and unlink the uplink.

You can use **`esxcli network nic`** to list all uplinks, to list information, to set attributes, and to bring a specified uplink down or up.

Manage Uplink Adapters with ESXCLI

You can use ESXCLI to manage uplink adapters.

The following example workflow lists all uplink adapters, lists properties for one uplink adapter, changes the uplink's speed and duplex settings, and brings the uplink down and back up. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. List all uplinks and information about each device.

   ```
   esxcli <conn_options> network nic list
   ```

   You can narrow down the information displayed by using `esxcli network nic get --nic-name=<nic>`.

2. Bring down one of the uplink adapters.

   ```
   esxcli <conn_options> network nic down --nic-name=vmnic0
   ```

3. Change uplink adapter settings.

   ```
   esxcli <conn_options> network nic set <option>
   ```

   You must specify one of the following options.

Option	Description
`-a|--auto`	Sets the speed and duplex settings to autonegotiate.
`-D|--duplex=<str>`	Duplex to set this NIC to. Acceptable values are `full` and `half`.
`-P | --phy-address`	Sets the MAC address of the device
`-l|--message-level=<long>`	Sets the driver message level. Message levels and what they imply differ per driver.
`-n|--nic-name=<str>`	Name of the NIC to configured. Must be one of the cards listed in the **`nic list`** command (required).
`-p|--port=<str>`	Selects the device port. The following device ports are available. • `aui` • `bnc` • `fibre` • `mii` • `tp`
`-S|--speed=<long>`	Speed to set this NIC to. Acceptable values are 10, 100, 1000, and 10000.

Table continued on next page

Continued from previous page

Option	Description
`-t|--transceiver-type=<str>`	Selects transceiver type. The following transceiver types are available. • `external` • `internal`
`-w|--wake-on-lan=<str>`	Sets Wake-on-LAN options. Not all devices support this option. The option value is a string of characters specifying which options to enable. • `p` – Wake on phy activity • `u` – Wake on unicast messages • `m` – Wake on multicast messages • `b` – Wake on broadcast messages • `a` – Wake on ARP • `g` – Wake on MagicPacket • `s` – Enable SecureOn password for MagicPacket

4. Bring the uplink adapter back up.

   ```
   esxcli <conn_options> network nic up --nic-name=vmnic0
   ```

Specifying Multiple Uplinks with ESXCLI

At any time, one port group NIC array and a corresponding set of active uplinks exist. When you change the active uplinks, you also change the standby uplinks and the number of active uplinks.

The following example illustrates how active and standby uplinks are set.

1. The port group NIC array is `[vmnic1, vmnic0, vmnic3, vmnic5, vmnic6, vmnic7]` and `active-uplinks` is set to three uplinks - `vmnic1`, `vmnic0`, `vmnic3`. The other uplinks are standby uplinks.
2. You set the active uplinks to a new set `[vmnic3, vmnic5]`.
3. The new uplinks override the old set. The NIC array changes to `[vmnic3, vmnic5, vmnic6, vmnic7]`. `vmnic0` and `vmnic1` are removed from the NIC array and `max-active` becomes 2.

If you want to keep `vmnic0` and `vmnic1` in the array, you can make those NICs standby uplinks in the command that changes the active uplinks.

```
esxcli network vswitch standard portgroup policy failover set -p testPortgroup --active-uplinks vmnic3,vmnic5 --standby-uplinks vmnic1,vmnic0,vmnic6,vmnic7
```

Linking and Unlinking Uplink Adapters with ESXCLI

You can use ESXCLI to link and unlink uplink adapters.

When you create a virtual switch by using **`esxcli network vswitch standard add`**, all traffic on that virtual switch is initially confined to that virtual switch. All virtual machines connected to the virtual switch can talk to each other, but the virtual machines cannot connect to the network or to virtual machines on other hosts. A virtual machine also cannot connect to virtual machines connected to a different virtual switch on the same host.

Having a virtual switch that is not connected to the network might make sense if you want a group of virtual machines to be able to communicate with each other, but not with other hosts or with virtual machines on other hosts. In most cases, you set up the virtual switch to transfer data to external networks by attaching one or more uplink adapters to the virtual switch.

You can use the following commands to list, add, and remove uplink adapters. When you link by using ESXCLI, the physical NIC is added as a standby adapter by default. You can then modify the teaming policy to make the physical NIC active by running the command **`esxcli network vswitch standard policy failover set`**.

- List uplink adapters.
  `esxcli <conn_options> network vswitch standard list`

  The uplink adapters are returned in the `Uplink` item.
- Add a new uplink adapter to a virtual switch.
  `esxcli <conn_options> network vswitch standard uplink add --uplink-name=vmnic15 --vswitch-name=vSwitch0`
- Remove an uplink adapter from a virtual switch.
  `esxcli <conn_options> network vswitch standard uplink remove --uplink-name=vmnic15 --vswitch-name=vSwitch0`

Adding and Modifying VMkernel Network Interfaces

VMkernel network interfaces are used primarily for management traffic, which can include vMotion, IP Storage, and other management traffic on the ESXi system. You can also bind a newly created VMkernel network interface for use by software and dependent hardware iSCSI by using the **`esxcli iscsi`** commands.

The VMkernel network interface is separate from the virtual machine network. The guest operating system and application programs communicate with a VMkernel network interface through a commonly available device driver or a VMware device driver optimized for the virtual environment. In either case, communication in the guest operating system occurs as it would with a physical device. Virtual machines can also communicate with a VMkernel network interface if both use the same virtual switch.

Each VMkernel network interface has its own MAC address and one or more IP addresses, and responds to the standard Ethernet protocol as would a physical NIC. The VMkernel network interface is created with TCP Segmentation Offload (TSO) enabled.

You can manage VMkernel NICs with ESXCLI.

Managing VMkernel Network Interfaces with ESXCLI

You can configure the VMkernel network interface for IPv4 or for IPv6 with ESXCLI.

For IPv4, see Add and Configure an IPv4 VMkernel Network Interface with ESXCLI. For IPv6, see Add and Configure an IPv6 VMkernel Network Interface with ESXCLI.

Add and Configure an IPv4 VMkernel Network Interface with ESXCLI

You can add and configure an IPv4 VMkernel NIC by using ESXCLI.

Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. Add a new VMkernel network interface.

   `esxcli <conn_options> network ip interface add --interface-name=vmk<X> --portgroup-name=<my_portgroup>`

You can specify the MTU setting after you have added the network interface by using `esxcli network ip interface set --mtu`.

2. Configure the interface as an IPv4 interface.

   You must specify the IP address by using `--ip`, the netmask, and the name. For the following examples, assume that VMSF-VMK-363 is a port group to which you want to add a VMkernel network interface.

   ```
   esxcli <conn_options> network ip interface ipv4 set --ipv4=<ip_address> --
   netmask=255.255.255.0 --type=<value> --interface-name=vmk<X>
   ```

   You can set the address as follows.

   - `--ipv4=<X.X.X.X> --netmask=<X.X.X.X> --type=static` – Static IPv4 address.
   - `--type=dhcp` – Use IPv4 DHCP.

   The VMkernel supports DHCP only for ESXi 4.0 and later.

   When the command finishes successfully, the newly added VMkernel network interface is enabled.

3. List information about all VMkernel network interfaces on the system.

   ```
   esxcli <conn_options> network ip interface list
   ```

   The command displays the network information, port group, MTU, and current state for each virtual network adapter in the system.

Add and Configure an IPv6 VMkernel Network Interface with ESXCLI

You can add and configure an IPv6 VMkernel NIC by using ESXCLI.

Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. Add a new VMkernel network interface.

   ```
   esxcli <conn_options> network ip interface add --interface-name=vmk<x> --portgroup-
   name=<my_portgroup>
   ```

   You can specify the MTU setting after you have added the network interface by using `esxcli network ip interface set --mtu`.

   When the command finishes successfully, the newly added VMkernel network interface is enabled.

2. Run **`esxcli network ip interface ipv6 address add`** to configure the interface as an IPv6 interface.

   You must specify the IP address using `--ip` and the name. For the following examples, assume that VMSF-VMK-363 is a port group to which you want to add a VMkernel network interface.

   ```
   esxcli <conn_options> network ip interface ipv6 address add --ip=<X:X:X::/X> --
   interface-name=vmk<X>
   ```

   You can set the address as follows.

   - <X:X:X::/X> - Static IPv6 address.
   - `--enable-dhcpv6` - Enables DHCPv6 on this interface and attempts to acquire an IPv6 address from the network.
   - `--enable-router-adv` - Use the IPv6 address advertised by the router. The address is added when the router sends the next router advert.

The VMkernel supports DHCP only for ESXi 4.0 and later.

When the command finishes successfully, the newly added VMkernel network interface is enabled.

3. List information about all VMkernel network interfaces on the system.

```
esxcli <conn_options> network ip interface list
```

The command displays the network information, port group, MTU, and current state for each virtual network adapter in the system.

4. Remove the IPv6 address and disable IPv6.

```
esxcli <conn_options> network ip interface ipv6 address remove --interface-
name=<VMK_NIC> --ipv6=<ipv6_addr>
```

```
esxcli <conn_options> network ip set --ipv6-enabled=false
```

Managing Standard Networking Services in the vSphere Environment

You can use ESXCLI commands to set up DNS, NTP, SNMP, and the default gateway for your vSphere environment.

Setting the DNS Configuration with ESXCLI

The **`esxcli network ip dns`** command lists and specifies the DNS configuration of your ESXi host.

> **IMPORTANT**
> If you try to change the host or domain name or the DNS server on hosts that use DHCP, an error results.

In network environments where a DHCP server and a DNS server are available, ESXi hosts are automatically assigned DNS names.

In network environments where automatic DNS is not available or you do not want to use automatic DNS, you can configure static DNS information, including a host name, primary name server, secondary name server, and DNS suffixes.

The **`esxcli network ip dns`** namespace includes two namespaces.

- **`esxcli network ip dns search`** includes commands for DNS search domain configuration.
- **`esxcli network ip dns server`** includes commands for DNS server configuration.

Set Up a DNS Server with ESXCLI

You can use ESXCLI to set up a DNS server.

The following example illustrates setting up a DNS server. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. Print a list of DNS servers configured on the system in the order in which they will be used.

```
esxcli <conn_options> network ip dns server list
```

If DNS is not set up for the target server, the command returns an empty string.

2. Add a server by running **`esxcli network ip dns server add`** and specifying the server IPv4 or IPv6 address.

```
esxcli <conn_options> network ip dns server add --server=<str>
```

3. Change the DNS settings.

- Specify the DNS server by using the
  `--dns`
  option and the DNS host.
  ```
  esxcli <conn_options> network ip dns server add --server=<server>
  ```
  Run the command multiple times to specify multiple DNS hosts.
- Configure the DNS host name for the server specified by
  `--server`
  or
  `--vihost`
  .
  ```
  esxcli <conn_options> system hostname set --host=<new_host_name>
  ```
- Configure the DNS domain name for the server specified by
  `--server`
  or
  `--vihost`
  .
  ```
  esxcli <conn_options> system hostname --domain=mydomain.biz
  ```

4. To turn on DHCP, enable DHCP and set the VMkernel NIC.
   - Turn on DHCP for IPv4.
     ```
     esxcli <conn_options> network ip interface ipv4 set --type dhcp/none/static
     ```
     ```
     esxcli <conn_options> network ip interface ipv4 set --peer-dns=<str>
     ```
   - Turn on DHCP for IPv6.
     ```
     esxcli <conn_options> network ip interface ipv6 set --enable-dhcpv6=true/false
     ```
     ```
     esxcli <conn_options> network ip interface ipv6 set --peer-dns=<str>
     ```

Modify DNS Setup for a Preconfigured Server with ESXCLI

You can use ESXCLI to modify the setup of a preconfigured DNS server.

Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. Display DNS properties for the specified server.
   a) List the host and domain name.
   ```
   esxcli <conn_options> system hostname get
   ```
   b) List available DNS servers.
   ```
   esxcli <conn_options> network ip dns server list
   ```
   c) List the DHCP settings for individual VMkernel NICs.
   ```
   esxcli <conn_options> network ip interface ipv4 get
   ```
   ```
   esxcli <conn_options> network ip interface ipv6 get
   ```

2. If the DNS properties are set, and you want to change the DHCP settings, you must specify the virtual network adapter to use when overriding the system DNS.

   You can override the existing DHCP setting by using the following commands.

```
esxcli <conn_options> network ip interface ipv4 set --type dhcp/none/static
esxcli <conn_options> network ip interface ipv6 set --enable-dhcpv6=true/false
```

Setting Up IPsec

You can set Internet Protocol Security by using **`esxcli network ip ipsec`**, which secures IP communications coming from and arriving at ESXi hosts. Administrators who perform IPsec setup must have a solid understanding of both IPv6 and IPsec.

ESXi hosts support IPsec only for IPv6 traffic, but not for IPv4 traffic.

You can run **`esxcli network ip ipsec`** commands with a vCenter Server system as a target, by using the `--vihost` option.

The VMware implementation of IPsec adheres to the following IPv6 RFCs.

- 4301 Security Architecture for the Internet Protocol
- 4303 IP Encapsulating Security Payload (ESP)
- 4835 Cryptographic Algorithm Implementation Requirements for ESP
- 2410 The NULL Encryption Algorithm and Its Use With IPsec
- 2451 The ESP CBC-Mode Cipher Algorithms
- 3602 The AES-CBC Cipher Algorithm and Its Use with IPsec
- 2404 The Use of HMAC-SHA-1-96 within ESP and AH
- 4868 Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512

Using IPsec with ESXi

When you set up IPsec on an ESXi host, you enable protection of incoming or outgoing data. What happens precisely depends on how you set up the system's Security Associations (SAs) and Security Policies (SPs).

- An SA determines how the system protects traffic. When you create an SA, you specify the source and destination, authentication, and encryption parameters, and an identifier for the SA with the following options.

esxcli network ip ipsec
`--sa-source` and `--sa-destination`
`--sa-spi`
`--sa-mode`
`--encryption-algorithm` and `--encryption-key`
`--integrity-algorithm` and `--integrity-key`

- An SP identifies and selects traffic that must be protected. An SP consists of two logical sections, a selector, and an action.
  The selector is specified by the following options.

esxcli network ip ipsec
--sa-source and --source-port
--destination-port
--upper-layer-protocol
--flow-direction

The action is specified by the following options.

esxcli network ip ipsec
--sa-name
--sp-name
--action

Because IPsec allows you to target precisely which traffic should be encrypted, it is well suited for securing your vSphere environment. For example, you can set up the environment so all vMotion traffic is encrypted.

Managing Security Associations

You can specify an SA and request that the VMkernel use that SA.

The following options for SA setup are supported.

esxcli Option	Description
sa-source <source_IP>	Source IP for the SA.
sa-destination <destination_IP>	Destination IP for the SA.
sa-spi	Security Parameter Index (SPI) for the SA. Must be a hexadecimal number with a 0x prefix. When IPsec is in use, ESXi uses the ESP protocol (RFC 43030), which includes authentication and encryption information and the SPI. The SPI identifies the SA to use at the receiving host. Each SA you create must have a unique combination of source, destination, protocol, and SPI.
sa-mode [tunnel | transport]	Either tunnel or transport. In tunnel mode, the original packet is encapsulated in another IPv6 packet, where source and destination addresses are the SA endpoint addresses.
encryption-algorithm [null | 3des-cbc | aes128-cbc]	Encryption algorithm to be used. Choose 3des-cbc or aes128-cbc, or null for no encryption.
encryption-key <key>	Encryption key to be used by the encryption algorithm. A series of hexadecimal digits with a 0x prefix or an ASCII string.
integrity-algorithm [hmac-sha1 | hmac-sha2-256]	Authentication algorithm to be used. Choose hmac-sha1 or hmac-sha2-256.
integrity-key	Authentication key to be used. A series of hexadecimal digits or an ASCII string.

You can perform these main tasks with SAs.

- Create an SA. You specify the source, the destination, and the authentication mode. You also specify the authentication algorithm and authentication key to use. You must specify an encryption algorithm and key, but you can specify `null` if you want no encryption. Authentication is required and cannot be `null`. The following example includes extra line breaks for readability. The last option, `sa_2` in the example, is the name of the SA.

```
esxcli network ip ipsec sa add

            --sa-source 2001:DB8:1::121

            --sa-destination 2001:DB8:1::122

            --sa-mode transport

            --sa-spi 0x1000

            --encryption-algorithm 3des-cbc

            --encryption-key 0x697076367265616479 6c6f676f33646573636362636f757432

            --integrity-algorithm hmac-sha1

            --integrity-key 0x69707636726561647 96c6f67736861316f757432

            --sa-name sa_2
```

- List an SA by using **`esxcli network ip ipsec sa list`**. This command returns SAs currently available for use by an SP. The list includes SAs you created.
- Remove a single SA by using **`esxcli network ip ipsec sa remove`**. If the SA is in use when you run this command, the command cannot perform the removal.
- Remove all SAs by using **`esxcli network ip ipsec sa remove --removeall`**. This option removes all SAs even when they are in use.

  CAUTION
  Running **`esxcli network ip ipsec sa remove --removeall`** removes all SAs on your system and might leave your system in an inconsistent state.

Managing Security Policies

After you have created one or more SAs, you can add security policies (SPs) to your ESXi hosts. While the SA specifies the authentication and encryption parameters to use, the SP identifies and selects traffic.

The following options for SP management are supported.

esxcli Option	Description
`sp-source <ip>/<p_len>`	Source IP address and prefix length.
`sp-destination <ip>/<p_len>`	Destination IP address and prefix length.
`source-port <port>`	Source port (0-65535). Specify `any` for any ports.
`destination-port <port>`	Destination port (0-65535). Specify `any` for any ports. If `ulproto` is `icmp6`, this number refers to the icmp6 type. Otherwise, this number refers to the port.

Table continued on next page

Continued from previous page

esxcli Option	Description
`upper-layer-protocol [any | tcp | udp | icmp6]`	Upper layer protocol. Use this option to restrict the SP to only certain protocols, or use any to apply the SP to all protocols.
`flow-direction [in | out]`	Direction in which you want to monitor the traffic. To monitor traffic in both directions, create two policies.
`action [none | discard | ipsec]`	Action to take when traffic with the specified parameters is encountered. • `none` - Take no action, that is, allow traffic unmodified. • `discard` - Do not allow data in or out. • `ipsec` - Use the authentication and encryption information specified in the SA to determine whether the data come from a trusted source.
`sp-mode [tunnel | transport]`	Mode, either `tunnel` or `transport`.
`sa-name`	Name of the SA to use by this SP.

You can perform the following main tasks with SPs.

- Create an SP by using **`esxcli network ip ipsec add`**. You identify the data to monitor by specifying the selector's source and destination IP address and prefix, source port and destination port, upper layer protocol, direction of traffic, action to take, and SP mode. The last two option are the name of the SA to use and the name of the SP that is being created. The following example includes extra line breaks for readability.

```
esxcli network ip ipsec add

   --sp-source=2001:0DB8:0001:/48

   --sp-destination=2001:0DB8:0002:/48

   --source-port=23

   --destination-port=25

   --upper-layer-protocol=tcp

   --flow-direction=out

   --action=ipsec

   --sp-mode=transport

   --sp-name sp_2
```

- List an SP by using **`esxcli network ip ipsec list`**. This command returns SPs currently available. All SPs are created by the administrator.
- Remove an SP by using **`esxcli network ip ipsec remove`**. If the SP is in use when you run this command, the command cannot perform the removal. You can run **`esxcli network ip ipsec remove --removeall`** instead to remove the SP even when it is in use.

  CAUTION

  Running **`esxcli network ip ipsec remove --removeall`** removes all SPs on your system and might leave your system in an inconsistent state.

Manage the ESXi Firewall

To minimize the risk of an attack through the management interface, ESXi includes a firewall between the management interface and the network.

