WASD Hypertext Services
- Technical Overview

5th December 1999

For version 6.1 release of the WASD VMS Hypertext Services HTTP server (December 1999).

Supercedes:

```
31st May 1999 (v6.0)
21st November 1998 (v5.3)
5th September 1998 (v5.2)
23rd July 1998 (v5.1)
30th March 1998 (v5.0)
15th November 1997 (v4.5)
1st October 1997 (v4.4)
1st August 1997 (v4.3)
1st July 1997 (v4.2)
1st June 1997 (v4.1)
1st October 1996 (v3.4)
1st December 1995 (initial freeware release, v3.1)
9th August 1995 (v2.3)
1st May 1995 (v2.1)
```

Please Note

As of July 1997 High Frequency Radar Division (HFRD) underwent a change of role and name. It is now Wide Area Surveillance Division (WASD). This package has been renamed in accordance with that.

Abstract

This document is a technical overview of the implementation of the WASD VMS HyperText Transport Daemon. It contains information on server startup, configuration, etc. Previous versions contained information on scripting and brief descriptions of the major code modules of the server, these have now been removed to a documents of their own (see below).

Also see “WASD Scripting Environment” for information on CGI, ISAP, OSU, etc., scripting, and “WASD Hypertext Environment” for information on using the WASD VMS Hypertext Services, and “WASD Nuts and Bolts” for technical information on server design and implementation.

It is strongly suggested those using printed versions of this document also access the Hypertext version. It provides online access to some examples, etc.

Author

Mark G. Daniel
Surveillance Systems Division
Defence Science and Technology Organisation

For WASD-related email please use
Mark.Daniel@vsm.com.au

Should the above address present problems or provide no response for an extended period then use
Mark.Daniel@dsto.defence.gov.au

+61 (8) 82596031 (bus)
+61 (8) 82596673 (fax)

PO Box 1500
Salisbury
South Australia 5108

Printed Copy

This book may be available for printing to a PostScript printer. Use a browser to access a print menu in this same location.

Some of the online demonstrations may not work due to the local organisation of the Web environment differing from WASD where it was originally written.

WASD VMS Hypertext Services

OpenSSL Project

This product *can* include software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/).

```
Redistribution and use in source and binary forms, with or without
modification, are permitted ...
```

Eric A. Young

This package *can* include cryptographic software written by Eric Young (eay@cryptsoft.com) and Tim Hudson (tjh@cryptsoft.com).

```
This library is free for commercial and non-commercial use provided ...
Eric Young should be given attribution as the author ...
copyright notice is retained
```

Ohio State University

This package contains software provided with the OSU (DECthreads) HTTP server package, authored by David Jones:

```
Copyright 1994,1997 The Ohio State University.
The Ohio State University will not assert copyright with respect
to reproduction, distribution, performance and/or modification
of this program by any person or entity that ensures that all
copies made, controlled or distributed by or for him or it bear
appropriate acknowlegement of the developers of this program.
```

RSA Data Security

This software contains code derived in part from RSA Data Security, Inc:

```
permission granted to make and use derivative works provided that such
works are identified as "derived from the RSA Data Security, Inc.
MD5 Message-Digest Algorithm" in all material mentioning or referencing
the derived work.
```

Bailey Brown Jr.

LZW compression is implemented using code derived in part from the PBM suite. This code is copyright by the original author:

```
 * GIF Image compression - LZW algorithm implemented with Tree type
 *                         structure.
 *                         Written by Bailey Brown, Jr.
 *                         last change May 24, 1990
 *                         file: compgif.c
 *
 *  You may use or modify this code as you wish, as long as you mention
 *  my name in your documentation.
```

Other

OpenVMS, **Digital TCP/IP Services for OpenVMS**, **VAX C**, **DEC C**, **VAX** and **AXP** are registered trademarks of Digital Equipment Corporation.

MultiNet is a registered trademark of Cisco Systems, Inc.

Pathway is a registered trademark of Attachmate, Inc.

TCPware is a registered trademark of Process Software Corporation.

Contents

Chapter 1 Introduction

Chapter 2 HyperText Transport Protocol Daemon - Overview

Chapter 3 New to WASD? Start Here!

Chapter 4 Installation and Update

Chapter 5 Server Account and Environment

Chapter 6 Server Configuration

Chapter 7 Server Configuration Directives

Chapter 8 Message Configuration

Chapter 9 Mapping Rules

Chapter 10 Authorization Quick Guide

Chapter 11 Authentication and Authorization

Chapter 12 Proxy Services

Chapter 13 Secure Sockets Layer

Chapter 14 Server Administration

Chapter 15 WATCH Facility

Chapter 16 Cache

Chapter 17 Server Performance

Chapter 18 HTTPd Web Update

Chapter 19 Utilities

Chapter 1

Introduction

This document provides an technical overview of the WASD VMS HTTP server. It is primarily written for use internal to WASD and assumes that perpsective without apology. Any additional usage is subordinate to its role within WASD. The software has been written expressly for supporting an intra-organisational hypertext environment on a VMS platform (June 1997 note: since first writing this the industry has settled on the term **intranet**). It too, is unreservedly tailored to this purpose and the requirements of WASD. All programs were designed only to specifically comply with the requirements of VAX C and DEC C, within a DEC TCP/IP Services for VMS environment.

The document **assumes** a basic understanding of the hypertext technologies and uses terms without explaining them (e.g. HTTP, HTML, URL, CGI, SSI, etc.) The reader is refered to documents specifically on these topics (these are often best consulted on-line, on the Internet WWW).

Also see ‘‘WASD Scripting Environment’’ for information on using CGI, ISAPI, OSU, etc. scripting, and ‘‘WASD Hypertext Environment’’ for information on using the WASD VMS hypertext facilities.

It is strongly suggested those using printed versions of this document also access the Hypertext version. It provides online demonstrations of some concepts.

Objectives

The primary impetus for an internal Web environment was a 1993 decision by Division management to make as much information as possible, both administrative and research, internally available on-line (to use the current term . . . an *intranet*). Early experimentation with a *Gopher* implementation soon made way for the obvious advantages of the emerging Web technology.

It then became the objective of this author to make *all* of our systems’ VMS-related resources available via HTTP and HTML, regardless of the underlying data or storage format. An examination of the WASD package will show that this objective is substantially achieved.

Reasons For a Local HTTPd

Reasons for developing a local HTTP server are few but compelling:

- The WASD (then HFRD) Web implementation began mid-1994 (October 1997 note).
- It was prefered to support the hypertext environment on a VMS platform. This is currently (1995) the most widely used and accessable environment within WASD.
- Existing servers (and there are quite a few variations) are largely Unix based, although it is being supported (to a greater or lesses extent) across a wide range of platforms. Ports to VMS, if they exist, are often in progress or half-baked, employing *Unix*isms that don't translate elegantly to the VMS environment.
- The VMS version of the CERN server (3.0-6) was evaluated during mid-1994:
  - It is not multi-threaded under VMS (i.e. cannot support concurrent clients). For example, a lengthy search may delay other clients for unacceptable periods.
  - Its performance was good with document transfers, but became poor when running a *script*.
  - It is acknowleged in the release notes that it cannot handle a client cancelling a data transfer (a not-uncommon action). This was confirmed experimentally.
- An early version of the OSU (DECthreads) server was evaluated via documentation mid-1994. (now in 1998 . . . I am reticent to make further comment on now such a widely deployed package :^)
- HyperText Transport Protocol, in its current form, is relatively simple to implement to the level required to support intra-Divisional requirements.
- **As of December 1995** the server has worked extremely well and has a number of facilities tailored for the VMS environment. It can continue to be utilized until there are overwhelming reasons for implementing something else.
- **As of June 1997** the server and its associated software continues to evolve and provide a stable and effective VMS Web environment, even with the advent of a small number of commercial VMS Web products.
- **As of October 1999** the package is beginning to mature as an HTTP/1.0 solution, providing not only a fast and stable server but an increasingly extensive collection of applications and tools.

Chapter 2

HyperText Transport Protocol Daemon - Overview

The most fundamental component of the WASD VMS Hypertext Services environment is the **HTTPd**, or HyperText Protocol Transport Daemon, or HTTP server. It has a single-process, multi-threaded, asynchronous I/O design.

2.1 Server Features

The package provides a complete implementation of a basic HTTP/1.0 server, including:

- concurrent, multi-threaded client support
- ‘‘GET’’, ‘‘HEAD’’, ‘‘POST’’ and ‘‘PUT’’ HTTP method support
- virtual services (servers)
- optional enhanced privacy using Secure Sockets Layer (SSL) technology (USA domestic, full-strength encryption)
- proxy serving, with local caching, for all HTTP/1.0 methods, plus the CONNECT method (allowing SSL tunnelling)
- ‘‘If-Modified-Since:’’ / ‘‘304 Not Modified’’ functionality (document is only sent if modified since time specified by client)
- HTTP/1.0 de-facto persistent connections (i.e. ‘‘Keep-Alive:’’, reducing the number of TCP/IP connects required)
- versatile directory listing (generic and VMS-style)
- CGI-compliant subprocess scripting (with configurable, automatic, MIME content-type initiated activation)
- ‘‘CGIplus’’ scripting (offering reduced latency, increased throughput and reduced system impact)
- ‘‘ISAPI’’ extensions/scripting (also offering reduced latency, increased throughput and reduced system impact)
- DECnet-based CGI scripting (with connection reuse)