To ensure the integrity of the host, only a small number of firewall ports are open by default. The *vSphere Security* documentation explains how to set up firewalls for your environment and which ports you might have to temporarily enable for certain traffic.
You manage firewalls by setting up firewall rulesets. The *vSphere Security* documentation explains how to perform these tasks by using the vSphere Client. You can also **`use esxcli network firewall`** to manage firewall rulesets and to retrieve information about them. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. Check firewall status and sshServer ruleset status.
   ```
   esxcli <conn_options> network firewall get


   Default Action: DROP

   Enabled: true

   Loaded: true


   esxcli <conn_options> network firewall ruleset list --ruleset-id sshServer


   Name       Enabled

   ---------  -------

   sshServer     true
   ```
2. Enable the sshServer ruleset if it is disabled.
   ```
   esxcli <conn_options> network firewall ruleset set --ruleset-id sshServer --enabled
   true
   ```
3. Obtain access to the ESXi Shell and check the status of the `allowedAll` flag.
   ```
   esxcli <conn_options> network firewall ruleset allowedip list --ruleset-id sshServer


   Ruleset    Allowed IP Addresses

   ---------  --------------------

   sshServer  All
   ```
   See *Getting Started with ESXCLI* for information on accessing the ESXi Shell.
4. Set the status of the `allowedAll` flag to `false`.
   ```
   esxcli <conn_options> network firewall ruleset set --ruleset-id sshServer --allowed-
   all false
   ```
5. Add the list of allowed IP addresses.

```
esxcli <conn_options> network firewall ruleset allowedip add --ruleset-id sshServer
--ip-address 192.XXX.1.0/24

esxcli <conn_options> network firewall ruleset allowedip add --ruleset-id sshServer
--ip-address 192.XXX.10.10
```

6. Check the allowed IP address list.

```
esxcli <conn_options> network firewall ruleset allowedip list --ruleset-id sshServer


Ruleset    Allowed IP Addresses

---------  ----------------------------

sshServer  192.XXX.10.10, 192.XXX.1.0/24
```

Monitor VXLAN

The **`esxcli network vswithch dvs vmware vxlan`** namespace supports commands for exploring VXLAN configuration details.

For a more detailed example of this functionality, see the VMware vSphere blog post about the topic.

1. List all available VXLAN vNetwork Distributed Switches.

```
esxcli network vswitch dvs vmware vxlan list
```

2. View the VXLAN statistics level.

```
esxcli network vswitch dvs vmware vxlan config stats get
```

3. Change the statistics level, for example, from 0 to 1.

```
esxcli network vswitch dvs vmware vxlan config stats set --level 1
```

   You can decide to filter statistics as follows.

   – For a vNetwork Distributed Switch, localized to an ESXi host
   – For a VTEP VMkernel interface
   – For a VXLAN segment ID
   – For a vNetwork Distributed Switch port ID

4. View statistics for a specific vNetwork Distributed Switch.

```
esxcli network vswitch dvs vmware vxlan config stats list --vds-name Cluster01-VXLAN-
VDS
```

5. View statistics for a VXLAN segment ID.
   – List the available segment IDs.

```
esxcli network vswitch dvs vmware vxlan network list -vds-name Cluster01-VXLAN-
VDS
```

   – View the network statistics for a particular segment ID.

```
esxcli network vswitch dvs vmware vxlan network stats list --vds-name Cluster01-
VXLAN-VDS --vxlan-id 5000
```

   – Retrieve network mapping if some virtual machine communication is occurring.

```
esxcli network vswitch dvs vmware vxlan network mapping list --vds-name
Cluster01-VXLAN-VDS --vxlan-id 5000
```

6. View VXLAN statistics for a VDS Port ID.

```
esxcli network vswitch dvs vmware vxlan network port list --vds-name Cluster01-VXLAN-
VDS --vxlan-id 5000
```

7. View the network statistics for a specific VDS Port ID.

```
esxcli network vswitch dvs vmware vxlan network port list --vds-name Cluster01-VXLAN-
VDS --vxlan-id 5000 vdsport-is 968
```

Monitoring ESXi Hosts

vCenter Server makes performance charts for CPU, memory, disk I/O, networking, and storage available.

You can view performance charts by using the vSphere Client and read about them in the *vSphere Monitoring* documentation. You can also perform some monitoring of your ESXi system by using ESXCLI commands.

Managing Diagnostic Partitions

Your host must have a diagnostic partition, also referred to as dump partition, to store core dumps for debugging and for use by VMware technical support.

A diagnostic partition is on the local disk where the ESXi software is installed by default. You can also use a diagnostic partition on a remote disk shared between multiple hosts. If you want to use a network diagnostic partition, you can install ESXi Dump Collector and configure the networked partition. See Manage Core Dumps with ESXi Dump Collector.

The following considerations apply.

- A diagnostic partition cannot be located on an iSCSI LUN accessed through the software iSCSI or dependent hardware iSCSI adapter. For more information about diagnostic partitions with iSCSI, see General Boot from iSCSI SAN Recommendations in the *vSphere Storage* documentation.
- A standalone host must have a diagnostic partition of 110 MB.
- If multiple hosts share a diagnostic partition on a SAN LUN, configure a large diagnostic partition that the hosts share.
- If a host that uses a shared diagnostic partition fails, reboot the host and extract log files immediately after the failure. Otherwise, the second host that fails before you collect the diagnostic data of the first host might not be able to save the core dump.

Diagnostic Partition Creation

You can use the vSphere Client to create the diagnostic partition on a local disk or on a private or shared SAN LUN. The SAN LUN can be set up with FibreChannel or hardware iSCSI. SAN LUNs accessed through a software iSCSI initiator are not supported.

CAUTION

If two hosts that share a diagnostic partition fail and save core dumps to the same slot, the core dumps might be lost.

If a host that uses a shared diagnostic partition fails, reboot the host and extract log files immediately after the failure.

Diagnostic Partition Management

You can use the **esxcli system coredump** command to query, set, and scan an ESXi system's diagnostic partitions. The *vSphere Storage* documentation explains how to set up diagnostic partitions with the vSphere Client and how to manage diagnostic partitions on a Fibre Channel or hardware iSCSI SAN.

Diagnostic partitions can include, in order of suitability, parallel adapter, block adapter, FC, or hardware iSCSI partitions. Parallel adapter partitions are most suitable and hardware iSCSI partitions the least suitable.

> **IMPORTANT**
> When you list diagnostic partitions, software iSCSI partitions are included. However, SAN LUNs accessed through a software iSCSI initiator are not supported as diagnostic partitions.

Managing Core Dumps

With **esxcli system coredump**, you can manage local diagnostic partitions or set up core dump on a remote server in conjunction with the ESXi Dump Collector.

For information about the ESXi Dump Collector, see the *vSphere Networking* documentation.

Manage Local Core Dumps with ESXCLI

You can use ESXCLI to manage local core dumps.

The following example scenario changes the local diagnostic partition by using ESXCLI. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. Show the diagnostic partition the VMkernel uses and display information about all partitions that can be used as diagnostic partitions.

   ```
   esxcli <conn_options> system coredump partition list
   ```

2. Deactivate the current diagnostic partition.

   ```
   esxcli <conn_options> system coredump partition set --unconfigure
   ```

   The ESXi system is now without a diagnostic partition, and you must immediately set a new one.

3. Set the active partition to `naa.<naa_ID>`.

   ```
   esxcli <conn_options> system coredump partition set --partition=naa.<naa_ID>
   ```

4. List partitions again to verify that a diagnostic partition is set.

   ```
   esxcli <conn_options> system coredump partition list
   ```

   If a diagnostic partition is set, the command displays information about it. Otherwise, the command shows that no partition is activated and configured.

Manage Core Dumps with ESXi Dump Collector

By default, a core dump is saved to the local disk. You can use the ESXi Dump Collector to keep core dumps on a network server for use during debugging.

The ESXi Dump Collector is especially useful for Auto Deploy. The ESXi Dump Collector supports other customization, including sending core dumps to the local disk.

The ESXi Dump Collector is included with the vCenter Server`autorun.exe` application. You can install the ESXi Dump Collector on the same system as the vCenter Server service or on a different Windows or Linux machine. See *vSphere Networking*.
You can configure ESXi hosts to use the ESXi Dump Collector by using the Host Profiles interface of the vSphere Client, or by using ESXCLI. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. Set up an ESXi system to use the ESXi Dump Collector by running **`esxcli system coredump`**.

   ```
   esxcli <conn_options> system coredump network set --interface-name vmk0 --server-ipv4=1-XX.XXX --server-port=6500
   ```

   You must specify a VMkernel port with `--interface-name`, and the IP address and port of the server to send the core dumps to. If you configure an ESXi system that is running inside a virtual machine, you must choose a VMkernel port that is in promiscuous mode.

2. Enable the ESXi Dump Collector.

   ```
   esxcli <conn_options> system coredump network set --enable=true
   ```

3. Check that the ESXi Dump Collector is configured correctly.

   ```
   esxcli <conn_options> system coredump network get
   ```

The host on which you have set up the ESXi Dump Collector sends core dumps to the specified server by using the specified VMkernel NIC and optional port.

Configuring ESXi Syslog Services

All ESXi hosts run a syslog service, which logs messages from the VMkernel and other system components to local files or to a remote host.

You can use the vSphere Client, or use the **`esxcli system syslog`** command to configure the following parameters of the syslog service.

- Remote host and port - Remote host to which syslog messages are forwarded and port on which the remote host receives syslog messages. The remote host must have a log listener service installed and correctly configured to receive the forwarded syslog messages. See the documentation for the syslog service installed on the remote host for information on configuration.
- Transport protocol - Logs can be sent by using UDP, which is the default, TCP, or SSL transports.
- Local logging directory - Directory where local copies of the logs are stored. The directory can be located on mounted NFS or VMFS volumes. Only the `/scratch` directory on the local file system is persistent across reboots.
- Unique directory name prefix - Setting this option to true creates a subdirectory with the name of the ESXi host under the specified logging directory. This method is especially useful if the same NFS directory is used by multiple ESXi hosts.
- Log rotation policies - Sets maximum log size and the number of archives to keep. You can specify policies both globally, and for individual subloggers. For example, you can set a larger size limit for the `vmkernel` log.

After making configuration changes, restart the `vmsyslogd` syslog service by running **`esxcli system syslog reload`**.

For audit events, you should use syslog with either TCP on port 514 or TLS on port 1514. To use unencrypted syslog, you must prefix the entry with `tcp://`. To use encrypted syslog through the TLS protocol, you must prefix the entry with `ssl://`. For security reasons, you should use the TLS protocol for communications. To use the TLS protocol, you must load a public CA certificate onto ESXi that is suitable for the syslog server.

The **`esxcli system syslog`** command allows you to configure the logging behavior of your ESXi system. You can manage the top-level logger and subloggers. The command has the following options.

Option	Description
`mark`	Marks all logs with the specified string.
`reload`	Reloads the configuration, and updates any changed configuration values.
`config get`	Retrieves the current configuration.
`config set`	Sets the configuration. Use one of the following options. • `--logdir=<path>` – Saves logs to a given path. • `--loghost=<host>` – Sends logs to a given host. • `--logdir-unique=<true|false>` – Specifies whether the log should go to a unique subdirectory of the directory specified in `logdir`. • `--default-rotate=<int>` – Default number of log rotations to keep. • `--default-size=<int>` – Size before rotating logs, in KB.
`config logger list`	Shows currently configured subloggers.
`config logger set`	Sets configuration options for a specific sublogger. Use one of the following options. • `--id=<str>` – ID of the logger to configure. Required. • `--reset=<str>` – Resets values to default. • `--rotate=<long>` – Number of rotated logs to keep for a specific logger. Requires `--id`. • `--size=<long>` – Size of logs before rotation for a specific logger, in KB. Requires `--id`.

esxcli system syslog Usage

The following workflow illustrates how you might use **`esxcli system syslog`** for log configuration. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. Show configuration options.

```
esxcli <conn_options> system syslog config get

Default Rotation Size: 1024

Default Rotations: 8

Log Output: /scratch/log

Logto Unique Subdirectory: false
```

```
Remote Host: <none>
```

2. Set all logs to keep twenty rotations before overwriting the oldest log.
   ```
   esxcli <conn_options> system syslog config set --default-rotate=20
   ```
3. Set the rotation policy for VMkernel logs to 10 rotations, rotating at 2 MB.
   ```
   esxcli <conn_options> system syslog config logger --id=vmkernel --size=2048 --
   rotate=10
   ```
4. Send logs to remote host `myhost.mycompany.com`. The logs will use the default transport (UDP) and port (514).
   ```
   esxcli system syslog config set --loghost='myhost.mycompany.com'
   ```
5. Save the local copy of logs to `/scratch/mylogs` and send another copy to the remote host.
   ```
   esxcli <conn_options> system syslog config set --loghost='tcp://
   myhost.mycompany.com:1514' --logdir='/scratch/mylogs'
   ```
   You can set the directory on the remote host by configuring the client running on that host. You can use the vSphere Client to redirect system logs to a remote host by changing the `System.global.logHost` advanced setting.
6. Send a log message to all logs simultaneously.
   ```
   esxcli <conn_options> system syslog mark --message="this is a message!"
   ```
7. Reload the syslog daemon and apply configuration changes.
   ```
   esxcli <conn_options> system syslog reload
   ```

Managing ESXi SNMP Agents

Simple Network Management Protocol (SNMP) allows management programs to monitor and control networked devices.

The host-based embedded SNMP agent is disabled by default. Configuring and enabling the agent requires that you perform the following tasks.

1. Configure SNMP communities. See Configuring SNMP Communities.
2. Configure the SNMP agent. See Configuring the SNMP Agent to Send Traps.

Configuring SNMP Communities

Before you enable the ESXi embedded SNMP agent, you must configure at least one community for the agent.

An SNMP community defines a group of devices and management systems. Only devices and management systems that are members of the same community can exchange SNMP messages. A device or management system can be a member of multiple communities.

To configure SNMP communities, run **`esxcli system snmp set`**, specifying a comma-separated list of communities as shown in the following example.

```
esxcli system snmp set -c public, internal
```

Each time you specify a community with this command, the settings that you specify overwrite the previous configuration.

Configuring the SNMP Agent to Send Traps

You can use the SNMP agent embedded in ESXi to send virtual machine and environmental traps to management systems.

To configure the agent to send traps, you must specify a target address, also referred to as receiver address, the community, and an optional port. If you do not specify a port, the SNMP agent sends traps to UDP port 162 on the target management system by default.

Configure a Trap Destination with ESXCLI

You can use ESXCLI to configure a trap destination and send traps.

Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. Make sure a community is set up.

   ```
   esxcli system snmp get <conn_options>


   Current SNMP agent settings:
   Enabled: 1
   UDP port: 161
   Communities: public
   Notification targets:
   ```

2. Set the target address, port number, and community.

   ```
   esxcli <conn_options> system snmp set -t target.example.com@163/public
   ```

   Each time you specify a target with this command, the settings you specify overwrite all previously specified settings. To specify multiple targets, separate them with a comma.

   You can change the port that the SNMP agent sends data to on the target using the `--targets` option. That port is UDP 162 by default.

3. Enable the SNMP agent if it is not yet running.

   ```
   esxcli <conn_options> system snmp set --enable=yes
   ```

4. Send a test trap to verify that the agent is configured correctly.

   ```
   esxcli <conn_options> system snmp test
   ```

   The agent sends a `warmStart` trap to the configured target.

Retrieving Hardware Information

Commands in different ESXCLI namespaces might display some hardware information, but the **`esxcli hardware`** namespace is specifically intended to give you that information. The namespace includes commands for getting and setting CPU properties, for listing boot devices, and for getting and setting the hardware clock time.

You can also use the **`ipmi`** namespace to retrieve IPMI system event logs (SEL) and sensor data records (SDR). The command supports both **`get`** (single return value) and **`list`** (multiple return values) commands and returns raw sensor information.

See the *`ESXCLI Reference`* for details.

Managing Security

You can use ESXCLI to manage security settings related to audit, certificates, cryptographic functionality, and physical device access.

Configuring and Managing the Audit System and Audit Data

You can record audit data on local storage or manage the audit record on remote hosts. You can use one or multiple syslog servers for remote management.

Enable Local Auditing with ESXCLI

You can use ESXCLI to configure local auditing.

The following example specifies the size of the audit record storage capacity and enables the audit record storage. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

> **NOTE**
> The size of the audit record storage capacity is measured in kilobytes in ESXi 7.0 and earlier, and in megabytes in ESXi 7.0 Update 1 and later.

1. Specify the audit record storage capacity.
   - If you are running the command against ESXi 7.0 or earlier, specify the value of the `--size` parameter in kilobytes.
     ```
     esxcli <conn_options> system auditrecords local set --size=4096
     ```
   - If you are running the command against ESXi 7.0 Update 1 or later, specify the value of the `--size` parameter in megabytes.
     ```
     esxcli <conn_options> system auditrecords local set --size=4
     ```
2. Enable the audit record storage.
   ```
   esxcli <conn_options> system auditrecords local enable
   ```

Enable the Transmission of Audit Records to a Remote Host with ESXCLI

You can use ESXCLI to configure transmitting audit records to a remote host.

You can specify multiple syslog servers, separated by commas. For protocol and port information, see Configuring Syslog Services.
Audit records are transmitted to a remote host as RFC-compliant syslog messages. Audit records have an RFC-specific prefix in the `<NNN>` format, where *NNN* is a packaged value representing the facility and severity, calculated as `(8 * facility) + severity`. The audit `facility` number is 13, and a typical severity is `info`, whose number is 6. By using the formula, the value of *NNN* for an informational audit message is 110, for an audit notice the value is 109, and for an audit error the value is 107.
The following example configures the firewall to permit outbound syslog access, enables remote audit logging, sets the syslog server, and clears the syslog configuration. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. Configure the firewall to permit outbound syslog transmissions.
   ```
   esxcli <conn_options> network firewall ruleset set --ruleset-id=syslog --enabled=true
   ```
   ```
   esxcli <conn_options> network firewall refresh
   ```
2. Enable remote audit logging.
   ```
   esxcli <conn_options> system auditrecords remote enable
   ```
3. Set the syslog server.

```
esxcli <conn_options> system syslog config set --loghost=”ssl://
syslog.example.com:port”
```

4. When configuring encrypted transmissions, you can enable CRL checking.

   NOTE
   The `--crl-check` option performs verification of X.509 CRLs, which are not checked by default in compliance with industry standards. If CRL checks are enabled, all certificates in a certificate chain must provide a CRL link. You should not enable CRL checking unless explicitly required, for example, to satisfy government certification requirements, because of the difficulty in properly configuring an environment that uses CRL checks.

   ```
   esxcli <conn_options> system syslog config set --crl-check=true
   ```

5. When configuring encrypted transmissions, you can enable strict compliance with X.509.

   NOTE
   The `--x509-strict` option performs additional validity checks on CA root certificates during verification. These checks are generally not performed because CA roots are inherently trusted, and might cause incompatibilities with existing, misconfigured CA roots. You should not enable X.509 strict checking unless explicitly required, for example, to satisfy government certification requirements.

   ```
   esxcli <conn_options> system syslog config set --x509-strict=true
   ```

6. Notify the syslog daemon of the configuration changes.

   ```
   esxcli <conn_options> system syslog reload
   ```

Manage CA Certificates with ESXCLI

You can use ESXCLI to manage certificates in the CA certificate store.

You must manage certificates in the CA certificate store when configuring a remote syslog server that uses the TLS protocol for communications. To establish secure communications, you must add the public CA certificate for the remote syslog server to the ESXi CA certificate store.
The following example adds a new CA certificate to the CA certificate store, lists all available certificates in the store, and removes a certificate from the store. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

1. Add a new CA certificate to the CA certificate store.

   ```
   esxcli <conn_options> system security certificatestore add --filename=<local_file>
   ```

2. List all certificates in the CA certificate store.

   ```
   esxcli <conn_options> system security certificatestore list
   ```

3. Remove a certificate from the CA certificate store.
   - Remove a certificate by specifying the file name.

     ```
     esxcli <conn_options> system security certificatestore remove --
     filename=<local_file>
     ```

   - Remove a certificate by specifying the issuer name and serial number.

     ```
     esxcli <conn_options> system security certificatestore remove --
     issuer=<issuer_name> --serial=<serial_number>
     ```

Configuring the Cryptographic Functionality

You can use ESXCLI to manage the cryptographic functionality.

The following examples require restarting the HTTP proxy or a host reboot. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

```
esxcli <conn_options> system settings advanced set -o /UserVars/ESXiVPsDisabledProtocols
-s <value>

esxcli <conn_options> system settings advanced set -o /UserVars/ESXiVPsAllowedCiphers -s
<value>
```

You can use the `/UserVars/ESXiVPsDisabledProtocols` advanced option to specify the protocols that are disabled when establishing secure communications with ESXi. The value is a comma-separated list of protocols. The valid values are `sslv3`, `tlsv1`, `tlsv1.1`, `tlsv1.2`. For example, to disable `sslv3`, `tlsv1`, and `tlsv1.1`, you can run the following command.

```
esxcli <conn_options> system settings advanced set -o /UserVars/ESXiVPsDisabledProtocols
-s "sslv3,tlsv1,tlsv1.1"
```

You can use the `/UserVars/ESXiVPsAllowedCiphers` advanced option to specify the ciphers allowed for secure communcations with ESXi. The value is a colon-separated list of ciphers, in the form required by the OpenSSL `SSL_CTX_set_cipher_list` API. The command has the following syntax.

```
esxcli <conn_options> system settings advanced set -o /UserVars/ESXiVPsAllowedCiphers -s
<cipherlist>
```

The following example can potentially affect guest-related key destruction.

How memory pages are zeroed out for virtual machines and user-space applications is determined by the `/Mem/MemEagerZero` advanced option. This option determines how long residual information resides in memory after it is no longer in use.

When `/Mem/MemEagerZero` is set to 0, which is the default value, memory pages are zeroed when they are allocated to virtual machines and user-space applications. While this prevents exposing information from virtual machines to other clients, previous content can remain present in memory for a long time if the memory is not reused.

For more immediate content destruction, you can set `/Mem/MemEagerZero` to 1. In this case, memory pages are zeroed when a user-space application exits. For virtual machines, memory pages are zeroed when the virtual machine powers off, when its pages are migrated, or when virtual machine memory is reclaimed.

For example, to set `/Mem/MemEagerZero` to 1, use the following command.

```
esxcli <conn_options> system settings advanced set -o /Mem/MemEagerZero -i 1
```

Configuring Virtual Machine Access to Physical Devices

You can use ESXCLI to control access to USB passthrough devices.

By enabling or disabling USB passthrough, you affect all virtual machine access to the affected device.

The following examples show the syntax for listing USB devices and their passthrough status, enabling, and disabling USB passthrough. For enabling and disabling a USB device, you can obtain the required device information from the device's entry in the USB device list. Specify one of the options listed in Connection Options for ESXCLI Host Management Commands in place of `<conn_options>`.

- List USB devices and their passthrough status.
  ```
  esxcli <conn_options> hardware usb passthrough device list
  ```
- Enable USB passthrough.
  ```
  esxcli <conn_options> hardware usb passthrough device enable -d
  busId:dev:VendorID:ProductID
  ```
- Disable USB passthrough.
  ```
  esxcli <conn_options> hardware usb passthrough device disable -d
  busId:dev:VendorID:ProductID
  ```

Managing Configuration Settings

Since ESXi 7.0 Update 3, some configuration settings that were stored in configuration files in earlier releases, are migrated to the ESXi Configuration Store.

You can use ESXCLI and the ESXi Configuration Store CLI to manage the migrated configuration settings. You can run ESXi Configuration Store CLI commands in the ESXi Shell.

Sticky Bit Configuration Files

You can use ESXCLI and the ESXi Configuration Store CLI to manage configuration settings that were part of sticky bit configuration files before ESXi 7.0 Update 3.

The following table contains the file paths to sticky bit files that were used before ESXi 7.0 Update 3, the configuration area of each file, and the commands you can use to manage the corresponding configuration settings.

Configuration Area	File Path	Commands
Hostd	/etc/vmware/hostd/config.xml /etc/vmware/hostd/probe-config.xml /etc/vmware/cpservices/esxtoken/config.xml	You can run the following ESXi Configuration Store CLI commands in the ESXi Shell. /bin/configstorecli config current get -c esx -g services -k hostd -outfile tmp.json /bin/configstorecli config current set -c esx -g services -k hostd -infile tmp.json /bin/configstorecli config current get -c esx -g services -k hostd_probe -outfile tmp.json /bin/configstorecli config current set -c esx -g services -k hostd_probe -infile tmp.json

Table continued on next page

Continued from previous page

Configuration Area	File Path	Commands

		`/bin/configstorecli config current get -c esx -g services -k esx_tokend -outfile tmp.json`
		
		`/bin/configstorecli config current set -c esx -g services -k esx_tokend -infile tmp.json`
		
		NOTE
		You must restart the service after running the commands.
		
		You can check all possible `Hostd` settings by providing the configuration store schema in JSON format by running the following command.
		`/bin/configstorecli schema get -c esx -g services -k hostd`

`Hostd`	`/etc/passwd`
	
	`/etc/shadow`
	
	`/etc/group`
		You can modify local accounts by using `esxcli system account` commands.

`Hostd`	`/etc/vmware/lockdown.conf`
	
	`/etc/vmware/hostd/authorization.xml`
		You can modify permissions by using `esxcli system permission` commands.

`vpxa`	`etc/vmware/vpxa/vpxa.cfg`	You can retrieve the current configuration by running the following ESXi Configuration Store CLI command.
		`configstorecli config current get -c esx -g services -k vpxa -outfile a.json`
		
		NOTE
		If you do not specify the `outfile` parameter, the result is written to standard output.
		
		You can modify the current configuration by performing the following steps.
		1. Retrieve the `a.json` file.
		`configstorecli config current get -c esx -g`

Table continued on next page

Continued from previous page

Configuration Area	File Path	Commands

		`services -k vpxa -outfile a.json`
		2. Edit the `a.json` file to modify the values of the properties you want to update.
		3. Update the `a.json` file.
		`configstorecli config current set -c esx -g services -k vpxa -infile a.json`
		4. Restart the `vpxa` service.
		`/etc/init.d/vpxa restart`

`ntp`	`/etc/ntp.conf`	`esxcli system ntp config get` `esxcli system ntp set`

`ptp`	`/etc/ptp.conf`	`esxcli system ptp get`
		`esxcli system ptp set`
		`esxcli network ip interface tag add`

`networking`	`/etc/hosts`	`esxcli network ip hosts list`
		`esxcli network ip hosts add`
		`esxcli network ip hosts list remove`

`networking`	`/etc/resolv.conf`	You can access DNS information by using `esxcli network ip dns` commands.

Namespaces from esx.conf

You can use ESXCLI to manage namespaces that were part of the `esx.conf` configuration file before ESXi 7.0 Update 3.

The following table contains the namespaces, the configuration area of each namespace, and the ESXCLI commands you can use to manage each namespace.

Configuration Area	Namespace	Commands

`VMKernel`	`Irq`	• Get
		`esxcli system settings advanced list -t /Irq`
		`esxcli system settings advanced list -o /Irq/<option_name>`
		• Set

Table continued on next page

Continued from previous page

Configuration Area	Namespace	Commands
		`esxcli system settings advanced set -o /Irq/ <option_name>`

VMKernel	Misc	• Get
		`esxcli system settings advanced list -t /Misc`
		`esxcli system settings advanced list -o /Misc/ <option_name>`
		• Set
		`esxcli system settings advanced set -o /Misc/ <option_name>`

VMKernel	Net	• Get
		`esxcli system settings advanced list -t /Net`
		`esxcli system settings advanced list -o /Net/ <option_name>`
		• Set
		`esxcli system settings advanced set -o /Net/ <option_name>`

VMKernel	Mem	• Get
		`esxcli system settings advanced list -t /Mem`
		`esxcli system settings advanced list -o /Mem/ <option_name>`
		• Set
		`esxcli system settings advanced set -o /Mem/ <option_name>`

VMKernel	Cpu	• Get
		`esxcli system settings advanced list -t /Cpu`
		`esxcli system settings advanced list -o /Cpu/ <option_name>`
		• Set
		`esxcli system settings advanced set -o /Cpu/ <option_name>`

VMKernel	Numa	• Get

Table continued on next page

Continued from previous page

Configuration Area	Namespace	Commands

		`esxcli system settings advanced list -t /Numa`
		`esxcli system settings advanced list -o /Numa/<option_name>`
		• Set
		`esxcli system settings advanced set -o /Numa/<option_name>`

VMKernel	LPage	• Get
		`esxcli system settings advanced list -t /LPage`
		`esxcli system settings advanced list -o /LPage/<option_name>`
		• Set
		`esxcli system settings advanced set -o /LPage/<option_name>`

VMKernel	Disk	• Get
		`esxcli system settings advanced list -t /Disk`
		`esxcli system settings advanced list -o /Disk/<option_name>`
		• Set
		`esxcli system settings advanced set -o /Disk/<option_name>`

VMKernel	FileSystem	• Get
		`esxcli system settings advanced list -t /FileSystem`
		`esxcli system settings advanced list -o /FileSystem/<option_name>`
		• Set
		`esxcli system settings advanced set -o /FileSystem/<option_name>`

VMKernel	Migrate	• Get `esxcli system settings advanced list -t /Migrate`

Table continued on next page

Continued from previous page

Configuration Area	Namespace	Commands

		`esxcli system settings advanced list -o /Migrate/<option_name>`
		• Set
		`esxcli system settings advanced set -o /Migrate/<option_name>`

VMKernel	Scsi	• Get
		`esxcli system settings advanced list -t /Scsi`
		`esxcli system settings advanced list -o /Scsi/<option_name>`
		• Set
		`esxcli system settings advanced set -o /Scsi/<option_name>`

VMKernel	User	• Get
		`esxcli system settings advanced list -t /User`
		`esxcli system settings advanced list -o /User/<option_name>`
		• Set
		`esxcli system settings advanced set -o /User/<option_name>`

VMKernel	UserMem	• Get
		`esxcli system settings advanced list -t /UserMem`
		`esxcli system settings advanced list -o /UserMem/<option_name>`
		• Set
		`esxcli system settings advanced set -o /UserMem/<option_name>`

VMKernel	UserObj	• Get
		`esxcli system settings advanced list -t /UserObj`
		`esxcli system settings advanced list -o /UserObj/<option_name>`
		• Set

Table continued on next page

Continued from previous page

Configuration Area	Namespace	Commands
		`esxcli system settings advanced set -o /UserObj/<option_name>`

VMKernel	World	• Get
		`esxcli system settings advanced list -t /World`
		`esxcli system settings advanced list -o /World/<option_name>`
		• Set
		`esxcli system settings advanced set -o /World/<option_name>`

VMKernel	NFS	• Get
		`esxcli system settings advanced list -t /NFS`
		`esxcli system settings advanced list -o /NFS/<option_name>`
		• Set
		`esxcli system settings advanced set -o /NFS/<option_name>`

VMKernel	NFS41	• Get
		`esxcli system settings advanced list -t /NFS41`
		`esxcli system settings advanced list -o /NFS41/<option_name>`
		• Set
		`esxcli system settings advanced set -o /NFS41/<option_name>`

VMKernel	VMKTRACING	• Get
		`esxcli system settings advanced list -t /VMKTRACING`
		`esxcli system settings advanced list -o /VMKTRACING/<option_name>`
		• Set
		`esxcli system settings advanced set -o /VMKTRACING/<option_name>`

VMKernel	VMFS3	• Get

Table continued on next page

Continued from previous page

Configuration Area	Namespace	Commands

		`esxcli system settings advanced list -t /VMFS3`
		`esxcli system settings advanced list -o /VMFS3/<option_name>`
		• Set
		`esxcli system settings advanced set -o /VMFS3/<option_name>`

VMKernel	BufferCache	• Get
		`esxcli system settings advanced list -t /BufferCache`
		`esxcli system settings advanced list -o /BufferCache/<option_name>`
		• Set
		`esxcli system settings advanced set -o /BufferCache/<option_name>`

VMKernel	BufferCache2	• Get
		`esxcli system settings advanced list -t /BufferCache2`
		`esxcli system settings advanced list -o /BufferCache2/<option_name>`
		• Set
		`esxcli system settings advanced set -o /BufferCache2/<option_name>`

VMKernel	LVM	• Get
		`esxcli system settings advanced list -t /LVM`
		`esxcli system settings advanced list -o /LVM/<option_name>`
		• Set
		`esxcli system settings advanced set -o /LVM/<option_name>`

VMKernel	DataMover	• Get `esxcli system settings advanced list -t /DataMover`

Table continued on next page

Continued from previous page

Configuration Area	Namespace	Commands

		`esxcli system settings advanced list -o /DataMover/<option_name>`
		• Set
		`esxcli system settings advanced set -o /DataMover/<option_name>`

VMKernel	COW	• Get
		`esxcli system settings advanced list -t /COW`
		`esxcli system settings advanced list -o /COW/<option_name>`
		• Set
		`esxcli system settings advanced set -o /COW/<option_name>`

VMKernel	SE	• Get
		`esxcli system settings advanced list -t /SE`
		`esxcli system settings advanced list -o /SE/<option_name>`
		• Set
		`esxcli system settings advanced set -o /SE/<option_name>`

VMKernel	Power	• Get
		`esxcli system settings advanced list -t /Power`
		`esxcli system settings advanced list -o /Power/<option_name>`
		• Set
		`esxcli system settings advanced set -o /Power/<option_name>`

VMKernel	FSS	• Get
		`esxcli system settings advanced list -t /FSS`
		`esxcli system settings advanced list -o /FSS/<option_name>`
		• Set

Table continued on next page

Continued from previous page

Configuration Area	Namespace	Commands
		`esxcli system settings advanced set -o /FSS/ <option_name>`

VMKernel	FDS	• Get
		`esxcli system settings advanced list -t /FDS`
		`esxcli system settings advanced list -o /FDS/ <option_name>`
		• Set
		`esxcli system settings advanced set -o /FDS/ <option_name>`

VMKernel	FT	• Get
		`esxcli system settings advanced list -t /FT`
		`esxcli system settings advanced list -o /FT/ <option_name>`
		• Set
		`esxcli system settings advanced set -o /FT/ <option_name>`

VMKernel	DirentryCache	• Get
		`esxcli system settings advanced list -t /DirentryCache`
		`esxcli system settings advanced list -o /DirentryCache/ <option_name>`
		• Set
		`esxcli system settings advanced set -o /DirentryCache/ <option_name>`

VMKernel	HBR	• Get
		`esxcli system settings advanced list -t /HBR`
		`esxcli system settings advanced list -o /HBR/ <option_name>`
		• Set
		`esxcli system settings advanced set -o /HBR/ <option_name>`

Table continued on next page

Continued from previous page

Configuration Area	Namespace	Commands

VMKernel	VProbes	• Get
		`esxcli system settings advanced list -t /VProbes`
		`esxcli system settings advanced list -o /VProbes/<option_name>`
		• Set
		`esxcli system settings advanced set -o /VProbes/<option_name>`

VMKernel	Digest	• Get
		`esxcli system settings advanced list -t /Digest`
		`esxcli system settings advanced list -o /Digest/<option_name>`
		• Set
		`esxcli system settings advanced set -o /Digest/<option_name>`

VMKernel	CBRC	• Get
		`esxcli system settings advanced list -t /CBRC`
		`esxcli system settings advanced list -o /CBRC/<option_name>`
		• Set
		`esxcli system settings advanced set -o /CBRC/<option_name>`

VMKernel	PageRetire	• Get
		`esxcli system settings advanced list -t /PageRetire`
		`esxcli system settings advanced list -o /PageRetire/<option_name>`
		• Set
		`esxcli system settings advanced set -o /PageRetire/<option_name>`

VMKernel	SvMotion	• Get `esxcli system settings advanced list -t /SvMotion`

Table continued on next page

Continued from previous page

Configuration Area	Namespace	Commands

		`esxcli system settings advanced list -o /SvMotion/<option_name>`
		• Set
		`esxcli system settings advanced set -o /SvMotion/<option_name>`

VMKernel	Nmp	• Get
		`esxcli system settings advanced list -t /Nmp`
		`esxcli system settings advanced list -o /Nmp/<option_name>`
		• Set
		`esxcli system settings advanced set -o /Nmp/<option_name>`

VMKernel	SunRPC	• Get
		`esxcli system settings advanced list -t /SunRPC`
		`esxcli system settings advanced list -o /SunRPC/<option_name>`
		• Set
		`esxcli system settings advanced set -o /SunRPC/<option_name>`

VMKernel	XvMotion	• Get
		`esxcli system settings advanced list -t /XvMotion`
		`esxcli system settings advanced list -o /XvMotion/<option_name>`
		• Set
		`esxcli system settings advanced set -o /XvMotion/<option_name>`

VMKernel	RCU	• Get
		`esxcli system settings advanced list -t /RCU`
		`esxcli system settings advanced list -o /RCU/<option_name>`
		• Set

Table continued on next page

Continued from previous page

Configuration Area	Namespace	Commands
		`esxcli system settings advanced set -o /RCU/<option_name>`

VMKernel	VVOL	• Get
		`esxcli system settings advanced list -t /VVOL`
		`esxcli system settings advanced list -o /VVOL/<option_name>`
		• Set
		`esxcli system settings advanced set -o /VVOL/<option_name>`

VMKernel	VFLASH	• Get
		`esxcli system settings advanced list -t /VFLASH`
		`esxcli system settings advanced list -o /VFLASH/<option_name>`
		• Set
		`esxcli system settings advanced set -o /VFLASH/<option_name>`

VMKernel	ISCSI	• Get
		`esxcli system settings advanced list -t /ISCSI`
		`esxcli system settings advanced list -o /ISCSI/<option_name>`
		• Set
		`esxcli system settings advanced set -o /ISCSI/<option_name>`

VMKernel	VSCSIStats	• Get
		`esxcli system settings advanced list -t /VSCSIStats`
		`esxcli system settings advanced list -o /VSCSIStats/<option_name>`
		• Set
		`esxcli system settings advanced set -o /VSCSIStats/<option_name>`

VMKernel	PMem	• Get

Table continued on next page

Continued from previous page

Configuration Area	Namespace	Commands

		`esxcli system settings advanced list -t /PMem`
		`esxcli system settings advanced list -o /PMem/<option_name>`
		• Set
		`esxcli system settings advanced set -o /PMem/<option_name>`

VMKernel	VmkAccess	• Get
		`esxcli system settings advanced list -t /VmkAccess`
		`esxcli system settings advanced list -o /VmkAccess/<option_name>`
		• Set
		`esxcli system settings advanced set -o /VmkAccess/<option_name>`

VMKernel	USB	• Get
		`esxcli system settings advanced list -t /USB`
		`esxcli system settings advanced list -o /USB/<option_name>`
		• Set
		`esxcli system settings advanced set -o /USB/<option_name>`

VMKernel	VisorFS	• Get
		`esxcli system settings advanced list -t /VisorFS`
		`esxcli system settings advanced list -o /VisorFS/<option_name>`
		• Set
		`esxcli system settings advanced set -o /VisorFS/<option_name>`

VMKernel	Vmknvme	• Get `esxcli system settings advanced list -t /Vmknvme`

Table continued on next page

Continued from previous page

Configuration Area	Namespace	Commands

		`esxcli system settings advanced list -o /Vmknvme/<option_name>`
		• Set
		`esxcli system settings advanced set -o /Vmknvme/<option_name>`

VMKernel	LoadESX	• Get
		`esxcli system settings advanced list -t /LoadESX`
		`esxcli system settings advanced list -o /LoadESX/<option_name>`
		• Set
		`esxcli system settings advanced set -o /LoadESX/<option_name>`

VSAN	CMMDS	• Get
		`esxcli system settings advanced list -t /CMMDS`
		`esxcli system settings advanced list -o /CMMDS/<option_name>`
		• Set
		`esxcli system settings advanced set -o /CMMDS/<option_name>`

VSAN	LSOM	• Get
		`esxcli system settings advanced list -t /LSOM`
		`esxcli system settings advanced list -o /LSOM/<option_name>`
		• Set
		`esxcli system settings advanced set -o /LSOM/<option_name>`

VSAN	Virsto	• Get
		`esxcli system settings advanced list -t /Virsto`
		`esxcli system settings advanced list -o /Virsto/<option_name>`
		• Set

Table continued on next page

Continued from previous page

Configuration Area	Namespace	Commands
		`esxcli system settings advanced set -o /Virsto/<option_name>`

VSAN	VSAN	• Get
		`esxcli system settings advanced list -t /VSAN`
		`esxcli system settings advanced list -o /VSAN/<option_name>`
		• Set
		`esxcli system settings advanced set -o /VSAN/<option_name>`

VSAN	VSAN-iSCSI	• Get
		`esxcli system settings advanced list -t /VSAN-iSCSI`
		`esxcli system settings advanced list -o /VSAN-iSCSI/<option_name>`
		• Set
		`esxcli system settings advanced set -o /VSAN-iSCSI/<option_name>`

VSAN	VSAN-DP	• Get
		`esxcli system settings advanced list -t /VSAN-DP`
		`esxcli system settings advanced list -o /VSAN-DP/<option_name>`
		• Set
		`esxcli system settings advanced set -o /VSAN-DP/<option_name>`

VSAN	UPIT	• Get
		`esxcli system settings advanced list -t /UPIT`
		`esxcli system settings advanced list -o /UPIT/<option_name>`
		• Set
		`esxcli system settings advanced set -o /UPIT/<option_name>`

OVF Tool User Guide

This *OVF Tool User Guide* provides information about how to use VMware® OVF Tool to package virtual machines and vApps into Open Virtualization Format (OVF) standard packages.