- user account scripting (provided using the DECnet environment)
- OSU (DECthreads server) scripting emulation, with connection reuse (as per OSU 3.3a), allowing many OSU scripts to be employed unmodified
- script processor (e.g. PERL) configurable on file type (extension)
- Server-Side Includes (HTML pre-processing)
- configurable cache, with time-based and forced re-validation (reload)
- clickable-image support (both NCSA and CERN formats)
- *smart* rule mapping (conditional rules)
- ''Basic'' and ''Digest'' user authentication and path/group-based authorization
- SYSUAF-authentication and VMS user security profile based file access control
- Web-standard, ''common'' and ''combined'' access log formats (allowing processing by most log-analysis tools), along with a user-definition capability allowing custom log formats
- logging periods, where log files automatically change on a daily, weekly or monthly basis (keeps log files ordered and at a managable size)
- host-level access control, on per-host or per-domain acceptance or rejection
- single server can support multi-homed (host name) and multiple port services
- customizable message database (capable of supporting non-English and concurrent, multiple languages)
- on-line server configuration, including reports on requests, loaded configuration, mapping rules, authorization information and graphical activity displays
- on-line, live server processing event report (WATCH)

2.2 Server Behaviour

The HTTPd executes permanently on the server host, listening for client connection requests on TCP/IP port 80 (by default). It provides concurrent services for a (technically) unlimitted number of clients (constrained only by the server resources). When a client connects the server performs the following tasks:

1. creates a *thread* for this request
2. reads and analyzes the HTTP request sent,
   depending on the nature of the request . . .
   - transfer of a file (possibly from cache)
   - processing of an SSI HTML file
   - processing of a clickable-image mapping file
   - directory listing
   - file/directory create/update
   - server administration

- web file-system update
- interpretation of a menu file
- spawn a subprocess to to provide a CGI scripting environment:
    - standard CGI
    - WASD CGIplus
- connects to a DECnet object to provide a scripting environment:
    - standard CGI
    - OSU (DECthreads) emulated

3. closes the connection to the client and disposes of the thread data structures

For I/O intensive activities like file transfer and directory listing, the AST-driven code provides an efficient, multi-threaded environment for the concurrent serving of multiple clients.

For scripts, the technique of using multi-threaded, concurrent, spawned subprocesses or network processes, attached to standard input/output streams, provides a versatile, extensible, powerful scripting environment. Any DCL procedure or image executing within the process can behave essentially as an HTTP server. This capability is employed to easily extend the basic services provided by the core daemon code. An HTTP script/server for this environment does not need to concern itself with network activities, it merely reads and writes from the standard I/O streams.

2.3 TCP/IP Packages

The WASD server uses the Compaq TCP/IP Services (UCX) BG $QIO interface. The following packges support this interface and may be used.

- Compaq TCP/IP Services for OpenVMS (Compaq Computer Corporation), any version
- Digital TCP/IP Services for OpenVMS (aka UCX), any version
- MultiNet for OpenVMS (Process Software Corporation), any version
- TCPware (Process Software Corporation), any version

2.4 International Features

As of version 4.4 the HTTP server has undergone a limited degree of *internationalization*. The impetus for this development was the use of the package in some European countries where English is not the first language.

As this was not within the original critera while building the server it has meant an extensive rewrite of some functionality within the code (I hope it's appreciated ;^) The objective is to provide users of the server with a more familiar and therefore more comfortable environment. Unfortunately the server administrators are still confronted by an English-language interface and documentation, but then the package doesn't pretend to be *international software*!

Aspects

The international features only apply to the server, not necessarily to any scripts!

- **Server Messages**

  The server uses an administrator-customizable database of messages that can contain multiple language instances of some or all messages, using the Latin-1 character set (ISO8859-1). Although the base English messages can be completely changed and/or translated to provide any message text required or desired, a more convenient approach is to supplement this base set with a language-specific one.

  One language is designated the prefered language. This would most commonly be the language appropriate to the geographical location and/or clientele of the server. Another language is designated the base language. This must have a complete set of messages and is a fall-back for any messages not configured for the additional language. Of course this base language would most commonly be the original English version.

  More than just two languages can be supported. If the browser has *prefered languages* set the server will attempt to match a message with a language in this preference list. If not, then the server-prefered and then the base language message would be issued, in that order. In this way it would be possible to simultaneously provide for English, French, German and Swedish audiences, just for example.

  For message configuration information see Chapter 8.

- **Server Dates**

  Dates appearing in server-generated, non-administrative content (e.g. directory listings, not META-tags, which use Web-standard time formats) will use the natural language specified by any SYS$LANGUAGE environment in use on the system or specifically created for the server.

- **Virtual Services**

  Virtual-server-associated mapping, authorization and character-sets allow for easy multiple language and environment sites. Further per-request tailoring may be deployed using conditional rule mapping described below.

  For virtual services information see Chapter 6.

- **Conditional Rule Mapping**

  Mapping rules map requested URL paths to physical or other paths (see Chapter 9). Conditional rules are only applied if the request matches criteria such as prefered language, host address (hence geographical location to a certain extent), etc. This allows requests for generic documents (e.g. home pages) to be mapped to language versions appropriate to the above criteria.

  For conditional mapping information see Section 9.6.

- **Character Sets**

Generally the default character set for documents on the Web is ISO-8859-1 (Latin-1). The server allows the specification of any character set as a default for text document responses (plain and HTML). In addition, text document file types may be modified or additional ones specified that have a different character set associated with that type. Furthermore, specific character sets may be associated with mapping paths. A site can therefore relatively easily support multiple character set document resources.

Major WASD scripts also use the server default character set and/or any request determined character set (based on request path with conditional mapping to language or character set preferences) via CGI variable contents.

For further information see [CharsetDefault] and [AddType] in Section 9.2.5.

2.5 HTTP Methods Usage

This section describes WASD-specific characteristics of the available HTTP/1.0 request methods.

2.5.1 GET

Of course, the GET method is used to access documents supplied by the server. There is nothing WASD server-specific about this method.

2.5.2 POST & PUT

The WASD HTTPd does not differentiate between POST and PUT methods, both are handled identically.

Script Handling

The ‘‘normal’’ usage of the POST method is to return data from a <FORM>..</FORM> construct to a script running on the server. In this regard WASD is no different to other any web server; the form data is delivered to the script’s standard input as a stream of URL-encoded text. For example:

```
name=Fred+Nurk&address=Fred%27s+House%0D%0A0+Nowhere+Lane&submit=Submit
```

Note that WWW_CONTENT_LENGTH will be the length of the form data. See ‘‘Scripting Environment’’ document for further information.

File Creation/Upload

If the client sends data back to the server using either the POST or the PUT methods, without a script being mapped to be executed in response to that data, the WASD HTTPd will create a file corresponding to the specified path. The data stream may be text or binary.

Of course, for the server to accept POST and PUT data in this manner, authentication and authorization must be enabled and allow such access to the request path.

The data stream is processed according to MIME content-type:

- **application/x-www-form-urlencoded**

The server specially processes ‘‘application/x-www-form-urlencoded’’ POSTS (i.e. those generated by <FORM>...</FORM>, allowing files to be created directly from HTML forms. The processing eliminates any field names from the URL-encoded data stream, placing only field values into the file. This capability can be quite useful and is demonstrated in the Update HTTPd module

online hypertext link.

- **multipart/form-data**

  This server can process a request body according to RFC-1867, ‘‘Form-based File Upload in HTML’’. As yet it is not a full implementation. It will not process "multipart/mixed" subsections. The implementation is basic, providing a facility to allow the upload of a file into the server administered file system. The ACTION= parameter of the <FORM> tag must specify the directory (as a URL path) in which the uploaded file will be created.

  The following example HTML illustrates how a form may be used to upload a file from the browser host file system:

  ```
  <FORM METHOD=POST ACTION="/web/directory/" ENCTYPE="multipart/form-data">
  <INPUT TYPE=submit VALUE=" Upload document ... ">
  <INPUT TYPE=file SIZE=50 NAME=uploadfile>
  </FORM>
  ```

  online hypertext link

  NOTE: This capability has only been tested against Netscape Navigator versions 2 and 3. VMS Netscape Navigator 3.0b5 hangs if an upload of a variable-record format file is attempted. Stick to STREAM-LF or fixed, or convert the file to STREAM-LF.

- **text/html**
  text/plain

  Text file created according to the path, VMS record type is STREAM-LF.

- **image/gif**
  application/octet-stream
  etc., etc., etc.

  Any other MIME type is considered *binary* and the created file is made an UNDEFINED record type.

Directory Creation

A directory will be created by the HTTPd if a directory path is provided with the POST or PUT methods. For example:

```
/dir1/dir2/dir-to-be-created/
```

File Deletion

A file will be deleted by the HTTPd if the file path ending with a wildard version specification is provided with the POST or PUT methods. For example:

```
/dir1/dir2/file-to-be.deleted;*
```

Directory Deletion

A directory will be deleted by the HTTPd if a directory path ending with a wildard version specification is provided with the POST or PUT methods. For example:

```
/dir1/dir2/dir-to-be-deleted/;*
```

2.5.3 DELETE

The DELETE method should delete the file or directory corresponding to the supplied path.

Chapter 3

New to WASD? Start Here!

This chapter provides a quick guide to getting your WASD package installed, configured and serving. This covers initial installation. As of v5.1 an installation DCL procedure simplifies the initial setup.

1. **Unzip Package**

   Install the files following the guidelines in Chapter 4. **Note** that more than one archive may be needed (Section 4.1).

2. **INSTALL Package**

   Server installation is performed using the INSTALL.COM DCL procedure, see Section 4.2.

   - **Build Package**

     Compile and link, or just link supplied object files to produce VMS executables for the system's version of VMS.

   - **Check Package**

     Using the installation procedure check the operation of the basic package as recommended in Section 4.4.

   - **Directory Protections**

     This section checks for correct directory protection in the HT_ROOT:[000000] tree and prompts to change it to WORLD:RE if it appears incorrect.

   - **Create Server Account**

     Using the installation procedure create the server account. For further information see Section 5.1.

   - **Disk Quota**

     Check if quotas are enabled on the target disk. Provide an ambit allocation for the server account. Review this at some stage.

   - **Copy Server Support and Configuration Files**

Using the installation procedure copy the example server support and configuration files as discussed in Section 5.2.

3. **Configure Package**

   Following the execution of the INSTALL.COM procedure the package should require only minor, further configuration.

   Initially two files may require alteration.

   1. The startup file, possibly to set the local HTTPD$GMT logical. Consider using the STARTUP_LOCAL.COM file for other site-specific requirements. Information is provided in Section 5.2.
   2. The only configuration that should require immediate attention will be to the mapping rules, see Chapter 9.

   More generally server runtime configuration involves the considerations discussed in Section 4.7 along with the following aspects:

   - Configuring the HTTP server run-time characteristics, see Chapter 6.
   - Mapping request paths to the VMS file system, and to other things such as scripts, see Chapter 9.
   - Customizing some messages (or all if non-English language environment), see Chapter 8.
   - Establishing an authentication and authorization environment, see Chapter 11.

4. **Start Server**

   Execute the startup procedure. Get your browser and connect!

5. **Find Out What's Wrong :^(**

   Of course *something* will not be right! This can happen with the initial configuration and sometimes when changing configuration. The server provides information messages in it's run-time log, look in HT_ROOT:[LOG.SERVER]

   Remember, the basic installation's integrity can always be checked as described in Chapter 4, Section 4.4. This uses the configuration files from the [EXAMPLE] directory, so provided these have not been altered the server should execute in *demonstration mode* correctly.

   Can't resolve it? See Section 4.8.

Chapter 4

Installation and Update

The WASD package and updates will always be distributed as ZIP archives.

The ZIP archive will contain brief installation instructions. Use the following command to read this and any other information provided.

```
$ UNZIP -z device:[dir]archive.ZIP
```

For complete package distributions the archive will contain the complete directory tree. Hence to install or do a complete update it is necessary to SET DEFAULT into the top-level directory of the disk the package is to be installed on.

```
$ SET DEFAULT device:[000000]
```

For updates to portions of the package only the tree below HT_ROOT:[000000] is provided, hence it is necessary to SET DEFAULT to HT_ROOT:[000000] before UNZIPping.

```
$ SET DEFAULT HT_ROOT:[000000]
```

In either case it is recommended to check the integrity of, then list the contents of, the archive before UNZIPping. This may be done as follows:

```
$ UNZIP -t device:[dir]archive.ZIP
$ UNZIP -l device:[dir]archive.ZIP
```

A complete installation/update (the former) will have the structure:

```
Archive:  HT_ROOT:[000000]HTROOT430.ZIP;1

 WASD (HFRD) VMS Hypertext Services, Copyright (c) 1996,1997 Mark G.Daniel.
 This package (all associated programs), comes with ABSOLUTELY NO WARRANTY.
 This is free software, and you are welcome to redistribute it
 under the conditions of the GNU GENERAL PUBLIC LICENSE, version 2.

  *  Complete v4.3.0 package.
  *  $ SET DEFAULT device:[000000]
  *  $ UNZIP "-V" device:[dir]HTROOT430.ZIP

 VMS file attributes saved ... use UnZip 5.n+ on OpenVMS

 Archive created 1-AUG-1997
```

```
  Length    Date    Time    Name
  ------    ----    ----    ----
       0  07-30-97  14:15   ht_root/$_read_1st/
       0  07-30-97  14:15   ht_root/aacrt060/
       0  07-30-97  14:15   ht_root/auth/
       0  07-30-97  14:15   ht_root/axp/
    5360  07-30-97  13:23   ht_root/changes.html
       0  07-30-97  14:15   ht_root/doc/
 .
 .
 .
   16896  07-30-97  09:23   ht_root/vax/wwwrkout.exe
  ------                    -------
19109754                    996 files
```

An update (the latter) will have the structure:

```
Archive:  HT_ROOT:[000000]HTROOT430_UPD_01.ZIP;1

 WASD (HFRD) VMS Hypertext Services, Copyright (c) 1996,1997 Mark G.Daniel.
 This package (all associated programs), comes with ABSOLUTELY NO WARRANTY.
 This is free software, and you are welcome to redistribute it
 under the conditions of the GNU GENERAL PUBLIC LICENSE, version 2.

  *  Mostly minor updates to scripts and utilities for the v4.3.0 package.
  *  A couple nuisance-value-bugs have also been fixed in these.
  *  SDM2HTM can now generate <FRAMESET> documents (see [DOC]*.COM)

 VMS file attributes saved ... use UnZip 5.n+ on OpenVMS

 Archive created 6-AUG-1997

 Length    Date    Time    Name
 ------    ----    ----    ----
  42957  08-04-97  07:57   src/utils/wwwrkout.c
  59664  08-04-97  07:57   src/utils/obj_axp/wwwrkout.obj
  19674  08-04-97  07:57   src/utils/obj_vax/wwwrkout.obj
  26624  08-04-97  07:57   axp/wwwrkout.exe
.
.
.
  45056  08-05-97  11:02   vax/sdm2htm.exe
    617  08-05-97  09:23   doc/all_docs_framed.com
     44  08-05-97  09:23   doc/all_docs_not_framed.com
    544  08-05-97  09:22   doc/readme.html
 ------                    -------
2773107                    90 files
```

The archive may then be UNZIPped. If this is a completely new installation a new directory tree will be created. If an update to an existing installation the tree will be created ‘‘over-the-top’’ of the existing structure.

When updating:

- Some *insurance* on the directory tree is recommended in case the update should fail or otherwise be unusable or problematic (of course, this is good advice whenever about to make major changes to anything!) This may be in the format of a regular site backup, special pre-update backup, or special pre-update ZIP archive of the directory tree. The latter two could be accomplished using commands similar to the following:

```
$ BACKUP HT_ROOT:[000000...] location:HTROOT.BCK/SAVE/VERIFY
```

```
$ ZIP "-V" location:HTROOT.ZIP device:[HT_ROOT...]*.*
$ ZIP "-T" location:HTROOT.ZIP
```

If using ZIP then ensure that a previous version of the target ZIP file does not already exist. If it does then that version is updated, a new version is not created.

- New files and directories are created.
- For existing files a new version is created (the first time this is about to occur the UNZIPper requests permission - either ‘‘A’’ for all, or ‘‘y’’ or ‘‘n’’ or a per-file basis).

4.1 Source Archive, Object Module Archives

The complete package, source code, documentation, examples, etc., is provided in a single main archive. Installation and other build procedures allow the entire package to be compiled and linked from this if prefered. This requires a later version of DEC C (preferably v5.*n* or greater). VAX C is no longer supported.

In addition, for those unable or not wishing to fully build the distribution, two other platform-specific archives are available, AXP (Alpha) and VAX, containing a complete set of object modules, allowing the package to be built via a link operation only.

If a complete build is planned then only the main archive is required. If a link-only build then an additional archive for each architecture must be UNZIPped as described above. This applies to both full installations and subsequent updates. The archives will be clearly identified with the architecture type, as illustrated in this example.

```
$ UNZIP "-V" device:[dir]archive-AXP.ZIP
$ UNZIP "-V" device:[dir]archive-VAX.ZIP
```

Note

The WASD distribution and package organisation fully supports mixed-architecture clusters (both AXP and VAX in the one cluster) as one integrated installation.

4.2 Installation DCL Procedure

The INSTALL.COM procedure assists with the first installation of WASD. It provides a *vanilla* setup, using the standard directories and account environment described in this document. All sections prompt before performing any action and default to ‘‘no’’. Read the questions carefully!

It performs the following tasks:

1. **Build Executables -** Either compile sources and link, or just link package object code to produce images for the local version of VMS.
2. **Server Quick-Check -** Executes a procedure that runs up the HTTPd in demonstration mode. Allows evaluation/checking of the basic package. See Section 4.4.
3. **Directory Protections -** This section checks for correct directory protection in the HT_ROOT:[000000] tree and prompts to change it to WORLD:RE if it appears incorrect.

4. **Create Server Account -** Creates an unprivileged account for the HTTP server to execute within. Sets permissions on files and directories allowing appropriate access for this account.
5. **Disk Quota -** Check if quotas are enabled on the target disk. Provide an ambit allocation for the server account. Review this at some stage.
6. **Server Support/Configuration Files -** Copies example HTTP server configuration and support files from the [EXAMPLE] directory to the [HTTP$SERVER] and [LOCAL] directories. No copy is made without prompting for permission.

After UNZIPping the package do the following:

```
$ SET DEFAULT device:[HT_ROOT]
$ @INSTALL
```

NOTE: The installation procedure provides a reasonable, *vanilla*, single-node set-up. If something different is required a little time and thought may be required further configuring startup, etc.

Support files to consider when customizing startup, etc. (see Section 5.2 for further detail):

- STARTUP.COM
- STARTUP_LOCAL.COM
- STARTUP_SERVER.COM

4.3 Update DCL Procedure

NOTE: When using WASD's SSL functionality **TWO consecutive updates** will need to be performed. First, the update of the basic WASD package, immediately followed by an update of the SSL components (see SSL Quick-Start).

The UPDATE.COM procedure assists with subsequent updates of WASD. It assumes a *vanilla* setup, using the standard directories and account environment described in this document. All sections prompt before performing any action and default to ''no''. Read the questions carefully!

It provides the following functions:

1. **Build Executables -** Either compile sources and link, or just link package object code to produce images for the local version of VMS.
2. **Server Quick-Check -** Executes a procedure that runs up the HTTPd in demonstration mode. Allows evaluation/checking of the basic package. See Section 4.4.
3. **Server Support/Configuration Files -** Copies changed example HTTP server configuration and support files from the [EXAMPLE] directory to the [HTTP$SERVER] and [LOCAL] directories. This copy is made without prompting for permission.
4. **Post-Update Cleanup -** Prompts for permission to execute the post-update procedure described below.

5. **Purge Files -** Prompts for permission to purge the entire HT_ROOT:[000000...] tree.

Of course it is best (read mandatory) for the server to be shut down during an update!

After UNZIPping the updated package do the following:

```
$ SET DEFAULT HT_ROOT:[000000]
$ @UPDATE
```

Post-Update

When confident regarding the success of the update (these activities can be part of the update DCL procedure).