Revision History

Revision History shows the changes made for various versions of this manual.

Table 120: Revision History

Revision	Description
09/2023	Revised with vSphere 8.0 U2. New `vmxFromConfigFile` option. Updated procedure for signing OVF packages.
06/2023	Changed chapter and section titles for search engine optimization.
04/2023	Revised with vSphere 8.0 U1. Deleted references to discontinued products, added pull mode.
09/2022	Revised with vSphere 8.0, including new virtual TPM support (vTPM).
09/2021	Reissued with vSphere 7.0 Update 3.
10/2020	Clarified password percent encoding for Update 1.
04/2020	Updated with new flags and features for vSphere 7.0 release.
04/2019	Clarified OVF Tool with ESXi as target.
05/2018	OVF Tool 4.3.0. Large upload retry, SHA digest fix, NVM and PMEM, Curl update, better logging, VBS and TPM support.
10/2016	OVF Tool 4.2.0. New options for SSL version and cipher list, NVRAM support for EFI boot.
02/2016	Mention need for explicit extraConfig flags on ESXi hosts.
03/2015	OVF Tool 4.1.0 User's Guide. Added DVS port group, and **`--proxy`** option for vSphere or vCloud.
12/2014	Increased Open SSL compatibility version to 1.0.1j. See http://www.openssl.org/news/vulnerabilities.html.
10/2014	OVF Tool 4.0.0 User's Guide. Added --allowAllExtraConfig and --decodeBase64 command line options.
06/2014	OVF Tool 3.5.2 User's Guide. Increased security for Open SSL.
04/2014	OVF Tool 3.5.1 User's Guide.
08/2013	OVF Tool 3.5.0 User's Guide. Includes new command line options.
08/2012	OVF Tool 3.0.1 User's Guide.
08/2011	OVF Tool 2.1 User's Guide.
06/2010	OVF Tool 2.0.1 Guide.
05/2009	OVF Tool 1.0 Guide.

Intended Audience

This book is intended for anyone who needs to convert an OVF package to a virtual machine, or a virtual machine to an OVF package. Users typically include: system administrators, software developers, QA engineers, and anyone who need to package or unpackage virtual machines using open industry standards.

Document Feedback

VMware welcomes your suggestions for improving our documentation. If you have comments, you can post your feedback in topics on the documentation website.

VMware OVF Tool

Open Virtualization Format (OVF) is an industry standard to describe metadata about virtual machine images in XML format. VMware OVF Tool is a command-line utility that helps users import and export OVF packages to and from a wide variety of VMware products.

The Open Virtualization Format

VMware implemented a tool for importing and exporting virtual machines in OVF standard format.

The OVF Standard

The OVF specification describes a secure, portable, efficient, and flexible method to package and distribute virtual machines and components. It originated from the Distributed Management Task Force (DMTF) after vendor initiative. Companies that contributed to the OVF standard include VMware, Dell, HP, IBM, Microsoft, XenSource, and Citrix.

Version 1.1 was published in January 2010, which supersedes the 1.0 specification published April 2009, and is available on the DMTF Web site, along with a white paper.

- Specification: http://www.dmtf.org/standards/published_documents/DSP0243_1.1.0.pdf
- Whitepaper: http://www.dmtf.org/standards/published_documents/DSP2017_1.0.0.pdf

Benefits of OVF

Using OVF to distribute virtual machines has the following benefits:

- Ease of use. When users receive a package in OVF format, they do not have to unzip files, execute binaries, or convert disk formats. Adding a vApp can be as simple as typing a URL and clicking Install.
- Virtual hardware validation. OVF supports fast and robust hardware validation. You do not have to install a complete virtual machine before determining whether it is compatible with an ESXi host (for example, because it uses IDE virtual disks).
- Metadata inclusion. Additional metadata, such as an end-user license agreement, can be packaged with the OVF and displayed before installation.
- Optimized download from the Internet. Large virtual disks are compressed for fast download and to reduce disk space for large template libraries.

Using the VMware OVF Tool

The VMware OVF Tool is available on many platforms and can be downloaded if not built into a product.

VMware Platforms Using the OVF Standard

VMware supports the OVF standard on the following platforms:

- The OVF Tool is useful with many VMware products, including Workstation, Player, VMware Fusion, ESXi hosts, vCenter Server, and vCloud Director.
- OVF version 0.9 and 1.0 are supported for import and export by VMware products.
- VMware Studio can generate OVF packages.

OVF support is built into the vSphere Client that installs from and is compatible with vCenter Server.

Setting Up the OVF Tool

You can find the latest information about System Requirements, supported VMware software and platforms, installation, and known issues by reading the latest release notes located at the following web page:

https://developer.broadcom.com/tools/open-virtualization-format-ovf-tool/latest/

OVF Tool Highlights

The OVF Tool provides the following key features:

- Supports import and generation of OVA packages (OVA is part of the OVF standard, and contains all the files of a virtual machine or vApp in a single file.)
- Directly converts between any vSphere, vCloud Director, VMX, or OVF source format to any vSphere, vCloud Director, VMX, or OVF target format
- Accesses OVF sources using HTTP, HTTPS, FTP, or from a local file
- Deploys and exports vApp configurations on vSphere 4.0 (and all newer) targets and on vCloud Director 1.5 (and all newer) targets
- Provides options to power on a VM or vApp after deployment, and to power off a virtual machine or vApp before exporting (caution advised)
- Show information about the content of any source in probe mode
- Provides context sensitive error messages for vSphere and vCloud Director sources and targets, showing possible completions for common errors, such as an incomplete vCenter inventory path or missing datastore and network mappings
- Provides an optional output format to support scripting when another program calls OVF Tool
- Uses new optimized upload and download API (optimized for vSphere 4.0 and newer)
- Signs OVF packages and validates OVF package signatures
- Validates XML Schema of OVF 1.0 and OVF 1.1 descriptors
- Import and export of OVF packages into a vApprun 1.0 workspace.
  For more information about vApprun, see http://labs.vmware.com/flings/vapprun.

Examples of OVF Tool Use

OVF import and export are built into the vSphere Client and the vSphere Web Client, and various other enterprise oriented VMware products.

For VMware products without built-in OVF support, or when you need to accomplish specialized OVF operations, you can download the OVF Tool over the Web.

Importing and Exporting OVF Using vSphere Client

Using the vSphere 5.0 or 4.0 Client, you can import an OVF package and export a vApp into an OVF package. For example, to import an OVF package using vSphere Client 4:

Click **File › Deploy OVF Template**.

For example, to export a vApp into an OVF package using vSphere Client 4:

Click **File › Export OVF Template** > **Export**.

Using the vSphere Client 2.5, you can import an OVF virtual machine into an ESXi host and export a virtual machine to an OVF file (note that vSphere Client 2.5 is limited to OVF 0.9). For example, to import an OVF vApp into an ESXi host using vSphere Client 2.5:

Click **File › Virtual Appliance › Import** .

For example, to export a virtual machine to an OVF file using vSphere Client 2.5:

Click **File › Virtual Appliance › Export** .

OVF packages imported or exported by OVF Tool are completely compatible with packages imported or exported by the vSphere Client or the vSphere Client.

VMware OVF Tool Delta Disk Facilities

VMware OVF Tool automatically compresses disk files. In the streaming VMDK files that OVF Tool generates, the tool compresses each 64KB disk grain. It is possible to achieve even better compression using the **`--compress`** option. In addition, if a package contains multiple virtual machines, it is possible to compress an OVF package even more using a technique called delta disk compression. This compression algorithm is invoked using the **`--makeDeltaDisks`** option.

```
ovftool --makeDeltaDisks package.ovf output-dir/
```

Delta disk compression identifies disk segments that are equal and combines these equal parts in a parent disk. This process prevents storing the same segment twice.

As an example, consider a software solution that consists of an Apache Web server virtual machine and a MySQL database virtual machine, both installed on top of a single-disk Ubuntu server. The two virtual machines were created with the following process:

1. Create a plain Ubuntu installation on one virtual machine.
2. Clone the virtual machine.
3. Install Apache on the first virtual machine.
4. Install MySQL on the second virtual machine.

Using delta disk compression on the two virtual machine disks creates a parent disk containing all of the information they share, which is essentially the entire operation system and two child disks containing the MySQL and Apache parts.

A plain Ubuntu server can use 400–500MB of space, and two would use 800–1000MB of space. By contrast, using delta disk compression, an OVF package with these two servers uses only 400–500MB (plus the size of the MySQL and Apache installations), which saves 400–500MB by not duplicating the Ubuntu server.

vSphere 4 and later support the deployment of OVF packages that contain delta disk hierarchies. Any number of disks can be combined creating various disk trees and saving more space.

For delta disk compression, keep in mind the following:

- Only disks with equal capacity can be combined. If you expect to use delta disk compression, you must keep disk capacities equal.
- Delta disk compression necessitates that segments that might be put in a parent disk are at the same offset from the beginning of their respective files. In the Ubuntu example, if the setup varies between the two installations, it can completely offset each segment on one of the disks from the segments on the other disk. In this case, delta disk compression does not produce any significant disk space savings. This is why the example specified cloning the Ubuntu server before installing the MySQL and Apache parts, respectively.
- Delta disk compression takes OVF packages and vSphere and VMX files as input, but not OVA packages.
- The delta disk compression algorithm needs to read the contents of each disk up to two times. It might make sense to invoke OVF Tool on a local copy of the OVF package.
- The delta disk compression algorithm always generates an OVF package in the given output directory. To convert this OVF package into an OVA package, reinvoke OVF Tool.

Using OVF Tool Commands

The VMware OVF Tool is a command-line utility that supports importing and exporting of OVF packages, VMX files, or virtual machines from ESXi hosts and other VMware products.

Definitions and Command Syntax

A source location or source URL locator refers to an OVF package, VMX file, or virtual machine in a VMware product, such as Workstation, Fusion, ESXi, vCenter Server, or vCloud Director.

A target location or destination URL locator specifies either a file location, or a location within a VMware product, such as Workstation, Fusion, ESXi, vCenter Server, or vCloud Director.

Run VMware OVF Tool From the Command Line

1. At the command-line prompt, run the OVF Tool as follows

   ```
   ovftool <source locator> <target locator>
   ```

   <source locator> and **<target locator>** are paths to the source and target for the virtual machine, OVF package, OVA package, or vSphere location. See Command-Line Options for options. Table : OVF Tool Definitions of Source and Target Locators describes the source and target locators. For details, see Specifying a Locator.

   If you are using an operating system where spaces are not allowed in paths on the command line, and need the full path to run OVF Tool, enclose the path in quotes as shown below:

   ```
   "/Applications/VMware OVF Tool/ovftool" --help
   ```

2. If you want to specify additional options, type them before the source and target locators.

   ```
   ovftool <options> <source locator> <target locator>
   ```

3. To display all options, type `ovftool -h`.

   Probe mode allows you to investigate the contents of a source. To invoke probe mode, use the **ovftool** command with only a source and no target. OVF Tool prints information about the source such as hardware, EULA, and OVF properties.

   ```
   ovftool <options> <source locator>
   ```

   Use probe mode to examine an OVF package before deploying it. For example, you can examine the download and deployment sizes, determine the set of networks to be mapped, determine the OVF properties to be configured, read the EULA, and determine the virtual hardware requirements.

   The probe operation is fast, as it only needs to access the OVF descriptor. It does not need to download the entire OVA or VMDK files. Probe mode also validates the certificate if the source is signed. For details about Probe Mode and sample output, see Using OVF Tool Probe Mode.

Run OVF Tool With ESXi and vCenter Target Locations

If you are deploying with the **ovftool** command targeting vCenter Server, you can use the command line options in the table below to specify parameters when you deploy.

If you are deploying with the **ovftool** command targeting an ESXi host, you must “inject” the parameters into the resulting VM when it is powered on. This is because the ESXi host lacks a cache to store the OVF parameters, as with vCenter Server. Therefore, you must use the **--X:injectOvfEnv** debug option with the **--poweron** flag in the command line if you are deploying a virtual machine targeting ESXi. Example below. (You can also do this using the Create a VM from an OVA/OVF option in the ESXi host client, then browse to the .ova file.)

```
>./ovftool/ovftool\

--name="Cloudvm_2074586_with_inject"\

--X:injectOvfEnv\

--X:logFile=ovftool.log\

--X:logLevel=trivia\
```

```
--acceptAllEulas\
-ds=cl-storage-1\
-dm=thin\
--net:'Network 1=VM Network'\
--X:enableHiddenProperties\
--noSSLVerify\
--allowExtraConfig\
--machineOutput\
--prop:vami.netmask0.VMware_vCenter_Server_Appliance=255.255.255.0\
--prop:guestinfo.cis.appliance.net.prefix=24\
--prop:guestinfo.cis.appliance.net.gateway=X.X.X.123\
--prop:guestinfo.cis.appliance.time.tools-sync=True\
--prop:vami.gateway.VMware_vCenter_Server_Appliance=X.X.X.123\
--prop:guestinfo.cis.appliance.net.dns.servers=X.X.X.1,X.X.X.2\
--prop:vami.ip0.VMware_vCenter_Server_Appliance=X.X.X.145\
--prop:guestinfo.cis.appliance.root.passwd=vmware\
--prop:guestinfo.cis.appliance.net.addr=X.X.X.145\
--prop:vami.DNS.VMware_vCenter_Server_Appliance=X.X.X.1,X.X.X.2\
--prop:vami.vmname=vmc-srm-vc10\
--prop:guestinfo.cis.appliance.root.shell=/bin/bash\
--prop:guestinfo.cis.vmdir.first-instance=True\
--prop:guestinfo.cis.appliance.ssh.enabled=True\
--prop:guestinfo.cis.appliance.net.mode=static\
--prop:guestinfo.cis.appliance.net.addr.family=ipv4\
--prop:guestinfo.cis.vmdir.domain-name=vsphere.local\
--prop:guestinfo.cis.vmdir.password=vmware\
--powerOn\
--X:waitForIp\
http://<directory_w/cloudvm>/VMware-vCenter-Server-Appliance-6.0.0.XXXX-XXXXXXX_OVF10.ovf\
vi://root:pwd@vm_name
```

You will need to replace the variables (IP addresses, build numbers, root password, and VM names) in the above example with values from your own system.

Deploy a Virtual Machine to Static DVS Port Group

At this time, you can not use the OVF Tool to deploy a virtual machine to a static port group.

To work around this issue:

1. Use vCenter Server to create an ephemeral port on the desired network.
2. Deploy the appliance to that port group on an ESXi host.
3. Switch the appliance over to that static port group.
4. Use vCenter Server to delete the ephemeral port group.

Command-Line Options

This section contains a long table of OVF Tool command-line options.

For every command, you specify source and target locators. Table : OVF Tool Definitions of Source and Target Locators defines each locator type.

Table 121: OVF Tool Definitions of Source and Target Locators

Locator	Definition
`<source locator>`	Path to source, which must be a virtual machine, vApp, vApprun workspace entity, or an OVF package. The source locator can be one of the following: • A path to an OVF or OVA file (a local file path, or an HTTP, HTTPS, or FTP URL). • A virtual machine (a local file path to a .vmx file). • A vSphere locator identifying a virtual machine or vApp on vCenter, ESXi, or VMware Server. • A vCloud Director locator identifying a virtual machine or a vApp in vCloud Director. • A local file path to a vApprun workspace entity.
`<target locator>`	The target locator can be one of the following: • A local file path for VMX, OVF, OVA, or vApprun workspace. • A vSphere locator identifying a cluster, host, or a vSphere location. • A vCloud Director locator identifying a virtual machine or a vApp in vCloud Director.

Table : OVF Tool Command-Line Options shows all the command-line options. Options perform actions only between certain source and target types. The table shows source and target types for each option. If you specify an option using an irrelevant source or target type, the command does nothing.

All options can be set using the form `--option=value`. Binary options can be switched on and off explicitly. For example: `--option=true` and `--option=false`.

Table 122: OVF Tool Command-Line Options

Option's Long Name (and Short Name)	Source Types	Target Types	Description
`--acceptAllEulas`	OVF, OVA	N/A	Accepts all end-user licenses agreements (EULAs) without being prompted. Binary option.
`--addDevice`			Adds a virtual device for all VirtualHardwareSections.

Table continued on next page

Continued from previous page

Option's Long Name (and Short Name)	Source Types	Target Types	Description
--allowExtraConfig			Lets you specify the extra config options that can be converted to `.vmx` format. These options are a security risk as they control low-level and potentially unsafe options on the VM. Each option must be specified using a series of key value pairs (sometimes referred to as a white list).
--annotation	All		Adds annotation to vi, vmx, vapprun, vCloud, OVF, and OVA source locators.
--authdPortSource	vSphere	vSphere	Overrides default VMware **authd** port (902) when using a host as source or as target.
--authdPortTarget	vSphere	vSphere	Overrides the default VMware **authd** port (902) when using a host as target.
--chunkSize	N/A	OVF, OVA	Specifies the chunk size to use for files in a generated OVF or OVA package. The default is not to chunk. If you don't specify a unit for chunk size, the chunk size is assumed to be in megabytes (mb). Accepted units are b, kb, mb, gb. Example: 2gb or 100kb. When using this option, all output files (except the OVF descriptor, manifest and certificate files) are sliced into the specified chunk size. This is useful if you need to transport an OVF package on a series of 800MB CD-ROMs, or are only able to create files up to 2GB on FAT32 file systems. When you use chunking with the OVA package option, the result is similar to OVF because all files are chunked, but the OVA package is still a single file.
--compress	N/A	OVF, OVA	Compresses the disk when given an OVF or OVA target locator. The value must be between 1 and 9. Use 9 for the slowest processing time, but best compression. Use 1 for the fastest processing time, but least compression.
--computerName			Sets the computer name in the guest virtual using the syntax **--computerName:<VMID>=<value>**. Only applies to vCloud targets of version 5.5 or later.
--configFile	vSphere	vSphere	Specifies a configuration file containing long-name options and settings. Useful for scripting, this overrides the local `ovftool.cfg` in the current directory, which overrides the global configuration file in `$HOME/ovftool.cfg` on Linux and OS X, or C:\Users\%USER%\AppData\Roaming\VMware\ovftool.cfg on Windows.
--coresPerSocket			Specifies the distribution of the total number of CPUs over a number of virtual sockets using the syntax --

Table continued on next page

Continued from previous page

Option's Long Name (and Short Name)	Source Types	Target Types	Description
			coresPerSocket:<VMID>=<value>. Only applies to vCloud targets of version 5.5 or later.
--datastore or **-ds**	N/A	vSphere	Target datastore name for a vSphere locator.
--decodeBase64			Enables Base64 decoding for values in the OVFTool command line.
--defaultStorageProfile			The storage profile for all VMs in the OVF package. The value should be an SPBM profile ID. Only applies to VI targets of version 5.5 or later.
--defaultStorageRawProfile			The storage profile for all VMs in the OVF package. The value should be a raw SPBM profile. The value overwrites that in **--defaultStorageProfile**. Only applies to VI targets of version 5.5 or later.
--deploymentOption	OVF, OVA	N/A	Deployment options for a deployed OVF package, if the source OVF package supports multiple options. An OVF package can contain several deployment configurations. This option allows you to select which configuration to use when deploying to the vSphere target.
--disableVerification	OVF, OVA	N/A	Skips validation of signature and certificate. Binary option.
--diskMode or **-dm**	N/A	VMX, vApprun,vSphere	Select target disk format. Supported formats are: monolithicSparse, monolithicFlat, twoGbMaxExtentSparse, twoGbMaxExtentFlat, seSparse (vSphere target), eagerZeroedThick (vSphere target), thin (vSphere target), thick (vSphere target), sparse, and flat.
--diskSize			Sets the size of a VM disk in megabytes using the syntax **--diskSize:<VMID>,<diskInstanceID>=<value>**. Only applies to vCloud targets of version 5.5 or later.
--eula	N/A	OVF, OVA	Inserts the EULA in the first virtual system or virtual system collection in the OVF. If the EULA is in a file, use this format: **--eula@=filename**
--exportFlags			Specifies the source of an export. The supported values for vSphere source are: mac, uuid, and extraconfig. The supported value for vCloud source is preserveIdentity. You can provide one or more options, separated by commas.
--extraConfig	N/A	All	Sets an ExtraConfig element for all VirtualHardwareSections. This option applies to vi, vmx, vapprun, vCloud, ovf, and ova source locators. The syntax is **-:extraConfig:<key>=<value>**

Table continued on next page

Continued from previous page

Option's Long Name (and Short Name)	Source Types	Target Types	Description
--fencedMode	vCloud		If a parent network exists on a vCloud target, this option specifies the connectivity to the parent. Possible values are bridged, isolated, allowIn, allowInOut, allowOut.
--help or **-h**	N/A	N/A	Prints the VMware OVF Tool message that lists the help options.
--hideEula	OVF,OVA	N/A	Does not include the EULA in the OVF probe output. Binary option.
--I:morefArgs	vSphere	vSphere	Integration option. Interpret arguments for networks, datastores, and folders as VIM Managed Object Reference identifiers (type:id) for vSphere source and destination locators.
--I:sourceSessionTicket	vSphere	vSphere	Integration option. Specifies the session ticket used for authenticating the vSphere source locator.
--I:targetSessionTicket	vSphere	vSphere	Integration option. Specifies the session ticket used for authenticating the vSphere target locator.
--importAsTemplate			Imports VM as a template when deployed on a VI target.
--ipAllocationPolicy	OVF,OVA	N/A	IP allocation policy for a deployed OVF package. Supported values are: dhcpPolicy, transientPolicy, fixedPolicy, or fixedAllocatedPolicy. In OVF descriptors, you can specify a VMware specific IP assignment policy that guides the deployment process by expressing which of the policies the OVF package supports. Only values listed in the OVF descriptor are supported when the OVF or OVA package is deployed.
--ipProtocol	OVF, OVA	N/A	Specifies which IP protocol to use. For example, **IPv4**, **IPv6**. As with the **ipAllocationPolicy** option, you can specify which IP version this OVF package uses when it is deployed. Use only the values listed in the OVF descriptor.
--lax	OVF, OVA	N/A	Relax OVF specification conformance and virtual hardware compliance checks. (For advanced users only.)
--locale	OVF, OVA	N/A	Selects the locale for the target.
--machineOutput	N/A	N/A	Outputs OVF Tool messages in a machine readable format. Binary option.
--makeDeltaDisks	OVF, vSphere, VMX, vApprun	Must be directory	Use delta disk compression to create an OVF package from a disk source. Binary option.
--maxVirtualHardwareVersion			The maximal virtual hardware version to generate.