- The procedure UPDATEAFTER.COM should be executed. This will remove files known to exist from previous versions and relocated or no longer required.

  ```
  $ SET DEFAULT HT_ROOT:[000000]
  $ @UPDATEAFTER
  ```

- The directory tree should be purged.

4.4 Quick-Check

Once installed or updated it is possible to check the basic package at any time using the FREEWARE_DEMO.COM procedure. Follow the displayed instructions.

```
@device:[HT_ROOT]FREEWARE_DEMO
```

Problem? SYSTEM-F-SHRIDMISMAT? See next section!

4.5 Re-Linking

After an update to the operating system the package may refuse to start, reporting a message like:

```
%DCL-W-ACTIMAGE, error activating image WHAT$EVER
 -CLI-E-IMGNAME, image file DKA0:[SYS0.SYSCOMMON.][SYSLIB]WHAT$EVER_SHR.EXE
 -SYSTEM-F-SHRIDMISMAT, ident mismatch with shareable image
```

This implies the executables require re-linking for your particular version of VMS. This can be accomplished quite simply, perform the linking section only of the update DCL procedure, Section 4.3.

4.6 Local Setup Suggestions

Package updates **will never** contain anything in these three directories:

- HT_ROOT:[HTTP$SERVER]
- HT_ROOT:[LOCAL]
- HT_ROOT:[SCRIPT_LOCAL]

To prevent the overwriting of local configuration files it is suggested these be placed in the HT_ROOT:[LOCAL] directory. Startup files will most probably be placed where-ever the local site manages system startup. These could be placed in the HT_ROOT:[LOCAL] directory if there is nowhere more appropriate. Authentication databases could also be placed in the [LOCAL] directory. The server account's support files (LOGIN.COM, etc.) may be copied from the HT_ROOT:[EXAMPLE] directory into HT_ROOT:[HTTP$SERVER] and customized as necessary.

In like manner, local DCL scripts or DCL *wrappers* for executable scripts should be placed in HT_ROOT:[SCRIPT_LOCAL]. Executables should continue to be placed in HT_EXE:

Changes that are made in package configuration files, etc., can then be propagated to the local areas as appropriate.

4.7 Organizing Documents

It is recommended that the server distribution tree and any document and other Web-specific data areas be kept separate and distinct. The former in HT_ROOT:[000000], the latter in something like **WEB:[000000]**. This logical device could be provided with the following DCL introduced into the server startup procedures:

```
$ DEFINE /SYSTEM /TRANSLATION=CONCEALED WEB DSA811:[WEB.]
```

The logical organisation of served data is largely hierarchical, organised under the Web-server path root, and is achieved via two mechanisms.

1. The natural tree structure provided by a hierarchical file system.
2. The logical hierarchy possible using rules within the mapping file to place disparate physical areas into a single logical structure (see Chapter 9).

WASD has a single section of the file system for the core Web files, such as the Division home page, help areas, Web documentation, etc., it can be accessed using the logical device WEB:[000000]. Incorporated with this are other subdirectories providing storage for specific collections of data, such as committee minutes, administration announcements, etc.

Physically distinct areas are used for good physical reasons (e.g. the area can best be hosted on a task-local disk), for historical reasons (e.g. the area existed before any Web environment existed) or for reasons of convenience (e.g. lets put this where access controls already allow the maintainers to manage it).

Reasons for an area being physically integrated with the core Web data area can be legitimate (e.g. there is really nowhere else it reasonably belongs), convenience (e.g. let's quickly put it here) or by logical necessity (it really does below as part of the core Web enviroment, e.g. documentation on HTML, etc.)

Guidelines

In general, only Web-related files need to go into the core physical Web file area. All other groupings should, if possible, be decentralised into the portion of the file system they represent and logically placed in the server's path using rules in the mapping file. That is, a given project's Web files should be located in the project's part of the file system. If it doesn't have any then it may be a candidate for location in the core area.

When locating a Web area in a physically distinct area it is possible the maintainers of that data will already have the correct access controls. If locating an area in the core hypertext environment it will be necessary to give the mantainer ownership of the directory area (and possibly disk quota), or provide ACL access if multiple maintains are involved.

When locating Web-accessable data in a physically distinct area it will be necessary to update the mapping file with a new rule (see Chapter 9). If located within the core Web area the rules do not need to be adjusted.

4.8 Reporting Problems

This package, as is generally the case with freeware, is mainly developed and supported outside of the author's main occupation and working hours. Reports of problems and bugs (while not necessarily welcome :^), as well as general queries, are responded to as soon a practicable. If the documentation is inaccurate or could benefit from clarification in some area please advise of this also (the better the documentation the less queries you have to field personally . . . or so the theory goes).

With all reports please include the version of the server or script, and the hardware platform, operating system and TCP/IP package and version in use.

If a server error message is being generated please examine the HTML source of the error page. The ''<META...>'' information contains version information as well as valuable source code module and line information. Include this with the report.

If the server is exiting with a server-generated error message this information also contains module and line information. Please include this with the report.

The WATCH facility (see Chapter 15) is often a powerful adjunct in problem investigation. It is also very useful when supplying details during problem resolution. **When supplying WATCH output as part of a problem report** please ZIP the file and include it an an e-mail attachment. Mailers often mangle the report format making it difficult to interpret.

Image crash dumps may also be generated, although these are of less value than the case of the previous two.

Reports may be e-mailed to
Mark.Daniel@vsm.com.au

Should the above address present problems or provide no response for an extended period then use
Mark.Daniel@dsto.defence.gov.au

Chapter 5

Server Account and Environment

The HTTP server account should be a standard account, preferably in a group of its own (definitely at least a non-system, non-user group), with sufficient quotas to handle the expected traffic.

5.1 VMS Account

NOTE: Server process quotas must be sufficient to support the expected traffic load. In particlular PRCLM must support expected script usage. BYTLM, BIOLM, DIOL, FILLM and PGFLQUO are all significant considerations. Symptoms of insufficient process quotas include:

- Textual pages OK, but pages with a significant number of images having some or all ‘‘broken’’.
- Scripts failing mysteriously, particularly when multiple in use concurrently.
- Server and associated scripts all apparently waiting MWAIT or RWAST states.

A general rule is more is better, after all, it will only use as much as it needs!

The following provides a guide to the account:

```
Username: HTTP$SERVER                      Owner:  HyperText Daemon
Account:  HTTPD                            UIC:    [377,377] ([HTTPD,HTTP$SERVER])
CLI:      DCL                              Tables: DCLTABLES
Default:  HT_ROOT:[HTTP$SERVER]
LGICMD:   LOGIN
Flags:  Restricted DisNewMail
Primary days:   Mon Tue Wed Thu Fri
Secondary days:                     Sat Sun
Primary   000000000011111111112222  Secondary 000000000011111111112222
Day Hours 012345678901234567890123  Day Hours 012345678901234567890123
Network:  ##### Full access ######            ##### Full access ######
Batch:    ##### Full access ######            ##### Full access ######
Local:    -----  No access  ------            -----  No access  ------
Dialup:   -----  No access  ------            -----  No access  ------
Remote:   -----  No access  ------            -----  No access  ------
Expiration:            (none)    Pwdminimum:  6   Login Fails:     0
Pwdlifetime:         90 00:00    Pwdchange:      (pre-expired)
Last Login:            (none) (interactive), 11-MAY-1995 08:44 (non-interactive)
Maxjobs:         0  Fillm:       300  Bytlm:       300000
Maxacctjobs:     0  Shrfillm:      0  Pbytlm:            0
Maxdetach:       0  BIOlm:       512  JTquota:       1024
Prclm:         100  DIOlm:       512  WSdef:         1000
Prio:            4  ASTlm:       600  WSquo:         2000
Queprio:         0  TQElm:       100  WSextent:     20000
CPU:        (none)  Enqlm:       256  Pgflquo:     200000
Authorized Privileges:
  NETMBX    TMPMBX
Default Privileges:
  NETMBX    TMPMBX
```

5.2 Account Support Files

NOTE: Support procedures often change between versions. It is always advisable to check the versions documentation before installing or updating.

Examples may be found in HT_ROOT:[EXAMPLE].

online hypertext link

HTTPd Executables

Two server executables can be built by the package.

- **HTTPD.EXE -** basic server
- **HTTPD_SSL.EXE -** SSL-enabled server (see Chapter 13).

Privileged Image

As this image is to be installed with privileges unauthorized use should be prevented by applying an ACL similar to the following against the executable image:

```
$ SET SECURITY HT_EXE:HTTPD.EXE -
  /ACL=((IDENT=HTTP$SERVER,ACCESS=R+E),(IDENT=*,ACCESS=NONE))
```

This can be done once, at installation, or for peace-of-mind *(a.k.a. VMS-ish paranoia)* at each server startup.