Table continued on next page

Continued from previous page

Option's Long Name (and Short Name)	Source Types	Target Types	Description
--memorySize			Sets the memory size in megabytes of a VM using the syntax **--memorySize:<VMID>=<value>.** Example: **--memorySize:vm1=1024**. Applies to VI and vCloud targets of version 5.5 or later.
--multiDatastore:dsk=ds	OVF, OVA	vSphere	Allows multi-disk VM deployment onto separate datastores. Use this option as many times as needed The VM disk is placed on datastore **ds** as follows, where **vmname** is the name specified by **--name** option or in the OVF file, and **vmdisk** is specified in the OVF disk section by **ovf:diskId**. `[ds] vmname/vmdisk1.vmdk`
--name or **-n**	N/A	All	Specifies the target name. Defaults to the source name.
--net	OVF, OVA	N/A	Sets a network assignment in the deployed OVF package. For example, **--net:<OVFname>=<target name>**. OVF packages contain symbolic names for network names which are assigned with this option. For multiple network mappings, repeat the option, separating them with a space, for example, **--net:s1=t1 --net:s2=t2 --net:s3=t3**. If the target is vCloud 5.5 or later, a fence mode can also be specified using the syntax **--net:<OVFname>=<targetName>,<fenceMode>**. Possible fence mode values are: **bridged**, **isolated**, and **natRouted**.
--network or **-nw**	OVF, OVA	N/A	Target network for a vSphere deployment. Use this option in place of the **--net** option when only one network exists in the OVF package. This option maps the symbolic OVF name to the specified network name.
--nic			Specifies NIC configuration in a VM using the syntax **--nic:<VMID>,<index>=<OVFnetName>,<isPrimary>,<ipAddressingMode>,<ipAddress>**. Possible values for ipAddressingMode are: **DHCP, POOL, MANUAL,** and **NONE**. ipAddress is optional and should only be used when **ipAddressingMode** is set to **MANUAL**. Only applies to vCloud targets of version 5.5 or later.
--noDestinationSSLVerify			Skips SSL verification for target VI connections.

Table continued on next page

Continued from previous page

Option's Long Name (and Short Name)	Source Types	Target Types	Description
--noDisks	N/A	All	Creates and uploads the virtual machine or vApps but does not upload any disk files. Disks are created empty. (Disables disk conversion.)
--noImageFiles	N/A	All	Creates and uploads the virtual machine or vApps but does not upload ISO files to a CD-ROM. That is, does not include image files in destination.
--noNvramFile	vSphere	vSphere	Skips the NVRAM file when exporting a VM to OVF or deploying OVF to a VM. It does not download the NVRAM file during VM export, nor does it upload the NVRAM file during deployment or OVF import.
--noProxyVerify	vCloud	vCloud	Do not check the proxy's SSL certificate, as is done by default.
--noSSLVerify	vSphere	vSphere	Skip SSL verification for vSphere connections. As of OVF Tool 4.4.0 you can skip source or destination verification with **--noSourceSSLVerify** (unless you also specify **--sourceSSLThumbprint**) or **--noDestinationSSLVerify**.
--numberOfCpus			Sets the number of CPUs for a VM using the syntax **--numberOfCpus:<VMID>=<value>**. Applies to VI and vCloud targets of version 5.5 or later.
--overwrite or **-o**	N/A	All	Forces an overwrite of existing files. Binary option.
--packageCert	OVF	OVA	Packages an OVF into OVA with a required signature file, as-is, instead of generating one. This is the only way to create an OVA with a certificate file in it, resulting in a signed package.
--parallelThreads=N	vSphere	vSphere	Upload to the host, or from the host, in parallel using N threads. VMX<=>OVF, OVF<=>VI, VMX<=>VI, but not OVA. N may be a number from 1 to 100 but should approximate the number of CPU cores minus one.
--powerOffSource	vCloud, vSphere	N/A	Ensures that a virtual machine or vApp is powered off before importing from a vSphere source. Binary option.
--powerOffTarget	N/A	vCloud, vSphere	Ensures that a virtual machine or vApp is powered off before overwriting a vSphere target. Binary option.
--powerOn	N/A	vCloud, vSphere	Powers on a virtual machine or vApp deployed on a vSphere target. Binary option.
--preCheck			Require pre check validations before import or export.
--privateKey	N/A	OVF, OVA	Signs the OVF package with the given private key (`.pem` file). The file must contain a private key and a certificate.
--privateKeyPassword	N/A	OVF, OVA	Password for the private key. Used in conjunction with **--privateKey** if the private key requires password authentication. If required but not specified, the tool prompts for the password.

Table continued on next page

Continued from previous page

Option's Long Name (and Short Name)	Source Types	Target Types	Description
--prop	OVF, OVA	N/A	Sets a property in the deployed OVF package. For example, **--prop:<key>=<value>**. Use probe mode to learn which properties an OVF package can set. For multiple property mappings, repeat the option, separating them with a blank, for example, **--prop:p1=v1 --prop:p2=v2 --prop:p3=v3**.
--proxy	OVF, OVA, vCloud, vSphere	OVF, OVA, vCloud, vSphere	Specifies and enables a proxy for HTTP, HTTPS, FTP, vSphere, and vCloud access. The proxy is expressed as the URL to the proxy. For example, for `proxy.example.com`, the option value is: https://proxy.example.com:345 OVF Tool supports proxies that require authentication. If you do not provide credentials in the URL, the OVF Tool prompts you for them. The **--proxy** flag enables both vSphere (VI) and vCloud proxies.
--proxyCert=vcdCert	vCloud	vCloud	Specifies the certification file for vCloud.
--proxyNTLMAuth	OVF, OVA, vCloud, vSphere	OVF, OVA, vCloud, vSphere	Enables support for the NTLM authentication and security protocol. NT LAN Manager is the authentication protocol used on networks that include systems running the Windows operating system and on stand-alone systems.
--proxyPassword			Proxy password.
--proxyUsername			Name of proxy user.
--pullUploadMode	HTTP, HTTPS	vSphere	Offers fast deployments of OVF packages by enabling a mode where OVF source files (VMDKs etc.) are copied directly from source to ESXi without **ovftool** in between. See Deploying an OVF Package using Pull Mode for details.
--quiet or **-q**	N/A	N/A	Prints only errors. No other output is sent to the screen. Binary option.
--requireSignature	OVF, OVA	OVF, OVA	A certificate file (signature) must be present for the package when reading (probing) or opening a package.
--schemaValidate	OVF, OVA	N/A	Validates OVF descriptor against the OVF schema. Binary option.
--shaAlgorithm	sha1, sha256, sha512		Use this option to condense with Secure Hash Algorithm (SHA) for manifest validation, digital signing, and OVF package creation. Can be either **sha1** (SHA-1), **sha256** (SHA-256), or **sha512** (SHA-512). The default value is sha256.

Table continued on next page

Continued from previous page

Option's Long Name (and Short Name)	Source Types	Target Types	Description
--skipManifestCheck	OVF, OVA	N/A	Skips validation of the OVF package manifest. Binary option.
--skipManifestGeneration	N/A	OVF, OVA	Skips generation of the OVF package manifest. Binary option.
--sourcePEM			File path to a Privacy Enhanced Mail (.pem) file used to verify vSphere connections. Example: **--sourcePEM:<filename>.pem**
--sourceSSLThumbprint	vSphere	N/A	SSL thumbprint of the source. OVF Tool verifies the SSL thumbprint that it receives from the source, if this value is set.
--sourceType or **-st**	OVF, OVA, VMX, VMX, VI, vCloud, ISO, FLP, vApprun	N/A	Explicitly expresses that the source is **OVF, OVA, VMX, VMX, vSphere, vCloud, ISO, FLP,** or **vApprun**.
--sslCipherList			Allows override of the default cipher list. Diffie-Hellman is disabled by default.
--sslVersion			Either **TLSv1_0**, **TLSv1_1**, or **TLSv1_2**. The **TLSv1_0** protocol is disabled by default.
--storageProfile			Sets the storage profile for a VM using the syntax **--storageProfile:<VMID>=<value>**. Applies only to vCloud targets of version 5.5 or later.
--targetPEM			File path to a Privacy Enhanced Mail (.pem) file used to verify vSphere connections. Example: **--targetPEM:<filename>.pem**
--targetSSLThumbprint	N/A	vSphere	SSL thumbprint of the target. OVF Tool verifies the SSL thumbprint that it receives from the target, if this value is set.
--targetType or **-tt**	N/A	OVF, OVA, VMX, VI, vCloud, ISO, FLP, vApprun	Explicitly express that the target is OVF, OVA, VMX, VMX, vSphere, vCloud, ISO, FLP, or vApprun.
--vCloudTemplate			Create only a vAppTemplate.
--vService	OVF, OVA	N/A	Set a dependency on a vService provider in the OVF package, using the following syntax: **--vService:<dependencyId>=<providerId>**
--verifyOnly	All	N/A	Do not upload the source; just verify it. This applies only to vSphere 4.
--verifyViTargetManifest	vSphere, VI	vSphere, VI	Upon import of OVF, does extra check and validates the SHA1 digest of a file transferred to a VI Target against the SHA1 digest of the source file being transferred. SHA1 takes effect for all SHA calculations, so the host must support SHA1. Not compatible with **--pullUploadMode**.

Table continued on next page

Continued from previous page

Option's Long Name (and Short Name)	Source Types	Target Types	Description
--version or **-v**	N/A	N/A	Shows version information for OVF Tool. Binary option.
--viCpuResource	N/A	vSphere	Specify the CPU resource settings for VI locator targets. The syntax is: **--viCpuResource=<shares>:<reservation>:<limit>**
--viMemoryResource	N/A	vSphere	Specify the memory resource settings for vSphere locator targets. The syntax is: **--viMemoryResource=<shares>:<reservation>:<limit>**
--vmFolder or **-vf**	N/A	vSphere	The target virtual machine folder in vSphere inventory (for a datacenter).
--X:ignoreLinkLocalIp	vSphere	vSphere	While waiting for an IPv4 or IPv6 address,ignore a link local address, 169.254.0.0/16 or fe80::/10.
--X:skipContentLength	N/A	N/A	Avoids sending the Content-Length header in HTTP(S) requests. This is the default option for vCloud targets.
--X:vmxFromConfigFile	vSphere	vSphere	Extract ConfigInfo from a live VM, convert it to VMX, and save results in the specified file. You can also specify a file to log this transaction. For example: **--X:vmxFromConfigFile=vmxConfigInfoAsVmx** **--X:logFile=cfgInfo.log**
--X:waitForIpv4	vSphere	vSphere	Have OVF Tool wait for an IPv4 address to be sure the network is initialized.
--X:waitForIpv6	vSphere	vSphere	Have OVF Tool wait for an IPv6 address to be sure the network is initialized.

Run **ovftool-hdebug** for explanations of the available **--X** options.

Creating and Using the VM ID

When the parameters for one of the command line options includes the *VM ID*, this ID refers to an attribute in the OVF descriptor file. Specifically, it is the ID attribute of the VirtualSystem element that will appear in the OVF file that describes the VM you want to create or customize. If you are creating a VM, you need to specify the ID in the descriptor file. For example, the ID of the VM specified in the descriptor fragment below is *vm1*.

```
<ovf:VirtualSystem ovf:id="vm1">
   <ovf:Info>A virtual machine</ovf:Info>
   <ovf:Name>WinServer2012</ovf:Name>
   <ovf:OperatingSystemSection ovf:id="74" vmw:osType="windows8Server64Guest">
```

```
      <ovf:Info>Specifies the operating system installed</ovf:Info>
      <ovf:Description>Microsoft Windows Server 2012 (64-bit)</ovf:Description>
   </ovf:OperatingSystemSection>
   ...
```

For example, you must use the VM ID when specifying the size of memory for a VM, as in this option: **`--memorySize:vm1=1024`**

If you are customizing an existing VM, look at the descriptor file to get the VM ID. You can also have the **`ovftool`** read an OVF file and extract the IDs before importing or deploying it.

```
> ovftool --verifyOnly --machineOutput <src ovf>
```

Specifying Disk ID to Set Size

When specifying disk sizes, you will need to specify the instance ID as well as the VM ID. The instance ID is the value of RASD InstanceID element of the virtual hardware section element describing the disk that should be resized.

```
<ovf:DiskSection>
   <ovf:Info>Virtual disk information</ovf:Info>
   <ovf:Disk ovf:capacity="4" ovf:capacityAllocationUnits="byte * 2^20"
      ovf:diskId="disk1" ovf:fileRef="disk1-file"
      ovf:format="http://www.vmware.com/interfaces/specifications/
vmdk.html#streamOptimized"/>
</ovf:DiskSection>
<ovf:VirtualSystem ovf:id="vm1">
   ...
   <ovf:VirtualHardwareSection>
   <ovf:Info>Virtual hardware requirements</ovf:Info>
   <ovf:Item>
      <rasd:AddressOnParent>0</rasd:AddressOnParent>
      <rasd:Description>SCSI Hard disk</rasd:Description>
      <rasd:ElementName>SCSI Hard disk 1</rasd:ElementName>
      <rasd:HostResource>ovf:/disk/disk1</rasd:HostResource>
      <rasd:InstanceID>2000</rasd:InstanceID>
      <rasd:Parent>2</rasd:Parent>
      <rasd:ResourceType>17</rasd:ResourceType>
   </ovf:Item>
   ...
```

In the above example specifying instance ID “2000” (without quotes) would cause the disk with ID “disk1” (without quotes) to be resized: --diskSize:vm1,2000=256 (set the size to 256).

Note that if multiple disk devices are backed by the same disk (i.e. the OVF contains multiple disk RASD items that refer to the same disk) you must specify the new size for all disk elements, not just one. Sharing disks between VMs is not common, but allowed in the OVF spec.

Note that you cannot shrink disks.

Specifying the Storage Profile ID

When you specify a storage profile, the ID refers to the UID of the storage profile in the vCD cell. This can be obtained using the vCD REST API.

More Help Topics

For more help, type: **`--help <topic>`**, where topics are:

locators	: detailed source and destination locator syntax
debug	: lists the debug settings
examples	: examples of usage
config	: syntax of configuration files
integration	: list of options primarily used when you execute the **`ovftool`** from another tool or shell script

Using the Log Settings

Use OVF Tool’s log options if you are not seeing the results you expect. The log options allow you to see the operations of OVF Tool, and send results to the console or to a file.

Two of the most commonly used log options are:

- **`--X:logFile=`** *`filename`* to log the complete **`ovftool`** session to a file.
- **`--X:logLevel=`** *`level`* to control the verbosity of logs.

For example, you can use a command like this to write the log in a file called `ovf.log`:

```
ovftool --X:logFile=ovf.log --X:logLevel=info LAMP.ovf vi://localhost/Datacenter/host/
Cluster
```

The following table lists all the log options.

Log Option	Use option to:
`--X:logFile=` *`filename`*	Log internal events to a specified log file.
`--X:logLevel=` *`level`*	Log level. Valid values are: **`none`**, **`quiet`**, **`panic`**, **`error`**, **`warning`**, **`info`**, **`verbose`**, and **`trivia`**).
`--X:logToConsole`	Log internal events to console
`--X:logTransferHeaderData`	Add transfer header data to the log. Use with care. Default value is **`false`**

OVF Tool includes many other debug options that you can set to retrieve specific data. You can see all the **`-X`** options and their definitions by running **`ovftool--helpdebug`**.

OVF Tool does not log passwords.

Specifying a Locator

A source or target locator points to a specific resource. Locators must specify a protocol, which defines how to reach the resource. Supported protocols are file access, vSphere, HTTP, HTTPS, and FTP.

File locators can point to an OVF package (`.ovf` or `.ova`), a virtual machine (`.vmx`). HTTP, HTTPS, or a vApprun workspace entity. FTP locators can point to OVF and OVA files. The resource type is determined from the filename suffix, unless one or both of the options **`--sourceType`** and **`--targetType`** are used explicitly.

vSphere locators can point to various resource types: virtual machines, vApps, hosts, clusters, or resource pools. For a source locator, the resource type must be a virtual machine or vApp. For a target locator, the resource type must be a host, cluster, or a resource pool. A vSphere locator is used for a vSphere server, vCenter Server, VMware Server, or an ESXi host.

At the command line, type `--help locators` to display the online help for locators.

Table : Source Locator and Table : Target Locator list the default extensions of the different source and target types, as well as which protocols are supported.

Table 123: Source Locator

Source Type	Default File Extension	Protocol	Example
OVF	`.ovf`	File, HTTP, HTTPS, FTP	/ovfs/my_vapp.ovf
OVA	`.ova`	File, HTTP, HTTPS, FTP	/ovfs/my_vapp.ova
VMX	`.vmx`	File	/vms/my_vm.vmx
vApprun	N/A	File	~/my_vApprun_workspace/MyVM
vCloud Director	N/A	HTTPS	vcloud://username:password@mycloud.org/ \ org=MyOrg&vdc=MyVDC&catalog=MyCatalog \ &vapp=myVapp
vSphere	N/A	vSphere	vi://username:pass@localhost/my_datacenter/vm/ \ my_vms_folder/my_vm_name

Table 124: Target Locator

Target Type	File Extension	Protocol	Example
OVF	`.ovf`	File	/ovfs/my_vapp.ovf
OVA	`.ova`	File	/ovfs/my_vapp.ova
VMX	`.vmx`	File (Source must be a single virtual machine)	/vms/my_vm.vmx
vApprun	N/A	File	~/my_vApprun_workspace/MyVM
vCloud Director	N/A	HTTPS	vcloud://username:password@mycloud.org/ \ org=MyOrg&vdc=MyVDC&catalog=MyCatalog \ &vapp=myVapp
vSphere	N/A	vSphere (If the vSphere target locator is on a VMware Server system, or directly on	vi://username:pass@localhost/my_datacenter/vm/ \ my_vms_folder/my_vm_name

Table continued on next page

Continued from previous page

Target Type	File Extension	Protocol	Example
		an ESXi host, the source must be a single virtual machine)	

File Locators

File locators are the same for source and target. They are specified using ordinary path syntax.

Windows Path Syntax

On Windows, paths are specified as either absolute or relative.

This is an example of an absolute path on Windows:

```
C:\folder1\folder2\package.ovf
```

These examples show relative paths on Windows:

```
..\folder1\package1.ovf
```

```
package1.ovf
```

Linux and Mac OS Path Syntax

On Linux, paths are specified, similarly, as either absolute or relative.

The following is an example of an absolute path on Linux:

```
/folder1/folder2/package.ovf
```

The following are examples of relative paths on Linux:

```
../folder1/package1.ovf
```

```
package1.ovf
```

URL File or Web Syntax

It is possible to specify files or websites as a URL by prefixing the path with `file://` or `http://`, as shown in the following examples.

Examples of file locations:

```
file://c:\folder1\folder2\package.ovf (Absolute, Windows)
```

```
file:///folder1/folder2/package.ovf (Absolute, Linux)
```

```
file://package.ovf (Relative for both Windows and Linux)
```

Encoding Special Characters in a URL

When you specify a URL, special characters must be escaped with percent % followed by the respective ASCII hex value. For instance, if you use at-sign @ in a URL, it must be specified as %40, and backslash in a Windows domain name (\) must be specified as %5c.

When you specify a login name and password in a URL, or interactively, **ovftool** uses an open source library for URL encoding, also called percent encoding, to make the result acceptable on the web. For VI and vCloud locations, percent encoding is unnecessary and not done.

HTTP, HTTPS, and FTP Locators

You can use HTTP, HTTPS, and FTP to refer to an OVF package (OVF or OVA file) on a Web server. You can only use these protocols to specify a source locator. In the following syntax, **protocol** is HTTP, HTTPS or FTP:

```
protocol://username:password@host:port/<path to OVF package>
```

It is possible to omit the user name and password from the locator. If needed, OVF Tool prompts you for them. If you use the standard port, it is not necessary to specify the port. Table : Standard Ports shows the standard ports.

Table 125: Standard Ports

Protocol	Port
HTTP	80
HTTPS	443
FTP	21

vSphere Locators

vSphere source locators point to a virtual machine or vApp within the virtual infrastructure. The vSphere target locator provides all required information for importing an OVF package or virtual machine into a cluster, host or resource pool. Both source and target locator use the same syntax:

```
vi://<username>:<password>@<host>:<port>/<search-term>
```

The server name and port can designate either a vCenter server or an ESXi host. If you omit credentials, OVF Tool prompts you for them. Default installations of vCenter Server and ESXi use port 443. If you are using the default port, you do not need to specify it. When using OVF Tool with VMware Server, you had to specify port 8333, the default port for this discontinued product.

The search term has the following format:

```
<path>[?<query>=<value>]
```

If a query is not given, a VC inventory path lookup is performed using the specified path. Otherwise, the object matching the query is used. The meaning of the query depends on the object type. Table : Source and Target Values for All Query Types shows the different values that you can use in the query field.

Table 126: Source and Target Values for All Query Types

Name	Query	Source	Target
BIOS	**bios**	BIOS ID of a virtual machine	BIOS ID of a host
Datastore	**ds**	Datastore path to a virtual machine	N/A
IP Address	**ip**	IP address of a virtual machine	IP address of a host
DNS	**dns**	DNS name of a virtual machine	DNS name of a host
Mo-Ref	**moref**	Managed object reference (vSphere specific identifier) of a virtual machine or vApp	Managed object reference (vSphere specific identifier) of a host, cluster, or resource pool

Table : Examples of Query Values shows example values for each query type.

Table 127: Examples of Query Values

Name	Query	Example Value
BIOS	**bios**	vi://localhost?bios=234290984
Datastore	**ds**	vi://localhost/TestDatacenter?ds=[foo]/myvm/myvm.vmx
IP Address	**ip**	vi://localhost?ip=123.231.232.232
DNS	**dns**	vi://localhost?dns=production-vm3.example.com
Mo-Ref	**moref**	vi://localhost?moref=vim.VirtualMachine:vm-23423
Note: For the same VM, the Managed Object Reference (MoRef) will be different for vCenter Server than it is for the ESXi host. For instance, the same VM can have a MoRef of 'vm-92' in vCenter Server and '118' in the ESXi host. Also note that the syntax contains 'vm-' for a vCenter Server MoRef. When the Motif is defined by vCenter Server, you can see it in the browser address and use it directly. The OVF Tool script doesn't define and parse the format of a MoRef.		

You can enter a partial source locator if you do not know the entire inventory path. In this case, the tool fails but suggests possible inventory path completions.

Specifying the Inventory Path to a Virtual Machine or vApp

To specify an inventory path for a virtual machine or vApp, use the following syntax:

```
<datacenter name>/vm/<folders>/<vm or vApp name>
```

or

```
<datacenter name>/host/<resource pool path>/<vm or vApp name>
```

The use of the **vm** tag after the datacenter name specifies that you are locating a virtual machine or vApp in the VM and Template view. Use the host tag after the datacenter name if you are locating a virtual machine or vApp in the Host and Clusters view.

The following example shows an inventory path without any folders:

```
MyDatacenter/vm/MyVM
```

The following example shows an inventory path with two nested folders:

```
MyDatacenter/vm/Folder 1/Sub Folder/MyVM
```

Specifying the Inventory Path for a Cluster, Host, or Resource Pool

You can specify an inventory path for a host or a resource pool. You can nest resource pools similar to folders. To specify an inventory path for a host or a resource pool as part of target locators, use the following syntax:

```
<datacenter name>/host/<host name>/Resources/<resource pool>
```

- **host** and **Resources**. Fixed parts of the path.
- **Resources**. Specify only when a resource pool is specified.
- **<resource pool>**. Can take the value of one or more nested resource pools. If no resource pools are specified, the default resource pool for the host is used.

The following example is of an inventory path without a specified resource pool:

```
vi://username:pass@localhost/my_datacenter/host/esx01.example.com
```

The following example is of an inventory path with a specified resource pool:

```
vi://username:pass@localhost/my_datacenter/host/esx01.example.com/Resources/
my_resourcepool
```

NOTE
You must specify the /host/ section of an inventory path when using a vi destination locator. If you are specifying the destination of a resource pool, you must include the /Resources/ section of the path.

vCloud Director Locators

The syntax for vCloud locators are the same as for other locators:

```
vcloud://username:password@host:port?org=name_of_org&vapp=name_of_deployed_vapp&
catalog=name_of_catalog&vappTemplate=name_of_vapp_template_in_catalog&vdc=name_of_vdc
```

Some of the options are not needed if there is only one virtual datacenter to choose from. If there are more than on datacenter, the catalog option is required. The **`org`** option is mandatory, because it is used to log in to vCloud Director.

NOTE
OVF Tool supports all source types for vCloud Director 1.5. For vCloud Director 1.0, OVF Tool only supports OVF/OVA/vCloud sources. OVF Tool does not support vi, vmx, or vapprun sources for vCloud Director 1.0.

Examples of vCloud Locators

The following example uploads and deploys an OVF named test into vCloud Director and names the vApp my_test1.

```
ovftool /tmp/test.ovf vcloud://user1:password@10.99.99.99:7443?org=o&vapp=my_test1
```

This example exports a vCloud Director vApp to the OVF file /tmp/test1.ovf

```
ovftool vcloud://user1:password@10.10.99.99.99:7443?org=o&vapp=my_test1 /tmp/test1.ovf
```

If you use a network, you map the network in the usual way:

```
--net:sourceNET=targetNET
```

You also apply properties in the usual way.

Partial Locators

When using OVF Tool, it is often not necessary to specify source and target types as long as certain filename conventions are used. It is possible to the ignore locator type and specify the source and target explicitly using the arguments `--sourceType=`... and `--targetType=`.

OVF Tool assumes the locator type based on the following rules:

- If the name starts with `vcloud://`, OVF Tool assumes vCloud Director type.
- If the name starts with `vi://`, OVF Tool assumes vSphere type.
- If the name ends with `.ovf`, OVF Tool assumes OVF type.
- If the name ends with `.vmx`, OVF Tool assumes VMX type.
- If the name ends with `.ova`, the OVF tool assumes OVA type.
- If the locator is a file path to a directory that represents a vApprun workspace or an entity in a vApprun workspace, then OVF Tool assumes vApprun type.

Similarly, source and target types can be inferred from folder locators. OVF Tool assumes the type using the following rules:

- If the source locator is a folder, OVF Tool assumes that the source is an OVF package and that the OVF descriptor is called the same as the folder, for example, `my-ovf/my-ovf.ovf`.
- If the source is an OVF package and the target locator is a directory, such as `MyVirtualMachines/`, OVF Tool assumes that the target is a VMX locator. The created VMX/VMDK file is put in a directory with the target name, for example, `MyVirtualMachines/MyVM/MyVM.vmx`.
- If the source is a VMX locator and the target locator is a directory, OVF Tool assumes that the target is an OVF package.
- If the source is a vSphere locator, and the target locator is a directory, OVF Tool assumes that the target is an OVF package.

OVF Tool supports partial vSphere locators when deploying or exporting. For an incomplete locator path, the tool suggests completions at the command line. Partial vSphere Locators at the Command Line shows the command-line dialog when partial locators are used.

Partial vSphere Locators at the Command Line

```
> ovftool LAMP.ovf vi://localhost/

Opening source: LAMP.ovf

Opening target: vi://user@localhost/

Error: Found wrong kind of object (Folder)

Possible completions are:

  Datacenter/

  Remote Datacenter/

  Secondary Datacenter/


> ovftool LAMP.ovf vi://localhost/Datacenter

Opening source: LAMP.ovf

Opening target: vi://user@localhost/Datacenter

Error: Found wrong kind of object (Datacenter)

Possible completions are:

  vm/

  host/


> ovftool LAMP.ovf vi://localhost/Datacenter/host

Opening source: LAMP.ovf

Opening target: vi://user@localhost/Datacenter/host

Error: Found wrong kind of object (Folder)

Possible completions are:

  host1.foo.com/
```

```
  host2.foo.com/


> ovftool LAMP.ovf vi://localhost/Datacenter/vm/host1.foo.com
```

OVF Tool supports partial vSphere locators when deploying or exporting. For an incomplete locator path, the tool suggests completions at the command line. Partial vCloud DirectorLocators at the Command Line shows the command-line dialog when partial locators are used. First, OVF Tool signals that there is more than one virtual datacenter present, then multiple catalogs, then multiple networks. At each attempt, you must select one of the options that OVF Tool presents.

Partial vCloud DirectorLocators at the Command Line

```
ovftool LAMP.ovf vcloud://jd:PASSWORD@example.com:443/?org=myOrg&vapp=test1

Opening OVF source: LAMP.ovf

Warning: No manifest file

Opening vCloud target: vcloud://js:PASSWORD@example.com:443/

Error: Multiple VDCs found. Possible VDC completions are:

  orgVdc

  orgVdc2

Completed with errors


ovftool LAMP.ovf "vcloud://jd:PASSWORD@example.com:443/?org=myOrg&vapp=test1&vdc=orgVdc"

Opening OVF source: LAMP.ovf

Warning: No manifest file

Opening vCloud target: vcloud://js:PASSWORD@example.com:443/

Error: Multiple catalogs found. Possible catalog completions are:

  catalog

  catalog2

Completed with errors


"vcloud://jd:PASSWORD@example.com:443/?org=myOrg&vapp=test1&vdc=orgVdc&catalog=catalog"

Opening OVF source: LAMP.ovf

Warning: No manifest file

Opening vCloud target: vcloud://js:PASSWORD@example.com:443/

Error: Multiple networks found on target. Possible completions are:

  extNet2

  extOrgNet
```

```
  intNet2
  intnet
Completed with errors


ovftool --net:"VM Network=intnet" LAMP.ovf "vcloud://jd:PASSWORD@example.com:443/
?org=myOrg&vapp=test1&vdc=orgVdc&catalog=catalog"
Opening OVF source: LAMP.ovf
Warning: No manifest file
Opening vCloud target: vcloud://js:PASSWORD@example.com:443/
Deploying to vCloud: vcloud://js:PASSWORD@example.com:443/
Disk Transfer Complete
Completed successfully
```

Configuration Files

OVF Tool has many options. Rather than repeatedly entering long commands on the command line, you can create a configuration file. A configuration file uses the following syntax:

```
option1=value
...
#comment
optionN=value
```

The following is an example of a configuration file:

```
proxy=http://proxy.example.com
datastore=storage-test42
# Comment on something
locale=dk
```

You can create local or global configuration files. The local configuration file **`ovftool.cfg`** is read in the folder where you invoke OVF Tool. The global configuration file is per user.

On Windows (after XP), the global configuration file is in the following location:

```
C:\Users\%USER%\AppData\Roaming\VMware\ovftool.cfg
```

Linux and OS X look for the global configuration file in the following location:

```
$HOME/ovftool.cfg (example ~/ovftool.cfg)
```

When reading configuration files, globally defined options are overwritten by locally defined options, and locally defined options are overwritten by the `--configFile` command-line option. Some variables, such as proxy, do not allow you to

specify the option in multiple places. If **ovftool** stops running with the error “already exists“ it is likely that you specified an option on the command line and in a configuration file.

You can use the `ovftool --help config` command to get information about how to use a configuration file. In addition, the current contents of the global configuration file as well as any local configuration file is shown.

Handling Authentication

OVF Tool generates AUTHENTICATION output messages if access to a resource requires username and password. A vSphere or vCloud locator, a proxy server, or an authenticated URL require usernames and passwords. OVF Tool generates AUTHENTICATION messages only for resources where passwords are not explicitly provided as command-line arguments or as part of the locator.

OVF Tool can authenticate the following types of objects:

- source locators
- target locators
- proxyServer

For source and target locators, you must provide the username on the command-line. If you do not provide a password, OVF Tool generates an AUTHENTICATION message and you must provide the password on STDIN. If the proxy server requires authentication, you must provide both the username and password on STDIN. OVF Tool supports the following commands on STDIN:

```
For the source password:

   PASSWORDSOURCE

   password

For the target password:

   PASSWORDTARGET

   password

For the proxy:

   PASSWORDPROXY

   username password
```

For an example of the output of running **machineOutput** in authentication mode, see Output from Running machineOutput in Import Mode.

Launching the OVF Tool as a Helper Process

You can use the integration options to make it more convenient to launch OVF Tool as a helper process to a client of the vSphere Web Services API, such as a script using Perl bindings.

If you use the **--I:morefArgs** argument, the values for **--vmFolder**, **--network**, **--net**, and **--datastore** are interpreted as MoRefs instead of names, as shown in the following example:

```
> ovftool --name=vm5 \
                --I:morefArgs \
                --net:VM Network=vim.Network:network-12 \
                --datastore=vim.Datastore:datastore-17 \
```

```
c:\temp\vm1\ \

vi://root:@localhost?moref=vim.ResourcePool:resgroup-42
```

Use the **`--I:sourceSessionTicket`** or **`--I:targetSessionTicket`** options to authenticate with a session ticket retrieved from **`SessionManager.AcquireSessionTicket`**, when using the vSphere source or destination.

Examples of OVF Tool Syntax

This chapter provides many examples of OVF Tool usage, that are divided into the following categories:

You can see similar examples within the OVF Tool, by typing `--help examples` on the command line while you are in the directory where the **`ovftool`** script is running.

Supported File Types and Package Formats

As discussed in earlier chapters, you can use the OVF Tool to package virtual machines as vApps (ready-to-use virtual machine groups with cooperating systems and applications). The package formats supported by the OVF Tool can be read and/or imported by other VMware and third-party software.

The table below describes each of the supported formats:

Table 128: Supported File and Package Types for OVF Tool Input and Output

Package Type	Full Name	Usage
OVF (.ovf)	Open Virtualization Format	National ANSI standard for packaging software for virtual machines, originally created by an industry consortium, the Distributed Management Task Force (DTMF). An OVF package includes: a descriptor file, optional manifest and certificate files, optional disk images, and optional resource files (such as ISOs). The disk image files can be in VMware's `.vmdk` disk image format or any other supported disk image format. OVF packages can be used by the software of any hypervisor or processor architecture that supports this format.
OVA (.ova)	Open Virtual Appliance	A TAR archive that contains an OVF package.
VMX (.vmx)	Virtual Machine Configuration File	When you create a new virtual machine, this file is created to store information about the operating system, disk sizes, networking, and virtual hardware. Files in this format and in `.vmdk` format are sometimes referred to together as VMware runtime format.
VMDK (.vmdk)	Virtual Machine Disk	Files with this extension may contain disk characteristics (,vmdk), contents (-flat.vmdk), or snapshot files (-delta.vmdk). These files are called out on the OVF Tool command line, but may exist within the package.
VI (vi://)	VMware Infrastructure	This is an older term that originated with ESX 3, but is still seen in the command line syntax for the OVF Tool. As an OVF command line option, 'vi//' is used before the credentials and path to a server.
vCloud	vCloud Director format	The vCloud Director REST API makes basic transfer between clouds possible using OVF packages, which preserve application properties, networking configuration and other settings.

Table continued on next page

Continued from previous page

Package Type	Full Name	Usage
ISO (.iso)	Optical Image File	An ISO archive is a CD/DVD image. Creating a package as an ISO image allows you to install a virtual appliance using a CD ROM drive. This type of archive is called an ISO because it was created by the International Standards Organization's 9660 standard.
FLP (.flp)	Floppy Disk Image File	Use this format if you need to transfer data from a floppy drive or to the virtual machine floppy drive. See the VMware Knowledge Base for FLP information.
vApprun	vApprun	This format allows you to run a virtual appliance on VMware Fusion or Workstation. You can use OVF Tool to convert vApps to the vApprun format, and you can use VMware Workstation to convert vApps to OVF format.

Use the OVF Tool with the Target Type option to specify the target out as OVF, OVA, VMX, VI, vCloud, ISO, FLP, vApprun.

In this following example, the target type is set to the 'vmx' or VMware runtime format (.vmx and .vmdk files)

```
> ovftool -tt=vmx /ovfs/my_vapp.ovf /vms/
```

The resulting files are: `/vms/my_vapp/my_vapp.vmx` and `/vms/my_vapp/my_vapp.vmdk`, the typical VMware runtime format for a virtual machine folder.

Changing File or Package Formats

The examples below show you how to convert from an existing format to a different format.

Converting an OVF Package to an OVA Archive

To convert an OVF package, to a single OVA archive, use the following syntax:

```
> ovftool /ovfs/my_vapp.ovf /ovfs/my_vapp.ova
```

Converting an OVA Archive to OVF Package

To extract an OVA archive to an OVF package, use the following syntax:

```
> ovftool /ovfs/my_vapp.ova /ovfs/my_vapp.ovf
```

Converting VMX Format to an OVF Package

To convert a virtual machine in VMware runtime format (.vmx and .vmdk files) to an OVF package, use the following syntax:

```
> ovftool /vms/my_vm.vmx /ovfs/my_vapp.ovf
```

The result is located in `/ovfs/my_vapp.ovf`

Converting VMX Format to an OVA Archive

To convert VMX files to an OVA archive, use the following syntax:

```
> ovftool vmxs/Nostalgia.vmx ovfs/Nostalgia.ova
```

Converting an OVA Archive to VMX Format

To convert an OVA archive to VMware runtime format (.vmx and .vmdk files), use the following syntax:

```
> ovftool https://my_ovf_server/ovfs/my_vapp.ova /vm/my_vm.vmx
```

Converting an OVF Package to VMX Format

To convert an OVF package to VMware runtime format (.vmx and .vmdk files), use the following syntax:

```
> ovftool http://www.mycompany.com/ovflib/BigDemo.ovf x:/myvms/BigDemo.vmx
```

Because the source is an OVF package, you can specify it as a URL or a local file path.

If you convert an OVF package to a VMX format without specifying the target directory, OVF Tool creates a directory using the OVF package name and writes the `.vmx` and `.vmdk` files in it.

```
> ovftool "Windows 7.ovf" .
```

The VMX file is written at `Windows 7/Windows 7.vmx`.