As the HTTP$SERVER account should be completely unprivileged, and the HTTPd image requires NETMBX, TMPMBX, PRMMBX, PSWAPM, SYSNAM and SYSPRV privileges (see the ‘‘Nuts and Bolts’’ document for a description of how and why the server uses these privileges). It must be installed using a command similar to the following:

```
$ INSTALL = "$SYS$SYSTEM:INSTALL/COMMAND_MODE"
$ INSTALL ADD HT_EXE:HTTPD.EXE /PRIVILEGE=(ALTPRI,PRMMBX,PSWAPM,SYSPRV,SYSNAM)
```

Logical Names

The following logical names are essential for the operation of the HTTPd server and must be defined before startup:

- **HTTPD$AUTH -** location of the authentication/authorization configuration file (defined system-wide, or in the job table if server-specific)
- **HTTPD$CONFIG -** location of the configuration file (defined system-wide, or in the job table if server-specific)
- **HTTPD$MAP -** location of the mapping rule file (defined system-wide, or in the job table if server-specific)
- **HTTPD$MSG -** location of the message file (defined system-wide, or in the job table if server-specific)
- **HTTPD$GMT -** offset from GMT (e.g. ‘‘+10:30’’, ‘‘-01:15’’). For systems supporting DTSS (e.g. DECnet-Plus) this logical may be left undefined, with server time being calculated using the SYS$TIMEZONE_DIFFERENTIAL logical.
- **HTTPD$LOG -** if logging is enabled and no log file name specified on the command line, this logical must be defined to locate the file. When a logging period is in use this logical need only contain the directory used to store the logs.
- **HTTPD$SSL_CERT -** when using the SSL executable this logical locates the default certificate.
- **HTTPD$STARTUP_SERVER . . . -** can be used to pass parameters to HTTPd startup.
- **CGI-BIN -** system logical defining a search list with the architecture-specific executable directory first, local script directory second, then the common script directory, as a concealed device.
- **HT_AUTH -** directory containing authentication/authorization databases (files)
- **HT_EXE -** directory containing the executable images
- **HT_LOGS -** optional definition, for convenient log file specification

- **HT_SERVER_LOGS -** optional definition, for convenient detached server process log file specification

The following logical name is created by the executing HTTP server and defines the name of the control mailbox:

- **HTTPD*port*$CONTROL**

The following logical names are created by the executing HTTP server if the HTTPd monitor utility is enabled:

- **HTTPD*port*$COUNT1**
- **HTTPD*port*$COUNT2**
- **HTTPD*port*$PID**
- **HTTPD*port*$REQUEST**

Server Process Logging Directory

The server process log directory (output for the detached HTTP server processes) may require explicit access controls for the HTTPd account. This can be done by applying an ACL similar to the following:

```
$ SET SECURITY HT_ROOT:[LOG]SERVER.DIR -
  /ACL=((IDENT=HTTP$SERVER,ACCESS=R+W+E, OPTIONS=DEFAULT), -
        (IDENT=HTTP$SERVER,ACCESS=R+W+E), -
        (IDENT=*,ACCESS=NONE, OPTIONS=DEFAULT), -
        (IDENT=*,ACCESS=NONE))
```

As with the ACL on the server executable this can be done once, at installation (or, if right over the top, at each server startup). Appropriate disk quotas may also need to be applied. Also see Section 5.3.2.6.

STARTUP.COM

Putting all this together the HTTP server startup procedure becomes something similar to the supplied example. It should be called from SYSTARTUP_VMS.COM or the site's equivalent.

This procedure will support simple and quite complex sites. It works closely with STARTUP_SERVER.COM (see below). It is designed to accept parameters from the command-line or as pre-assigned symbols. Operating in this fashion should mean that no modifications will need to be made to the procedure itself. Startup characteristics are essentially determined by DCL symbol values. Some symbols are booleans, switching functionality off and on, others require string values. When relevant startup values are not assigned a reasonable default will be applied. See the following examples.

Startup characteristics can be determined by supplying symbol assignment values as command-line parameters when calling the procedure.

```
$ @$1$DKA0:[HT_ROOT.LOCAL]STARTUP WASD_DECNET=1 WASD_SSL=1 -
WASD_SSL_CERTIFICATE="HT_ROOT:[LOCAL]ALPHA.PEM"
```

Startup characteristics can also be determined by assigning the symbol values before calling the procedure itself.

```
$ WASD_DECNET = 1
$ WASD_SSL = 1
$ WASD_SSL_CERTIFICATE = "HT_ROOT:[LOCAL]ALPHA.PEM"
$ @$1$DKA0:[HT_ROOT.LOCAL]STARTUP
```

Check the procedure itself for detail on symbol names and functionality.

See HT_ROOT:[EXAMPLE]STARTUP.COM

STARTUP_LOCAL.COM

This file is automatically executed by the STARTUP.COM procedure immediately before the server is actually started. It is provided to supply all the local site's additional startup requirements. Place site-specific server environment startup in here, leaving STARTUP.COM alone as much as possible.

See HT_ROOT:[EXAMPLE]STARTUP_LOCAL.COM

STARTUP_SERVER.COM

This procedure serves a dual purpose.

1. In a simple site setup it is submitted to the SYS$BATCH queue during startup. The batch portion creates a detached process, which then again uses this procedure as input, supporting the executing HTTPd.
2. For a site that wishes to support different services across multiple nodes in a cluster, or to support multiple HTTPd processes on the one system (each by implication having differing services, see Section 6.2), this procedure acts as a template for supporting these (and then is used in the same batch and detached manner described above).

   After deciding on the organisation of nodes and/or services copy this procedure, locating the files in the same directory as STARTUP.COM, and having destination file names representing the nodes and/or ports desired. Ensure any copies have read and execute permission for the HTTP$SERVER account!

   - If multiple nodes in a cluster are to provide different services then create a file for each system as with the destination name being

     ```
     STARTUP_SERVER_nodename.COM
     ```

   - If a single system is to provide multiple services using multiple processes then the target file name must also include the identifying port number (the first in the service list), as follows

     ```
     STARTUP_SERVER_nodename_port.COM
     ```

   For any given system, STARTUP.COM searches for these, based on the node name, and will execute any found, one per node and/or node-port. If none of these are found it executes the basic STARTUP_SERVER.COM, which itself should not be modified.

Backward-compatibility is provided for pre-v5.3 installations, with STARTUP.COM executing HT_ROOT:[HTTP$SERVER]HTTPD_BATCH.COM and HT_ROOT:[HTTP$SERVER]HTTPD80.COM if none of the above are found.

To pass parameters to the HTTPd executable for startup configuration (e.g. /SERVICE, /SYSUAF, etc.) two methods are available, the second is the recommended.

1. ''Hard-wire'' them by modifying the procedure's HTTPD_PARAMETERS symbol.
2. Define a system logical containing the required parameters in the site's STARTUP_LOCAL.COM procedure. The logical name should be begin ''HTTPD$'' and then be continued with the procedure's name, that is

```
HTTPD$STARTUP_SERVER
HTTPD$STARTUP_SERVER_nodename
HTTPD$STARTUP_SERVER_nodename_port
```

as in the following example

```
$ DEFINE /SYSTEM HTTPD$STARTUP_SERVER_ALPHA -
         "/SYSUAF /SERVICE=""http://alpha.wasd.dsto.defence.gov.au:8000"""
```

See HT_ROOT:[EXAMPLE]STARTUP_SERVER.COM

LOGIN.COM

This procedure provides the HTTP server account login control.

See HT_ROOT:[EXAMPLE]LOGIN.COM

5.3 HTTPd Command Line

Command-line qualifiers provide some server startup control as well as server runtime control.

5.3.1 Server Startup

When starting up the server several characteristics of the server may be specified using qualifiers on the command line. If not specified appropriate defaults are employed. For recommended methods of passing parameters to the executable at server startup see STARTUP_SERVER.COM.

- **/ACCEPT=** comma-separated list of hosts/domains allowed to connect to the server
- **/AUTHORIZATION=[SSL,ALL]** the ''SSL'' keyword causes all authentication (both SYSUAF and HTA database) to be available only via ''https:'' requests. See Chapter 13. The ''ALL'' keyword forces the server to deny access to any path that does not have authorization in place against it. See Section 11.1.
- **/CGI_PREFIX=** the prefix to the CGI symbol names created for a script (defaults to ''WWW_'', similar to the CERN VMS HTTPd, see ''Scripting Environment'' document)
- **/DO=** command to be performed by the executing server (see Section 5.3.2)
- **/FILBUF=** number of bytes in the read buffer for files open for processing (i.e. menu files, image mapping configuration files, pre-processed HTML files, etc., not direct file transfers)

- **/FORMAT=** overrides the configuration parameter [LogFormat]
- **/REJECT=** comma-separated list of hosts/domains not allowed to connect to the server
- **/[NO]LOG[=name]** either disables logging (overrides configuration directive), or enables logging and optionally specifies the log file name (also see section Logical Names, logging is disabled by default). If the file specification is ‘‘SYS$OUTPUT’’ the server issues log entries to <stdout>, allowing user-defined log formats to be easily checked and refined.
- **/NETBUF=** minimum number of bytes in the network read buffer
- **/OUTBUF=** number of bytes in the output buffer (for direct file transfers, buffered output from menu interpretation, HTML-preprocessing, etc.)
- **/PERIOD=** overrides the configuration parameter [LogPeriod]
- **/PORT=** overrides the configuration parameter [Port] BUT is in turn overridden by the [Service] configuration parameter and /SERVICE= qualifier (is really only useful for use with the /DO= qualifier).
- **/PRIORITY=** server process priority (default is 4)
- **/[NO]PROFILE** allows SYSUAF-authenticated username security profiles to be used for file access (see Section 11.9
- **/PROMISCUOUS[=password]** server will accept any authentication username/password pair (used for testing, demonstrations, etc.)
- **/SERVICE=** comma-separated, list of server services (overrides the [Service] configuration parameter)
- **/[NO]SSL[=version]** controls Secure Sockets Layer protocol behaviour. The version can be any of ‘‘2’’, ‘‘3’’ or ‘‘23’’ (i.e. both 2 and 3, and the default) specifying which SSL protocol version the server will service.
- **/SUBBUF=** number bytes in a subprocess’ SYS$OUTPUT buffer
- **/[NO]SWAP** controls whether the server process may be swapped out of the balance set (default is swapping disabled)
- **/[NO]SYSUAF[=ID,SSL,WASD]** allows or disallows (D) username authentication using the server system’s SYSUAF (see Section 11.9, the optional ‘‘SSL’’ keyword causes SYSUAF authentication to be available only via ‘‘https:’’ requests (see Chapter 13), and the optional ‘‘ID’’ keyword makes SYSUAF authentication only available to account possessing a specific identifier. See Section 11.9. The ‘‘WASD’’ keyword make the deprecated, "hard-wired" WASD identifier environment available to this server. See Section 11.9.2.
- **/VERSION** merely displays the executable’s version string and the copyright notice
- **/[NO]WATCH=** controls the use of the WATCH reporting facility. See Chapter 15 for further details.

Note: buffer sizes apply on a per-request (thread) basis, and may be tailored for specific environments at server startup.