You can also convert from an ovf format to a vmx format using a URL, as shown:

```
> ovftool https://my_ovf_server/ovfs/my_vapp.ova /vm/my_vm.vmx
```

Installing an ESXi host from an OVF Package

To install an OVF package as an ESXi host, use the following syntax:

```
> ovftool /ovfs/my_vapp.ovf vi://username:pass@my_esx_host
```

(Uses default mappings.)

Installing an ESXi host from an OVF Package on a Web Server

To install an OVF package on a web server as an ESXi host, use the following syntax:

```
> ovftool http://my_ovf_server/ovfs/my_vapp.ovf vi://username:pass@my_esx_host
```

(Uses default mappings.)

Installing an ESXi host or Adding Files from a VMX Format

To install an ESXi host from a VMware runtime format (.vmx and .vmdk files), or to add .vmx files to an ESXi host, use the following syntax (uses default mappings):

```
> ovftool /ovfs/my_vm.vmx vi://username:pass@my_esx_host
```

```
> ovftool Nostaliga.vmx vi://user:pwd@host/Datacenter/host/host1.foo.com
```

Installing a vCenter Server or Adding Files from an OVF Package

To install or add files to a vCenter Server from an OVF package, use the following syntax (uses a managed ESXi host's IP address):

```
> ovftool /ovfs/my_vapp.ovf vi://username:pass@my_vc_server/?ip=10.20.30.40
```

Converting a VM on ESXi or vCenter Server to an OVF Package

This example uses a datastore location query to convert a VM (located on a vCenter Server) to OVF format.

```
> ovftool vi://username:pass@my_vc_server/my_datacenter?ds=[Storage1] foo/foo.vmx c:\ovfs\
```

or

```
> ovftool vi://username:pass@my_host/my_datacenter/vm/my_vm_folder/my_vm_name /ovfs/
my_vapp.ovf
```

Installing vCenter Server from an OVF Package Using an Inventory Path

This example uses a vSphere inventory path to install or add files to a vCenter Server from an OVF package.

```
> ovftool /ovfs/my_vapp.ovf vi://username:pass@my_vc_server/my_datacenter/host/my_host
```

Setting OVF Package Properties

The following sections show you how to set properties for OVF packages.

Setting OVF Properties When Deploying to vSphere or vCloud Director

OVF descriptors can contain configuration properties for the deployed OVF package. You can set only one property at a time, but you can have multiple instances of the option per command. The property option has the following syntax:

```
--prop:<option>=<value>
```

This example sets two properties: the administrator's email address and the number of concurrent sessions.

```
> ovftool --prop:adminEmail=john@example.com --prop:concurrentSessions=200 package.ovf
vi://localhost/?dns=fast-esx=host1.example.com
```

Setting OVF Network Maps When Deploying to vSphere

OVF descriptors can use symbolic identifiers for network names. These identifiers must be mapped to a network that is available on the chosen vSphere platform. If only one network is available on the target and only one network is described in the OVF descriptor, OVF Tool selects that network automatically. In this case, you do not need to specify a network map. The **`--net`** option has the following syntax:

```
--net:<OVF network name>=<target network name>
```

In the following example, a network is selected.

```
> ovftool --net:"Example net 1"="VM Network" <source> <vSphere locator>
```

If the OVF descriptor only specifies one network name, you can specify the target network name of the network mapping, as in the following example:

```
> ovftool --network="VM Network" <source> <vSphere locator>
```

Setting a vService Dependency

You can dependency so that your application deploys as a service using the following option:

```
ovftool --vService:vDep1=provider_1 /ovfs/my_vapp.ovf
     vi://username:pass@localhost/my_datacenter/host/esx01.example.com
```

Modifying an OVF Package

The following sections show you how to modify OVF packages.

Renaming the OVF Package

You can rename an OVF package by converting the OVF to an OVF. This action also renames all the disk names and changes the references in the OVF descriptor.

```
> ovftool "Windows 7.ovf" win7.ovf
```

Omitting Disks in the VMware OVF Tool Output

If you want only information about the OVF descriptor and not about the disks that it refers to, you can suppress output. The following example command omits disk output and simply copies the OVF descriptor and any message bundle files that might be associated with it:

```
> ovftool --noDisks http://example.com/ovf/InterestingVirtualAppliance package.ovf
```

Compressing an OVF Package

For maximum compression of an OVF package with multiple virtual machines, set both the **`--compress=9`** and **`--makeDeltaDisks`** options. The following are examples of using maximum compression:

```
> ovftool --compress=9 --makeDeltaDisks package.ovf output-dir
```

```
> ovftool --compress=9 --makeDeltaDisks vi://localhost/dc/vm/VirtualAppDemo output-dir/
```

If the source contains only a single virtual machine, the **`--makeDeltaDisks`** option does not yield any compression boost. In this case, the **`--compress=9`** option gives maximum compression.

Chunking or Splitting OVF Packages

Some file systems have a restriction on maximum file size. For example, FAT32 allows files only up to 2GB.You can split the OVF files from a generated package into pieces of a specified maximum size. The default measurement is megabytes (keyword **`mb`**). You can specify other units using one of the following keywords:

Unit	Keyword
Bytes	**`b`**
Kilobytes	**`kb`**
Gigabytes	**`gb`**

For example, to create an OVF package optimized for a FAT32 file system, use the following command:

```
> ovftool --chunkSize=2gb <source> package.ovf
```

Each file chunk has a sequentially numbered suffix. For example, for a 6GB disk, the chunks have these names:

```
disk1.vmdk.000000000, disk1.vmdk.000000001, disk1.vmdk.000000002
```

TPM as a Virtual Device in OVF

Trusted platform module (TPM) is a hardware device that can establish a chain of trust for the software that is being loaded onto a machine. VMware virtual TPM (vTPM) is a fully compatible implementation for virtual machine equivalence.

If you want a VM to contain a vTPM device, you can either create it as a placeholder in the OVF and have it auto configured, or you add the vTPM device after OVF import.

Prepare OVF with Virtual TPM Placeholder

This procedure is convenient for OVF re-use in the long run, but requires more work initially:

1. Prepare the guest VM for vTPM removal by ensuring that none of its applications are using the vTPM device, such as BitLocker, etc.
2. Remove the vTPM device from the VM. The VM home will be decrypted unless there is a feature or policy that requires encryption.
3. If it was encrypted, decrypt the VM. OVF export does not work on encrypted VMs, so the VM must be decrypted before export.
4. Export the VM.
   - Using **`ovftool`** (recommended) you can add a vTPM placeholder in the destination OVF descriptor with the `addDevice:vtpm` option. This option creates the vTPM placeholder in the destination OVF descriptor (*`file`*`.ovf`) and calculates the proper manifest file checksum. For example:

     ```
     ovftool --addDevice:vtpm vi://user:pass@my_vc/datacenter?ds=[Storage1] vm/vm.vmx
     c:\ovfs\
     ```

   - Using the vSphere Client UI (more difficult) you can export the VM to an OVF template. After export, open the OVF descriptor file (*`file`*`.ovf`) and manually add a vTPM device placeholder, as below.
     You must also open the OVF manifest file (`.mf`) and fix the SHA256 checksum for the `.ovf` file. Because you manually edited the OVF descriptor file, the checksums in the OVF manifest are no longer correct. You must recalculate the checksum for the `.ovf` file and manually set it in the manifest file.

     (Optional) Re-sign the OVF template if it should be security signed. Perform this step after editing the `.ovf` and `.mf` files.

     ```
     <Item ovf:required="false">
       <rasd:AutomaticAllocation>false</rasd:AutomaticAllocation>
       <rasd:ElementName>Virtual TPM</rasd:ElementName>
       <rasd:InstanceID>13</rasd:InstanceID>
       <rasd:ResourceSubType>vmware.vtpm</rasd:ResourceSubType>
       <rasd:ResourceType>1</rasd:ResourceType>
     </Item>
     ```

5. Import the OVF to VSphere. You can do this with **`ovftool`**. Upon importing the VM, the vTPM device will be auto-provisioned, and a unique Endorsement Key (EK) will be generated.

   NOTE
   The OVF descriptor will contain a placeholder for a vTPM device, which does not contain any secrets. The actual EK will be set upon when creating a VM from this OVF template upon import.

Prepare OVF and Add Virtual TPM During or After Import

1. Prepare the guest VM for vTPM removal by ensuring that none of its applications are using the vTPM device, such as BitLocker, etc
2. Remove the vTPM device from the VM. The VM home will be decrypted unless there is a feature or policy that requires encryption.

3. If it was encrypted, decrypt the VM. OVF export does not work on encrypted VMs, so the VM must be decrypted before export.
4. Export the VM. You can do this either with the vSphere Client UI or with **ovftool**.
5. Import the OVF into vSphere.
   - Unless the OVF contains a vTPM placeholder, import using the **ovftool**`--addDevice:vtpm` option. Specifying this option on the command line creates a vTPM device attached to the destination VM. If the OVF contains a vTPM placeholder, you do not need the `--addDevice` option. The placeholder will automatically become a vTPM device on the imported VM.
   - With the vSphere Client, import the VM. After importing the VM, you can add a vTPM device manually using the vSphere Client.

Deploying OVF Packages

The following sections show various ways to deploy OVF packages.

Deploying an OVF Package Directly on an ESXi Host

The following command deploys an OVF package on an ESXi host.

```
> ovftool package.ovf vi://my.esx-machine.example.com/
```

If your host has multiple data stores, select with `-ds` option:

```
> ovftool package.ovf -ds=storage1 vi://my.esx-machine.example.com/
```

See also Run OVF Tool With ESXi and vCenter Target Locations.

Deploying an OVF Package using Pull Mode

OVF Tool 4.4.1 with vSphere 6.7 introduced the pull mode flag, which allows an ESXi host to directly download (pull) from a remote URL, assuming connectivity. Previously, data had to pass through another system, adding an unnecessary hop if an ESXi host can directly access the remote URL.

Pull mode speeds up file transfers and allows for quicker deployments of Virtual Machines. It enables an option in which OVF source files (VMDKs and others) are copied directly to ESXi from HTTP(S) source without going through **ovftool**.

Pull mode is actually a capability of ESXi hosts that **ovftool** takes advantage of. If property `pullModeSupported` is true in the `HttpNfcLease` managed object, then pull mode is triggered for deployments using the **--pullUploadMode** flag. Here are caveats for this option:

- The ESXi host must have access to HTTP(S) source, and permission to view the source files.
- The source must be HTTP(S) and not a filesystem based URL, for example http://www.example.com:80/path/to/package.ova or https://www.example.com:443/path/to/package.ova
- The ESXi host validates the source HTTP server before access, so the Thumbprint of the HTTP source server should be supplied. If the Thumbprint is not supplied the ESXi host may or may not be able to validate the server. If the source HTTP server is not known to the ESXi then source validation fails and deployment fails also.
- Validation described in last step, with Pull mode, cannot be suppressed with the **--noSSLVerify** flag.
- Flag **--verifyViTargetManifest** will not work with Pull mode due to the way checksum of the transferred file is done. In Pull mode **ovftool** is not involved in transferring the file so it cannot calculate the checksum and compare it to what the manifest shows.

Syntax and example of working command (final two lines are continuations):

```
ovftool --pullUploadMode source target

ovftool --acceptAllEulas --pullUploadMode
```

```
https://download3.vmware.com/software/vmw-tools/nested-esxi/AV50_EX_OVF.ovf

vi://user:pass@192.168.1.1/Datacenter/host/Cluster
```

Deploying an OVF Package through vCenter Server

The following command deploys `testVM.ovf` from a local Windows disk to a data store named storage1 on host (10.2.3.56) from vCenter (10.2.3.4). the VM will be named **`myVM`** on the host. (To test this on your system, replace the incorrect IP addresses in this example with your actual data store and host IP addresses.)

```
> ovftool -ds=storage1 -n=myVM C:\testVM.ovf vi://user1:passwd@10.2.3.4/?ip=10.2.3.56.
```

Deploying an OVF Package and Powering It On

OVF Tool can power on a virtual machine or vApp after deployment. This action can be done on all supported platforms. The following example powers on the VM or vApp on a particular host through vCenter Server:

```
> ovftool --powerOn package.ovf vi://MyvCenterServer/?dns=fast-esx-host1.example.com.
```

Deploying an OVF Package into vCloud Director

You can deploy an OVF package from OVF Tool into vCloud Director. The following example connects to vCloud Director and deploys the OVF package LAMP.ovf.

```
> ovftool --net:"VM Network=intnet" LAMP.ovf "vcloud://jd:PASSWORD@example.com:443/?
org=myOrg&vapp=test1&vdc=orgVdc&catalog=catalog"
```

Deploying an OVF Package into a vApprun Workspace

A vApprun workspace allows Workstation and Fusion users to run vApps. It provides a complete vApprun execution environment, that includes nested vApps, OVF properties, and an OVF environment. The environment is fully compatible with vSphere 4 and all later releases.

Read more about vApprun at: http://labs.vmware.com/flings/vApprun.

To deploy an OVF package into a vApprun workspace, simply use a target locator that points to your vApprun workspace, as shown in the following example:

```
> ovftool myOvfPackage c:\My_vApprun_workspace\
```

A common scenario is that the current directory is the vApprun workspace (since all vApprun commands are relative to this), so you can just use a “.” as the target locator, as shown in the following example:

```
> ovftool http://www.mycompany.com/ovflib/BigDemo.ovf.
```

Importing an OVF Package

The following sections show you how to import OVF packages.

Importing an OVF File into a vCloud instance

This section provides two examples of importing using a URL to create a vCloud instance from an OVF file. The first example names the resulting vApp as ‘vApp’.

```
> ovftool http://my_ovflib/vm/my_vapp.ovf \

           vcloud://username:pass@my_cloud?
org=MyOrg&vdc=MyVDC&catalog=MyCatalog&vapp=myVapp
```

(Imports an OVF from http into a vCloud instance and names the vApp myVapp)

The second example also creates a vApp template:

```
> ovftool http://my_ovflib/vm/my_vapp.ovf \

vcloud://username:pass@my_cloud?
org=MyOrg&vdc=MyVDC&catalog=MyCatalog&vappTemplate=myTemplate
```

(This imports an OVF from http into a vCloud instance and creates a vApp template)

Importing a Virtual Machine from vSphere to vCloud

The following command imports a virtual machine from vSphere into a vCloud instance and names the resulting vApp 'myVapp'.

```
> ovftool vi://username:pass@my_host/my_datacenter/vm/my_vm_folder/my_vm_name \

           vcloud://username:pass@my_cloud?
org=MyOrg&vdc=MyVDC&catalog=MyCatalog&vapp=myVapp
```

Importing VMX Files into a vApprun Workspace

To import a VMX format into a vApprun workspace, use the following syntax:

```
> ovftool /virtualmachines/MyVM.vmx ~my_vApprun_workspace/
```

Exporting Virtual Machines to OVF Packages

The following sections show you how to export virtual machines within OVF packages.

Exporting a Virtual Machine from a vCloud instance to an OVF Package

The following command exports a Virtual Machine from a vCloud instance into an OVF package.

```
> ovftool vcloud://username:pass@my_cloud?
org=MyOrg&vdc=MyVDC&catalog=MyCatalog&vapp=myVapp \

           /ovfs/myVapp.ovf
```

Exporting a Running Virtual Machine or vApp from vSphere

You must power off a virtual machine or vApp before exporting it. The following example locates the virtual machine or vApp based on its DNS name through the vCenter Server and powers it off:

```
> ovftool --powerOffSource vi://MyvCenterServer/?dns=test-vm test-vm.ova
```

> **NOTE**
> This option does not perform a shutdown of the operating system. This is only a power off operation.

Exporting a vApprun Entity to an OVF Package

Both virtual machine and vApp entities in your vApprun workspace can be exported as OVF packages, as shown in the following example:

```
> ovftool c:\My_vApprun_workspace\BigDemo c:\ovflib\
```

Prepend the name of the entity to export to the path. If the current directory is the vApprun workspace, you only specify the name, as shown in the following example. This example takes advantage of the fact that any source locator can be used with any destination locator. Thus, the vApp transfers directly from the vApprun workspace to the vCenter installation.

```
> ovftool BigDemo vi://MyvCenterServer/...
```

> **NOTE**
> vApprun does not keep the same level of meta-data around as vSphere. Thus, the vApprun-created OVF packages will not contain any EULAs, description of properties, and such.

Displaying Summary Information

To display a summary of information about the OVF package [in probe mode], use the following syntax:

```
> ovftool https://my_ovflib/vm/my_vapp.ovf
```

(shows summary information about the OVF package [probe mode])

Validating an OVF 1.0 or OVF 1.1 Descriptor

If you are generating OVF 1.0 or OVF 1.1 descriptors manually, you can check whether the descriptors comply with OVF 1.0 or OVF 1.1. The following examples show how to validate descriptors:

```
> ovftool --schemaValidate package.ovf

> ovftool --schemaValidate package.ova

> ovftool --schemaValidate http://example.com/folder1/package.ovf

> ovftool --schemaValidate http://example.com/folder1/package.ova
```

If everything is correct, OVF Tool shows the result of probing OVF. Otherwise, it shows warnings and errors.

> **IMPORTANT**
> Being compliant with OVF 1.0 or 1.1 is only part of the requirements for a valid OVF package. Schema validation does not check for all the requirements specified in the OVF 1.0 and OVF 1.1 specifications.

Downloading an OVF Package from a Protected Web Site

The OVF Tool can read sources given by a URL using both HTTP and HTTPS. You access it with the user name and password. The following example downloads the LAMP OVF package and puts it in an OVA package. If you omit the user name and password, the OVF Tool will prompt you for them.

```
> ovftool https://user:pass@example.com/repository/ovf/LAMP.ovf LAMP.ova
```

Using a Proxy

You can specify a proxy for the OVF Tool. The following examples show the use of the **`--proxy`** option:

```
> ovftool --proxy=proxy.example.com http://external-site.com/ovf/package.ovf

> ovftool --proxy=http://proxy.example.com http://external-site.com/ovf/package.ovf
```

The OVF Tool allows proxies that require authentication. Credentials are supplied in the proxy path as shown in the following example:

```
> ovftool --proxy=user:pass@proxy.example.com http://external-site.com/ovf/package.ovf
```

If you omit the username or password for a proxy server that requires authentication, OVF Tool prompts for them. The --proxy option enables a proxy for either vSphere (VI) or vCloud.

```
> ovftool --proxy=proxy.example.com package.ovf vi://my.esx-machine.example.com/

> ovftool --proxy=proxy.example.com package.ovf

"vcloud://USERNAME:PASSWORD@example.com:443/?
org=myOrg&vapp=test&vdc=orgVdc&catalog=catalog"
```

Overwriting a Running Virtual Machine or vApp from vSphere

The OVF Tool supports overwriting existing targets. If a target virtual machine or vApp has the same name as the source, OVF Tool overwrites the target when the **`--overwrite`** option is specified. If the target virtual machine or vApp is running, OVF Tool cannot overwrite it. OVF Tool does not automatically power off the target. To power off the target before overwriting it, use the **`--powerOffTarget`** option.

```
> ovftool --overwrite --powerOffTarget package.ovf vi://localhost/?dns=production-
host.example.com
```

You can also power on the newly written virtual machine or vApp at the same time. In the following example, the target machine is powered off and deleted, the package.ovf is imported, and the imported virtual machine or vApp is powered on.

```
> ovftool --overwrite --powerOffTarget --powerOn package.ovf vi://localhost/?
dns=production-host.example.com
```

Canceling the VMware OVF Tool While it Is Running

To cancel OVF Tool while it is running, type `Ctrl-C`. This halts OVF Tool and cleans up any generated files.

OVF Package Signing

A valid OVF signature requires two special files, a manifest (**`.mf`**) file that contains the SHA hash codes of all the files in the package (except the **`.mf`** and `.cert` files), and a certificate file (`.cert`) that contains the signed SHA of the manifest file and the X.509 encoded certificate. This appendix specifies how to use OpenSSL and VMware OVF Tools commands to sign and validate OVF packages.

Publishing an OVF Package with a Digital Signature

If you are a vendor who has produced a virtual appliance that you want to publish as an OVF appliance for public consumption, you should incorporate a digital signature.

Digital signatures work by proving that a message or document was not modified (intentionally or unintentionally) from the time it was signed. Digital signatures do this by generating a unique hash of the message or document and encrypting it using the sender's private key. For more information, see *`Understanding Digital Signatures`* at the cisa.gov website.