5.3.2 Server Command Line Control

A foreign command for the HTTPD control functionality will need to be assigned in the adminstration users' LOGIN.COM, for example:

```
HTTPD == "$HT_EXE:HTTPD"
!HTTPD == "$HT_EXE:HTTPD_SSL"
```

Some control of the executing server is available from the DCL command line on the system on which it is executing. This functionality, via the /DO= qualifier, is available to the privileged user. If a non-default server port, or multiple servers on the one system are being used, then it will be necessary to provide a /PORT= qualifier with any command, and possibly issue it multiple times. Of course, these commands are available from batch jobs as well as interactively.

5.3.2.1 Accounting

Server counters may be zeroed. These counters are those visible from the *statistics* admininstration menu item and when using the HTTPDMON utility.

```
$ HTTPD /DO=ZERO
```

5.3.2.2 Authentication

See Chapter 11.

The authorization rule file (HTTP$AUTH) may be reloaded.

```
$ HTTPD /DO=AUTH=LOAD
```

The authentication cache may be purged, resulting in re-authentication for all subsequent authorization-controlled accesses. This may be useful when disabling authorization or if a user has been locked-out due to too many invalid password attempts (see Section 11.8).

```
$ HTTPD /DO=AUTH=PURGE
```

5.3.2.3 Cache

Server cache control may also be exercised from the server administration menu, see Chapter 14. The file cache (see Chapter 16) may be enabled, disabled and have it's contents purged (declared invalid and reloaded) using

```
$ HTTPD /DO=CACHE=ON
$ HTTPD /DO=CACHE=OFF
$ HTTPD /DO=CACHE=PURGE
```

5.3.2.4 DCL/Scripting Subprocesses

These commands can be useful for flushing any currently executing CGIplus applications from the server, enabling a new version to be loaded with the next access. See ''Scripting Environment'' document.

All scripting subprocesses, busy with a request or not, can be deleted (this may cause the client to lose data).

```
$ HTTPD /DO=DCL=DELETE
```

A gentler alternative is to delete idle subprocesses and mark busy ones for deletion when completed processing.

```
$ HTTPD /DO=DCL=PURGE
```

5.3.2.5 DECnet Scripting Connections

All DECnet connections, busy with a request or not, can be disconnected (this may cause the client to lose data).

```
$ HTTPD /DO=DECNET=DISCONNECT
```

Purging is a better alternative, disconnecting idle tasks and marking busy ones for disconnection when complete.

```
$ HTTPD /DO=DECNET=PURGE
```

5.3.2.6 Logging

Server logging control may also be exercised from the server administration menu, see Chapter 14.

Open a log file.

```
$ HTTPD /DO=LOG=OPEN
```

Optionally open and specify a log file name. This overrides any /LOG=*file-name* or HTTPD$LOG specified file name.

```
$ HTTPD /DO=LOG=OPEN=file-name
```

The log file may be closed and reopened using a different name (allowing continuous operation but, for example, with daily log files). This overrides any /LOG=*file-name* or HTTPD$LOG specified file name.

```
$ HTTPD /DO=LOG=REOPEN=file-name
```

Close the log file.

```
$ HTTPD /DO=LOG=CLOSE
```

Unwritten log records may be flushed to the file.

```
$ HTTPD /DO=LOG=FLUSH
```

The format and period specification of the log may be changed (only takes effect after the next log file open/reopen) using

```
$ HTTPD /DO=LOG=FORMAT=string
$ HTTPD /DO=LOG=PERIOD=string
```

5.3.2.7 Mapping

See Chapter 9.

The mapping rule file (HTTPD$MAP) may be reloaded.

```
$ HTTPD /DO=MAP
```

5.3.2.8 Shutdown and Restart

Server shutdown may also be exercised from the server administration menu, see Chapter 14.

The server may be shut down, without loss of existing client requests. Connection acceptance is stopped and any existing requests continue to be processed until conclusion.

```
$ HTTPD /DO=EXIT
```

The server may be immediately and unconditionally shut down.

```
$ HTTPD /DO=EXIT=NOW
```

The server may be restarted, without loss of existing client requests. Connection acceptance is stopped and any existing requests continue to be processed until conclusion. This effectively causes the server to exit normally and the DCL *wrapper* procedure to restart it.

```
$ HTTPD /DO=RESTART
```

This variant restarts the server immediately regardless of existing connections.

```
$ HTTPD /DO=RESTART=NOW
```

5.4 Multi-Homed Hosts With Digital TCP/IP

A multi-homed host responds to connections for more than one network address. Details on configuring Digital TCP/IP (UCX) for this behaviour may be found in the *Digital TCP/IP Management* manual, section on ‘‘Subnetwork Routing’’ (3.4.8 in the November 1995 edition) and *Digital TCP/IP Management Command Reference*, section on ‘‘SET INTERFACE’’ (November 1995 edition). The server configuration parameter [Service] allows for the processing of requests for multiple host-names and/or ports, supporting **virtual servers** (see Section 6.2) while conditional mapping rules can further enhance this basic capability (see Section 9.6).

Assuming an alternate, unused host name exists (for example):

```
$ UCX
UCX> SET HOST "second.host.name" /ADDRESS=131.185.30.2 /ALIAS=("second","SECOND")
UCX> [EXIT]
$
```

The system IP interface can be determined as follows (in this example it is SE0, the LO0 is the localhost interface):

```
UCX> SHOW INTERFACE
                                                           Packets
Interface   IP_Addr         Network mask          Receive          Send     MTU

 SE0        131.185.30.1    255.255.0.0            595567        262764    1500
 LO0        127.0.0.1       255.0.0.0                   1           573   65535  ?
UCX> SHOW INTERFACE SE0 /FULL
Interface: SE0
  IP_Addr: 131.185.30.1      NETWRK: 255.255.0.0       BRDCST: 131.185.255.255
                      Ethernet_Addr: AA-00-04-00-59-04    MTU:  1500
.
.
.
  Restarting attempts                  0
  Successful restarts                  0
```

Note the network and broadcast masks of the interface above. The basic procedure then is to create a pseudo-interface (the ‘‘A’’ added to the hardware device name) as follows:

```
UCX> SET INTERFACE SEA0 /HOST="second" /NET=255.255.0.0 /BROAD=131.185.255.255
UCX> SET PROTOCOL IP /FORWARD
```

After the above commands the interfaces should appear as shown below:

```
UCX> SHOW INTERFACE
                                                                Packets
Interface   IP_Addr         Network mask          Receive          Send     MTU

 SE0        131.185.30.1    255.255.0.0            595567        262764    1500
 SEA0       131.185.30.2    255.255.0.0               169            99    1500
 LO0        127.0.0.1       255.0.0.0                   0          2263   65535  ?
```

These commands can be done interactively, during each startup, or placed in the permanent UCX configuration database once, as follows:

```
UCX> SET CONFIG INTERFACE SEA0 /HOST="second" /NET=255.255.0.0 /BROAD=131.185.255.255
UCX> SET CONFIG PROTOCOL IP /FORWARD
```

Note: Originating Host Address

Digital TCP/IP Services v4.1 at least, appears to determine an outgoing connection’s originating host address from the IP address of the ‘‘highest’’ pseudo-interface defined (in this case from the SEA0, resulting in an apparent source address of 131.185.30.2). This can cause some confusion on remote hosts as to the origin of a connection, and otherwise be generally undesirable.

A solution appears to be to assign the primary address to the last defined interface. This means the original interface must be removed using a

```
UCX> SET CONFIG NOINTERFACE SE0
```

then explicitly redefining it and then the new interface, ensuring the first is the non-primary address and the second the primary.

Chapter 6

Server Configuration

Server configuration basically concerns itself the very fundamental behaviour of the server process. Requirements such as buffer and cache sizes, timeout values, scripting limits, content-types, icons, number and type of services offerted, etc., are determined from configuration information.

By default, the system-table logical name HTTPD$CONFIG locates a common configuration file, unless an individual configuration file is specified using a job-table logical name. Simple editing of this file changes the configuration. Comment lines may be included by prefixing them with the hash ‘‘#’’ character. Configuration file directives are not case-sensitive. Any changes to the configuration file can only be enabled by restarting the HTTPd process using the following command on the server system.

```
$ HTTPD /DO=RESTART
```

A server’s currently loaded configuration can be interrogated. See Chapter 14 for further information.

6.1 Content-Type Configuration

HTTP uses a quasi-standard implementation of the MIME (Multi-purpose Internet Mail Extensions) specification for identifying the type of data returned in a response. A MIME content-type consists of a plain text string describing the data as a *type* and slash-separated *subtype*, as illustrated in the following examples:

```
text/html
text/plain
image/gif
image/jpeg
application/octet-stream
```

The content-type is returned to the client as part of the HTTP response, the client then using this information to correctly process and present the data contained in that response.

In common with most HTTP servers WASD uses a file's type (extension, suffix, e.g. ‘‘.HTML’’, ‘‘.TXT’’, ‘‘.GIF’’) to identify the data type within the file. The [AddType] directive (see below) is used in configuration to bind a file type to a MIME content-type. To make the server recognise and return specific content-types the [AddType] directives matches file types to content-types.

NOTE: When adding a totally new content-type to the configuration be sure also to bind an icon to that type using the [AddIcon] directive (see below). When this is not done a directory listing shows ‘‘[?]’’ in place of an icon.

With the VMS file system there is no effective file characteristic or algorithm for identifying a file's content without an exhaustive examination of the data contained there-in . . . a very expensive process (and probably still inconclusive in many cases), hence the reliance on the file type.

If a file type is not recognised (i.e. no [AddType] corresponding to the file type) then by default WASD identifies its data as *application/octet-stream* (i.e. essentially binary data). Most browsers respond to this content-type with a download dialog, allowing the data to be saved as a file. Most commonly these unknown types manifest themselves when authors use ‘‘interesting’’ file names to indicate their purpose. Here are some examples the author has encountered:

```
README.VMS
README.1ST
READ-ME.FIRST
BUILD.INSTRUCTIONS
MANUAL.PT1 (.PT2,  . . .  )
```

If the site administrator would prefer another default content-type, perhaps ‘‘text/plain’’ so that any unidentified files default to plain text, then this may be configured by specifying that content-type as the *description* of the catch-all file type entry. Examples (use one of):

```
[AddType]
*  internal/x-unknown  -
*  internal/x-unknown  -  application/octet-stream
*  internal/x-unknown  -  text/plain
*  internal/x-unknown  -  something/else-entirely
```

It is the author's opinion that unidentified file types should remain as binary downloads, not ‘‘text’’ documents, which they are probably more often not, but it's there if it's wanted.

Explicitly Specifying Content-Type

When accessing files it is possible to explicitly specify the identifying content-type to be returned to the browser in the HTTP response header. Of course this does not change the actual content of the file, just the header content-type! This is primarily provided to allow access to plain-text documents that have obscure, non-‘‘standard’’ or non-configured file extensions.

It could also be used for other purposes, ‘‘forcing’’ the browser to accept a particular file as a particular content-type. This can be useful if the extension is not configured (as mentioned above) or in the case where the file contains data of a known content-type but with an extension conflicting with an already configured extension specifying data of a different content-type.

Enter the file path into the browser's URL specification field ("Location:", "Address:"). Then, for plain-text, append the following query string:

```
?httpd=content&type=text/plain
```

For another content-type substitute it appropriately. For example, to retrieve a text file in binary (why I can't imagine :^) use

```
?httpd=content&type=application/octet-stream
```

This is an example:

online demonstration

It is posssible to "force" the content-type for all files in a particular directory. Enter the path to the directory and then add

```
?httpd=index&type=text/plain
```

(or what-ever type is desired). Links to files in the listing will contain the appropriate ''?httpd=content&type=...'' appended as a query string.

This is an example:

6.2 Virtual Services

The WASD server is capable of concurrently supporting the same host name on different port numbers and a number of different host names (aliased or multi-homed) using the same port number. This capability is generally known as a *virtual server*. Virtual services offer versatile and powerful multi-site capabilities using the one system and server process. Service determination is based on the contents of the request's ''Host:'' header field. If none is present it defaults to base service for the interface's IP address and port.

The same mechanism also effectively allows a single instance of the configuration files to support multiple server processes (using the /SERVICE qualifier), either on the one system or across multiple systems, as in a cluster (i.e. supports ''virtual'' and ''real'' servers.) See STARTUP_SERVER.COM for further information on startup support for these configurations.

Using the [Service] configuration parameter or the /SERVICE qualifier the server creates an HTTP service for each specified. If the host name is omitted it defaults to the local host name. If the port is omitted it defaults to 80. The first port specified in the service list becomes the ''administration'' port of the server, using the local host name, appearing in administration reports, menus, etc. This port is also that specified when sending control commands via the /DO= qualifier (see Section 5.3.2).

This rather contrived example show a server configured to provide four services over two host names.

```
[Service]
alpha.wasd.dsto.defence.gov.au
alpha.wasd.dsto.defence.gov.au:8080
beta.wasd.dsto.defence.gov.au
beta.wasd.dsto.defence.gov.au:8000
```

WASD provides server process run-time parameters via the HTTPD$CONFIG configuration file. These provide settings for logging, scripting, timeouts, file content-type mappings, etc. The HTTPD$MSG file provides configurable system messages.

[[virtual-server]]

The essential profile of a site is established by it's mapped resources and any authorization controls, the HTTPD$MAP and HTTPD$AUTH configuration files respectively, and these two files support directives that allow configuration rules to be applied to all virtual services (i.e. a default), to a host name (all ports), or to a single specified service (host name and specific port).

To restrict rules to a specified server (virtual or real) add a line containing the server host name, and optionally a port number, between double-square brackets. All following rules will be applied only to that service. If a port number is not present it applies to all ports for that service name, otherwise only to the service using that port. To resume applying rules to all services use a single asterisk instead of a host name. In this way default (all service) and server-specific rules may be interleaved to build a composite environment, server-specific yet with defaults. Note that service-specific and service-common rules may be mixed in any order allowing common rules to be shared. This descriptive example shows a file with one rule per line.

```
# just an example
 this rule applies to all services
 so does this
 and this one
 [[alpha.wasd.dsto.defence.gov.au]]
 this one however applies only to ALPHA, but to all ports
 as indeed does this
 [[beta.wasd.dsto.defence.gov.au:8000]]
 now we switch to the BETA service, but only port 8000
 another one only applying to BETA
 and a third
 [[*]]
 now we have a couple default rules
 that again apply to all servers
```

Both the mapping and authorization modules report if rules are provided for services that are not configured for the particular server process (i.e. not in the server's [Service] or /SERVICE parameter list). This provides feedback to the site administrator about any configuration problems that exist, but may also appear if a set of rules are shared between multiple processes on a system or cluster where processes deliver differing services. In this latter case the reports can be considered informational, but should be checked initially and then occasionally for misconfiguration.

Unknown Virtual Server

If a service is not configured for the particular host address and port of a request one of two actions will be taken.

1. If the configuration directive [ServiceNotFoundURL] is set the request will be redirected to the specified URL. This should contain a specific host name, as well as message page. For the default page use:

   ```
   [ServiceNotFoundURL]  //server.host.name/httpd/-/servicenotfound.html
   ```

2. If the above directive is not set the request is mapped using the default rules (e.g. [[*]]). It is possible to specify a rule set containing a default rule for each virtual server. The unmatched request is then handled by a fallback rule, as illustrated in the following.

   ```
   pass /*/-/admin/*
   pass /*/-/* /ht_root/runtime/*/*
   exec /cgi-bin/* /cgi-bin/*
   [[virtual1.host.name]]
   /* /web/virtual1/*
   / /web/virtual1/
   [[virtual2.host.name]]
   /* /web/virtual2/*
   / /web/virtual2/
   [[virtual3.host.name]]
   /* /web/virtual3/*
   / /web/virtual3/
   [[*]]
   /* /web/servicenotfound.html
   ```

This applies to dotted-decimal addresses as well as alpha-numeric. Therefore if there is a requirement to connect via a numeric IP address such a service must have been configured.

Note also that the converse is possible. That is, it's possible to configure a service that the server cannot ever possibly respond to because it does not have an interface using the IP address represented by the service host.

6.3 Error Reporting

By default the server provides it's own internal error reporting facility. In addition the [ErrorReportPath] configuration directive allows a redirection path to be specified for error reporting, allowing the site administrator to tailor both the nature and format of the information provided. A Server Side Include document, CGI script or even standard HTML file(s) may be specified. An SSI document is recommended for it's simplicity yet versatility.

Site-specific error reporting works by internal redirection. When an error is reported the original request is concluded and the request reconstructed using the error report path before internally being reprocessed. Error information becomes available via a specially-built query string, and from that as CGI variables in the error report context. One implication is the original request path and query string are no longer available. All error information must be obtained from the error information in the new query string.

Using an SSI Document

Three example SSI error report documents are provided. See HT_ROOT:[EXAMPLE]REPORTERROR*.SHTML The first providing a report identical with those internally generated, the second a small variation on this, and the third considerably different and with much less specific error information (which some administrator's may consider advantageous).

It is suggested with any use of this facility the reporting document(s) be located somewhere local, probably HT_ROOT:[LOCAL], and then enabled by placing the appropriate path into the [ErrorReportPath] configuration directive, as shown for the examples here

```
[ErrorReportPath] /ht_root/example/reporterror1.shtml
```

Note that virtual services can subsequently have this path mapped to other documents (or even scripts) so that some or all services may have custom error reports.

The following SSI variables are available specifically for generating error reports. The <!– #printenv –> statement near the top of the file may be uncommented to view all SSI and CGI variables available.

- **ERROR_LINE -** The HTTPd source code line from where the error was generated.
- **ERROR_MODULE -** The HTTPd source code module corresponding to the line described above.
- **ERROR_REPORT -** A single HTML string providing a detailed error message.
- **ERROR_REPORT2 -** A single HTML comment providing more detailed VMS error information (if available).
- **ERROR_REPORT3 -** A server-generated HTML string providing a brief explanation of the error (if available).
- **ERROR_STATUS_CODE -** The HTTP response status code representing the error (e.g. 404).
- **ERROR_STATUS_CLASS -** Essentially the single hundreds digit from the status code (e.g. 4).
- **ERROR_STATUS_TEXT -** The human-readable statement representing the status code (e.g. ‘‘not found’’).
- **FORM_ERROR_ . . . -** A series of CGI variables providing the sources for the above SSI variables, as well as other general environment information.

Using a Script

It is also possible to report using a script. This is discouraged dues to it’s relative complexity, and should only be undertaken for verify specific reasons of limitation within the SSI implementation. The source code HT_ROOT:[SRC.MISC]REPORTERROR.C provides such an implementation example.

Using HTML Documents

SSI documents are the obvious choice for reporting errors but they do take time and CPU for processing, and this may be a significant consideration on busy sites. If prefered, static HTML documents may be used to report errors. Generally one per possible response error status code is required. When providing a error report path including a ‘‘!UL’’ introduces the response status code into the file path, providing a report path that includes a three digit number representing the error. A file for each possible number must then be provided.

```
[ErrorReportPath] /ht_root/local/reporterror!UL.html
```

This mapping will generate paths such as the following, and require at least these to respond to general errors.

```
/ht_root/local/reporterror400.html
/ht_root/local/reporterror401.html
/ht_root/local/reporterror403.html
/ht_root/local/reporterror404.html
/ht_root/local/reporterror500.html
/ht_root/local/reporterror501.html
/ht_root/local/reporterror502.html
/ht_root/local/reporterror503.html
```

6.4 Access Logging

WASD provides a versatile access log, allowing data to be collected in Web-standard *common* and *combined* formats, as well as allowing customization of the log record format. It is also possible to specify a log period. If this is done log files are automatically changed according to the period specified. Exclude requests from specified hosts using the [LogExcludeHosts] configuration parameter.