OVF Tool helps in distributing virtual appliances by performing all required operations to digitally sign an OVF package. Operation steps include:

1. Calculate the SHA hashes of each file (OVF file, VMDK files, and so on).
2. Create a manifest file and store the calculated SHA hashes from the previous step.
3. Use the vendor's private key to digitally sign the manifest file's content thereby producing a digital signature.
4. OVF Tool then stores the digital signature from the previous step in the cert file. OVF Tool also stores the vendor's public key in the cert file for client side signature verification.

The steps above can also be done using the **`openssl`** tool.

> **NOTE**
> SHA1 is no longer considered secure. SHA256 and above should be used instead. See the `--shaAlgorithm` command line option.

When a customer receives the vendor's OVF package, they can perform the following steps to verify integrity of the OVF package contents:

1. Customer uses the public key in the cert file to verify (for example with **`openssl`**) the signature of the manifest file content in the cert file, checking that the signature matches the manifest file contents.
2. Customer uses SHA hashes in the (now verified) manifest file to verify (with **`openssl`**) that the OVF file hash and the VMDK file hash match those in the manifest file. This proves that the files have not been corrupted or tampered with since the vendor published them.

OVF Tool can perform the above steps automatically and does so by default if a `.cert` file is found in the OVF package.

> **NOTE**
> If option `--requireSignature` is passed to the OVF Tool command, the `.cert` and `.mf` files must exist in the OVF package and they must be valid.

VMware signs their own appliances. OVF Tool and vCenter Server Appliance have an embedded copy of the VMware public key to verify digital signatures of those products.

Creating an RSA Public/Private Key Pair and Certificate

To sign a package, a public/private key pair and certificate that wraps the public key is required. The private key and the certificate, which includes the public key, is stored in a `.pem` file.

The following OpenSSL command creates a `.pem` file:

```
openssl req -x509 -nodes -sha256 -days 365 -newkey rsa:1024 -keyout myself.pem -out
myself.pem
```

> **NOTE**
> No password is necessary. To include a password, remove the **`--nodes`** option.

Contents of the resulting `myself.pem` file are shown below.

myself.pem File

```
-----BEGIN RSA PRIVATE KEY-----

    MIICXAIBAAKBgQDe0dCCKNfQ45+D0ezGGAuVSbhE8buqFCQnQnfi27Wt6bu4DhcE
```

```
    bQtjgfzuEpcl4e31txJcu18XTv4icRL74DP7i2pMN2UVj6DZW/B7jIw4UPG2g96f
    ...
    -----END RSA PRIVATE KEY-----
    -----BEGIN CERTIFICATE-----
    MIIC5DCCAk2gAwIBAgIJAKgUiZPOajC0MA0GCSqGSIb3DQEBBAUAMFYxCzAJBgNV
    BAYTAkRLMRMwEQYDVQQIEwpTb21lLVN0YXRlMQ8wDQYDVQQHEwZBYXJodXMxITAf
    ...
    -----END CERTIFICATE-----
```

To display the contents of a `.pem` file at the command line, type the following:

```
openssl -x509 -text -noout -in <filename>.pem
```

Contents of the file display as follows:

```
Certificate:
     Data:
        Version: 3 (0x2)
        Serial Number:
....
```

To create a trusted certificate, use the OpenSSL command, omitting the **`-x509`** option. This creates a certificate request in a `.pem` file that you can send to any public authority such as Verisign.

Signing an OVF Package

Signing an OVF package enables the person deploying it to validate the authenticity of the OVF package. Once the package is signed, OVF package files cannot be changed, without invalidating the signature. When a package comes from a trusted source and has a valid OVF signature, you can deploy the package knowing it has not been tampered with.

Signing an OVF package requires a `.pem` file that contains a private key and a certificate, as shown in section Creating an RSA Public/Private Key Pair and Certificate.

To sign a generated OVF package, include the --privateKey option. The option syntax is shown in the following example:

```
> ovftool --privateKey=<path to .pem file> <source> <output OVF or OVA file>
```

When this option is used, OVF Tool uses the private key and certificate to generate a signature based on the SHA digest of each file that is included in the OVF package, including the OVF descriptor itself.

OVF Tool generates an additional `.cert` file with a signed SHA signature and the certificate used to sign it. Certificate File Created by OVF Tool shows an example of the `.cert` file generated by OVF Tool.

Certificate File Created by OVF Tool

```
SHA256(signed-
package.mf)=5d9a307f0acdc1a424079eb38ff8954c153f978e599ed374dd784c853bab1856415fa16ef378bd
e3487cd5dfa4d11a3017eda91886f98e3bba3adc2f4e28ce6d0ba3a19eef80ac0729511311603dcb221f9ba7a6
008f1a87fe15ebf3699c8a8744bd05c43b1387dd53d73723e7f0a3720d489e147e31c4570d15fb7a3beae770
```

```
-----BEGIN CERTIFICATE-----
MIIDTzCCArigAwIBAgIJAKDgFLg9WvBwMA0GCSqGSIb3DQEBBQUAMHkxCzAJBgNV
BAYTAkRLMQ8wDQYDVQQHEwZBYXJodXMxFTATBgNVBAoTDFZNd2FyZSwgSW5jLjEM
MAoGA1UECxMDVklNMREwDwYDVQQDEwhLcmlzdGlhbjEhMB8GCSqGSIb3DQEJARYS
a2xhc3NlbkB2bXdhcmUuY29tMB4XDTA5MDMwNjEzMDUwNFoXDTEwMDMwNjEzMDUw
NFoweTELMAkGA1UEBhMCREsxDzANBgNVBAcTBkFhcmh1czEVMBMGA1UEChMMVk13
YXJlLCBJbmMuMQwwCgYDVQQLEwNWSU0xETAPBgNVBAMTCEtyaXN0aWFuMSEwHwYJ
KoZIhvcNAQkBFhJrbGFzc2VuQHZtd2FyZS5jb20wgZ8wDQYJKoZIhvcNAQEBBQAD
gY0AMIGJAoGBAM2xxX9a1YITiiRrxpXGg9xbEP4Oepcs71ZcNp8Z3mQIb95mpEc6
SZemmjOsqwpkvV/82RALOBgmJ/hot1noSkiAZi0liPmX1M0BU3OS/pSim7VNKBmV
SUJfOC4T6/MygVpyfkSUhB5EWx0JCUvowRex6Ytl220MOGcXnLpvdfO9AgMBAAGj
gd4wgdswHQYDVR0OBBYEFM2KkX7pWTQmMg+iD6HWMOZRLrfJMIGrBgNVHSMEgaMw
gaCAFM2KkX7pWTQmMg+iD6HWMOZRLrfJoX2kezB5MQswCQYDVQQGEwJESzEPMA0G
A1UEBxMGQWFyaHVzMRUwEwYDVQQKEwxWTXdhcmUsIEluYy4xDDAKBgNVBAsTA1ZJ
TTERMA8GA1UEAxMIS3Jpc3RpYW4xITAfBgkqhkiG9w0BCQEWEmtsYXNzZW5Adm13
YXJlLmNvbYIJAKDgFLg9WvBwMAwGA1UdEwQFMAMBAf8wDQYJKoZIhvcNAQEFBQAD
gYEANaNxv4QrN7iI0rDCordYDh1G7Z3jl28ntSoxehGmz6ghYAfBNhTVhWUZuX9X
UXKn8QltOF/Ynijuo6JTJwO/5V1o6TAaCmFahDW/Om02AXPdSbw4UQdidGmmgrAs
DYVQz2CNPk2YbkXITNeGBNHomTqsVU7MGDjReu96+V6O2zY=
-----END CERTIFICATE-----
```

Validating an OVF Package

If an OVF certificate file is present, OVF Tool always verifies if the signature fits the SHA digest of the files in the package and tests the authenticity of the certificate.

To quickly validate the authenticity of an OVF package, use the probe mode as shown in the following example:

```
> ovftool signed-package.ovf
```

Using OVF Tool Probe Mode

About OVF Tool Probe Mode

Probe mode reveals information about the content of a source. You can probe OVA and OVF packages, VMX, and vSphere source types. You can then use the information gathered to find out how it can be configured when you deploy it.

To use the probe feature, omit the target locator when invoking OVF Tool. For example, type: **`ovftoolLAMP.ovf`** at the command line. OVF Tool displays all available information about the `LAMP.ovf` package.

When probe mode is used on an OVF or OVA package, OVF Tool also validates the certificate file, if present. As of vSphere 8.0 U2, OVF Tool probe mode also detects and handles the virtual TPM (vTPM) device.

As part of the information displayed in probe mode, the EULA is displayed by default. To prevent the EULA from displaying, use the **`--hideEula`** option.

```
> ovftool --hideEula LAMP.ovf
```

Example of Probe Mode

The following example shows the result of probing the `LAMP.ovf` package.

```
OVF version:   1.0
VirtualApp:    true
Name:          LAMP running PHP-Fusion
Version:       0.1
Vendor:        VMware Aarhus
Product URL:   http://example.com/ovf/1.0/LAMP/readme.txt

Annotation:  This vApp offers the programming environment stack: Linux, Apache,
           MySQL and PHP programming environment -- LAMP. More specifically
           the vApp contains a Database server running MySQL and Web server
           VM running Apache2 and PHP.

End-user License Agreements:

  EULA for LAMP.

Download Size:   604.07 MB

Deployment Sizes:
Flat disks:     16.00 GB
Sparse disks:   Unknown
Networks:
Name:        VM Network
Description: The VM Network network
```

```
Virtual Hardware:
Family:       vmx-04
Disk Types:   SCSI-lsilogic

Properties:
Key:         db_ip
Label:       IP address
Type:        ip:VM Network
Description: The IP address of the database server.

Key:         ws_ip
Label:       IP address
Type:        ip:VM Network
Description: The IP address of the Web server.

IP Allocation Policy:
Schemes:     ovfenv dhcp
Protocols:   IPv4
```

Using OVF Tool Machine Mode

About Machine Mode

You can use the --machineOutput option to run OVF Tool from another program or script. With the --machineOutput option, OVF Tool provides output in the following format:

```
STATUS-CODE details <blank line>.
```

OVF Tool inserts a blank line to signal the end of an operation. Each response line is prefixed with a plus (+) to avoid confusion with the terminating blank line. The last status it sends is always RESULT. OVF Tool sends all output, including errors and warnings, to standard output (stdout) so clients can listen on only one stream.

Table 129: Machine Mode Status

Status	Details	Description
PROBE	XML	Probe result with information about the source.
VALIDATEHOST	XML	Shows whether the VI target is compatible with the input arguments.
AUTHENTICATION	source/target/proxy server locator or fileName	Shows that authentication is required.
CERTIFICATE	Validate, Self-signed, or Failed validate	Signals that a certificate is present and shows the result of the validation.
MANIFEST	**`Validate`** or Failed validate	Signals that a manifest is present and shows the result of validation.
PROGRESS	Number (0-100)	Shows the percentage progress during upload and download.
TARGET_ID	Text, for example SugarCRM.ovf or vim.VirtualMachine:vm-415.	Shows the target ID after upload and download finishes.
RESULT	ERROR or SUCCESS	Always use as the last command sent.

Running Machine Output

You can run the --machineOutput option in different modes: Probe, Validate host, Import, and Export.

Running machineOutput in Probe Mode

When you run the machineOutput option in probe mode, OVF Tool reports the following status sequence:

1. AUTHENTICATION (zero or more)
2. PROBE (exactly one)
3. RESULT (exactly one)

To run the machineOutput option in probe mode, you run the following command.

```
ovftool.exe --machineOutput source_locator
```

For an example, see Output from Running machineOutput in Probe Mode.

Running machineOutput in Validate Host Mode

When you run machineOutput in validate host mode, OVF Tool reports the following status sequence:

1. AUTHENTICATION (zero or more)
2. VALIDATEHOST (exactly one)
3. PROGRESS (exactly one)
4. TARGET_ID (exactly one)
5. RESULT (exactly one)

To run the machineOutput option in validate host mode, you run the following command.

```
ovftool.exe --machineOutput --verifyOnly source_locator destination_locator
```

For an example, see Output from Running machineOutput in Validate Host Mode.

Running machineOutput in Import to vSphere Mode

When you run the machineOutput option in import mode, OVF Tool reports the following status sequence:

1. AUTHENTICATION (zero or more)
2. MANIFEST (zero or one)
3. CERTIFICATE (zero or one)
4. PROGRESS (one or more)

5. TARGET_ID (exactly one)
6. RESULT (exactly one)

To use machine mode to upload an OVF to vSphere, you run the following command.

```
ovftool.exe --machineOutput                                          \

            --acceptAllEulas                                         \

            --I:morefArgs                                            \

            --I:targetSessionTicket=<session ticket>                 \

            --net:<ovf netname>=vim.Network:<moref-id>               \

            --datastore=vim.Datastore:<moref-id>                     \

            --vmFolder=vim.Folder:<moref-id>                         \

            --deploymentOption=<value>                               \

            --diskMode=<value>                                       \

            --ipAllocationPolicy=<value>                             \

            --ipProtocol=<value>                                     \

            --name=<value> (optional)                                \

            --overwrite (optional)                                   \

            --powerOffTarget (optional)                              \

            --powerOn (optional)                                     \

            --prop:<key>=<value>                                     \

            <src URL or PATH>                                        \

             vi://<servername>?moref=vim.ResourcePool:<moref-id>
```

For an example, see Output from Running machineOutput in Import Mode.

Running the Machine Mode Export from vSphere Operation

When you run the machineOutput option in export mode, OVF Tool reports the following status sequence:

1. AUTHENTICATION (zero or more)
2. MANIFEST (zero or one)
3. CERTIFICATE (zero or one)
4. PROGRESS (one or more)
5. TARGET_ID (exactly one)
6. RESULT (exactly one)

To use machine mode to download an OVF from vSphere, you run the following command.

```
ovftool.exe --machineOutput                                          \

            --I:sourceSessionTicket=<session ticket>                 \

            -tt <OVA or OVF>                                         \

            -n=<name>                                                \

            --overwrite (optional)                                   \
```

```
--powerOffSource (optional)                              \
--chunkSize=<value> (optional)                           \
--compress=<value> (optional)                            \
vi://<servername>?moref=<type>:<moref-id>                \
<directory>
```

The type value is either vim.VirtualMachine or vim.VirtualVApp.
When you specify --machineOutput, OVF Tool monitors STDIN, and cancels the operation if it reads the ABORT\n line in stdin.
For an example, see Output from Running machineOutput in Export Mode.

Example Output

You can run the OVF Tool machine mode **`--machineOutput`** option in probe mode, validate host mode, or import mode. In import and validate Host modes, **`--machineOutput`** is meant for third party program to use silently without any interactive prompt appearing in the standalone OVF tool command console. If your OVF file contains EULA information, you should add this option **`--acceptAllEulas`** so that the program doesn't pause to wait for acceptance of the license agreement. This section contains the following topics:

- Output from Running machineOutput in Probe Mode
- Output from Running machineOutput in Validate Host Mode
- Output from Running machineOutput in Import Mode
- Output from Running machineOutput in Export Mode

Output from Running machineOutput in Probe Mode

The following example shows the output of a --machineOutput PROBE operation on a file named LAMP.ovf.

Output from Running machineOutput in Probe Mode

```
ovftool --machineOutput LAMP.ovf
PROBE
+ <probeResult>
+ <virtualApp>
+ true
+ </virtualApp>
+ <productInfo>
+ <name>
+ LAMP running PHP-Fusion
+ </name>
+ <productUrl>
+ http://example.com/ovf/1.0/LAMP/readme.txt
+ </productUrl>
```

```
+ <version>
+ 0.1
+ </version>
+ <fullVersion>
+
+ </fullVersion>
+ <vendor>
+ VMware
+ </vendor>
+ <vendorUrl>
+
+ </vendorUrl>
+ </productInfo>
+ <annotation>
+ This vApp offers the programming environment stack: Linux, Apache, MySQL and PHP
programming
environment -- LAMP. More specifically the vApp contains a Database server running MySQL
and Web
server VM running Apache2 and PHP.
+ </annotation>
+ <eulas>
+ <eula>
+
+    Eula for OVF
+
+ </eula>
+ </eulas>
+ <sizes>
+ <download>
+ 633412608
+ </download>
+ <flat>
+ 17179869184
```

```
+ </flat>
+ <sparse>
+ Unknown
+ </sparse>
+ </sizes>
+ <networks>
+ <network>
+ <name>
+ VM Network
+ </name>
+ <description>
+ The VM Network network
+ </description>
+ </network>
+ </networks>
+ <properties>
+ <property>
+ <classId>
+
+ </classId>
+ <key>
+ db_ip
+ </key>
+ <instanceId>
+
+ </instanceId>
+ <category>
+
+ </category>
+ <label>
+ IP address
+ </label>
```

```
+ <type>
+ ip:VM Network
+ </type>
+ <description>
+ The IP address of the database server.
+ </description>
+ <value>
+
+ </value>
+ </property>
+ <property>
+ <classId>
+
+ </classId>
+ <key>
+ ws_ip
+ </key>
+ <instanceId>
+
+ </instanceId>
+ <category>
+
+ </category>
+ <label>
+ IP address
+ </label>
+ <type>
+ ip:VM Network
+ </type>
+ <description>
+ The IP address of the Web server.
+ </description>
```

```
+ <value>
+
+ </value>
+ </property>
+ </properties>
+ <deploymentOptions>
+ </deploymentOptions>
+ <ipAllocationSchemes>
+ ovfenv,dhcp
+ </ipAllocationSchemes>
+ <ipProtocols>
+ IPv4
+ </ipProtocols>
+ </probeResult>


RESULT
+ SUCCESS
```

Output from Running machineOutput in Validate Host Mode

The following example shows the output of a --machineOutput VALIDATEHOST operation on file LAMP.ovf.

Output from Running machineOutput in Validate Host Mode

```
ovftool --machineOutput --acceptAllEulas --verifyOnly LAMP.ovf vi://
myuser:mypassword@myvc.example.com/dc/host/myhost.example.com

ovftool --machineOutput --acceptAllEulas --verifyOnly LAMP.ovf vi://
myuser:mypassword@myvc.example.com/dc/host/myhost.example.com

VALIDATEHOST
+ <supportedDiskProvisioning>
+ <type>
+ monolithicFlat
+ </type>
+ <type>
+ thin
+ </type>
```

```
+ <type>
+ thick
+ </type>
+ <type>
+ flat
+ </type>
+ <type>
+ eagerZeroedThick
+ </type>
+ </supportedDiskProvisioning>



PROGRESS
+ 0


TARGET_ID
+


RESULT
+ SUCCESS
```

Output from Running machineOutput in Import Mode

The following example shows the output of a --machineOutput import operation on a file named LAMP.ovf.

Output from Running machineOutput in Import Mode

```
ovftool --machineOutput --acceptAllEulas LAMP.ovf vi://myuser:mypassword@myvc.example.com/
dc/host/myhost.example.com

PROGRESS
+ 0
+ 1
+ 2
+ 3
....
```

```
+ 98
+ 99
+ 100


TARGET_ID
+ vim.VirtualApp:resgroup-v61


RESULT
+ SUCCESS
```

Output from Running machineOutput in Export Mode

The following example shows the output of a --machineOutput export operation on a file named LAMP.ovf.

Output from Running machineOutput in Export Mode

```
ovftool -o --machineOutput --acceptAllEulas vi://myuser:mypassword@myvc.example.com/dc/vm/
LAMP /tmp/LAMP.ovf
PROGRESS
+ 0
+ 1
+ 2
+ 3
...
+ 98
+ 99
+ 100


TARGET_ID
+ /tmp/LAMP.ovf
RESULT
+ SUCCESS
```

Documentation Legal Notice

Information about the documentation legal notice.

This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the “Documentation”) is for your informational purposes only and is subject to change or withdrawal by Broadcom at any time. This Documentation is proprietary information of Broadcom and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of Broadcom.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all Broadcom copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to Broadcom that all copies and partial copies of the Documentation have been returned to Broadcom or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, BROADCOM PROVIDES THIS DOCUMENTATION “AS IS” WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL BROADCOM BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF BROADCOM IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice

The manufacturer of this Documentation is Broadcom Inc.

Provided with “Restricted Rights.” Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2005–2025 Broadcom. All Rights Reserved. The term “Broadcom” refers to Broadcom Inc. and/or its subsidiaries. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.