6.4.1 Log Format

The configuration parameter [LogFormat] and the server qualifier /FORMAT specifies one of three pre-defined formats, or a user-definable format. Most log analysis tools can process the three pre-defined formats. There is a small performance impost when using the user-defined format, as the log entry must be specially formatted for each request.

1. **COMMON -** This is the most common, base logging format for Web servers. COMMON is the default log format.
2. **COMMON_SERVER -** This is an optional format used, for one, by the NCSA server. It is basically the common format, with the server host name appended to the line (used for multi-homed servers, see Section 6.2).
3. **COMBINED -** This is an optional format used, for one again, by the NCSA server. It too is basically the common format, with the HTTP referer and user agent appended.

User-Defined

The user-defined format allows customised log formats to be specified using a selection of commonly required data. The specification must begin with a character that is used as a substitute when a particular field is empty (use "\0" for no substitute, as in the "windows log format" example below).

Two different "escape" characters introduce the following parameters:

A ‘‘!’’ followed by

- **AR -** authentication realm (if any)
- **AU -** authenticated user name (if any)
- **BB -** bytes in body (excludes response header)
- **BY -** bytes in response (includes header)
- **CA -** client address

- **CN -** client host name (or address if DNS lookup disabled)
- **EM -** request elapsed time in milliseconds
- **ES -** request elapsed time in fractional seconds
- **ME -** request method
- **PA -** request path (not to be confused with ‘‘RQ’’)
- **QS -** request query string (if any)
- **RF -** referer (if any)
- **RQ -** complete request string (see below)
- **RS -** response status code
- **SN -** server host name
- **SC -** script name (if any)
- **SM -** request scheme (http: or https:)
- **SP -** server port
- **TC -** request time (common log format)
- **TG -** request time (GMT)
- **TV -** request time (VMS format)
- **UA -** user agent

A ‘‘\’’ followed by

- **0 -** a null character (used to define the empty field character)
- **! -** insert an ‘‘!’’
- **\ -** insert a ‘‘\’’
- **n -** insert a newline
- **q -** insert a quote (so that in DCL the quotes won’t need escaping!)
- **t -** insert a TAB

Any other character is directly inserted into the log entry.

Note on “PA” and “RQ”

The ‘‘PA’’ and ‘‘RQ’’ have distinct roles. In general the ‘‘RQ’’ (request) directive will always be used as this is the full request string; script component (if any), path string and query string component (if any). The ‘‘PA’’ directive is merely the path string after any script and query string components have been removed.

Examples

1. The equivalent of the common log format is:

   ```
   -!CN - !AU [!TC] \q!RQ\q !RS !BY
   ```

2. The combined log format could be specified as:

   ```
   -!CN - !AU [!TC] \q!RQ\q !RS !BY \q!RF\q \q!UA\q
   ```

3. The *O'Reilly WebSite* ''windows log format'' would be created by:

   ```
   \0!TC\t!CA\t!SN\t!AR\t!AU\t!ME\t!PA\t!RQ\t!EM\t!UA\t!RS\t!BB\t
   ```

4. The common log format with appended request duration in seconds could be provided using:

   ```
   -!CN - !AU [!TC] \q!RQ\q !RS !BY !ES
   ```

6.4.2 Log Period

The access log file may have a period specified against it, producing an automatic generation of log file based on that period. This allows logs to be systematically named, ordered and kept to a managable size. The period specified can be one of

- DAILY
- weekly as . . .
  MONDAY
  TUESDAY
  WEDNESDAY
  THURSDAY
  FRIDAY
  SATURDAY
  SUNDAY
- MONTHLY

The log file changes on the first request after midnight of the new period. When using a weekly period the new log file comes into effect on the first request following midnight on the specified day.

When using a periodic log file, the file name specified by HTTPD$LOG or the configuration parameter [LogFileName] is partially ignored, only partially because the directory component of it is used to located the generated file name. The periodic log file name generated comprises

- server host name
- server port
- year (YYYY)
- month (MM)
- day (DD)

as in the following example

```
HT_LOGS:WASD_80_19971013_ACCESS.LOG
```

For the daily period the date represents the request date. For the weekly period it is the date of the previous (or current) day specified. That is, if the request occurs on the Wednesday for a weekly period specified by Monday the log date show the last Monday's. For the monthly period it uses the first.

6.4.3 Log Per-Service

By default a single access log file is created for each HTTP server process. Using the [LogPerService] configuration directive a log file for each service provided by the HTTPd is generated (see Section 6.2).

6.4.4 Log Naming

When per-period or per-service logging is enabled the access log file has a name generated. Part of this name is the host's name or IP address. By default the host name is used, however if the host IP address is specified the dot-numeric address is used, hyphens being substituted for the periods. Accepted values for the [LogNaming] configuration directive are:

- NAME (default)
- ADDRESS

Examples of generated per-service (non-per-period) log names:

```
HT_LOGS:WASD_80_ACCESS.LOG
HT_LOGS:131-185-250-202_80_ACCESS.LOG
```

Examples of generated per-period (with/without per-service) log names:

```
HT_LOGS:WASD_80_19971013_ACCESS.LOG
HT_LOGS:131-185-250-202_80_19971013_ACCESS.LOG
```

online demonstration

Chapter 7

Server Configuration Directives

NOTE: The format of the configuration file has been changed between version 3 and version 4. This was to support on-line server configuration and administration (see Chapter 14). The version 3 format will still be read by the HTTPd but the first time the configuration is updated on-line it is saved in the version 4 format, described below.

The example configuration file for the WASD HTTP server can be viewed.

online hypertext link

Some directives take a single parameter, such as an integer, string or boolean value. Other directives can/must have multiple parameters. The version 4 configuration requires the directive to be placed on a line by itself and each separate parameter on a separate line following it. All parameter lines apply to the most recently encountered directive.

Note that all *boolean* directives are *disabled* (OFF) by default. This is done so that there can be no confusion about what is enabled and disabled by default. To use directive controlled facility it **must** be explicitly enabled.

7.1 Functional Groupings

Authentication/Authorization

[AuthBasic]	enable BASIC method
[AuthCacheMinutes]	minutes before explicitly reauthorizing user from sources
[AuthDigest]	enable DIGEST method
[AuthDigestGetLife]	DIGEST method GET lifetime
[AuthDigestPutLife]	DIGEST method PUT lifetime
[AuthFailureLimit]	retries allowed before username is marked as intruder
[AuthRevalidateUserMinutes]	minutes before use needs to reenter password

Buffer Sizes

[BufferSizeDclCgiPlusIn]	number of bytes allocated to scripting subprocess CGIPLUSIN mailbox
[BufferSizeDclCommand]	bytes allocated to scripting subprocess SYS$COMMAND mailbox
[BufferSizeDclOutput]	bytes allocated to scripting subprocess SYS$OUTPUT mailbox
[BufferSizeNetRead]	bytes allocated to client request read buffer, adn to the scripting subprocess SYS$INPUT mailbox
[BufferSizeNetWrite]	bytes allocated to client output buffer

Directory Listing

[AddIcon]	path to icon for a specified content-type
[AddBlankIcon]	path to blank icon
[AddDirIcon]	path to directory icon
[AddParentIcon]	path to parent icon
[AddUnknownIcon]	path to icon for unknown content-type
[DirAccess]	enable and form of listing
[DirDescriptionLines]	number of HTML file lines searched for document title
[DirLayout]	layout of the various listing components
[DirNoImpliedWildcard]	do not add wildcards to request if not present in path
[DirNoPrivIgnore]	ignore, do not report, privilege violations on files/directories
[DirOwner]	allow owner of file to be included in layout directive
[DirPreExpired]	pre-expire listing responses
[DirReadMeFile]	specify read-me files
[DirWildcard]	allow wildcards to be specified at all

Error Reporting

[ErrorReportPath]	path to script, SSI or ‘‘flat’’ error document
[ErrorRecommend]	for server generated error include probable cause
[ErrorSourceInfo]	for server generated error include module and line reporting it

File Cache

[CacheChunkKBytes]	memory block allocation size
[CacheEntriesMax]	maximum number of files allowed in cache
[CacheFileKBytesMax]	maximum size of a file
[CacheFrequentHits]	identify active files
[CacheFrequentSeconds]	identify active file
[CacheHashTableEntries]	size of hash table
[CacheTotalKBytesMax]	maximum memory to be consumed by cache
[CacheValidateSeconds]	maximum period before the cache checks for file modification

Logging

[Logging]	enable logging
[LogExcludeHosts]	hosts to be excluded from log
[LogFormat]	nature and layout of log contents
[LogNaming]	how the log name is be constructed
[LogPeriod]	period at which new logs are created
[LogPerService]	create a separate log for each configured service

Miscellaneous

[Accept]	restrictive list of host from which to accept requests
[ActivityDays]	activity graph duration
[AddType]	add a content-type
[Busy]	number of concurrent connections allowed
[CommentedInfo]	add server information to directory listings, etc.
[CharsetDefault]	default character set for text responses
[DNSLookup]	enable host lookup
[Monitor]	enable HTTPDMON data logicals
[Port]	default port
[PutMaxKBytes]	maximum size of a POST or PUT
[PutVersionLimit]	maximum RMS file versions retained in a POST or PUT
[Reject]	proscriptive list of hosts from which request will be rejected
[RequestHistory]	number of requests kept for request report

[SearchScript]	path to default search script
[Service]	list of host names and/or port to create services for
[ServiceNotFoundURL]	redirection URL when a request service is not configured
[StreamLF]	enable and set maximum size of automatic Stream-LF conversion
[Welcome]	list of file names that are checked for as home pages

Proxy Serving

[ProxyCache]	enable proxy caching
[ProxyCacheFileKBytesMax]	maximum size of response for caching
[ProxyCacheRoutineHourOfDay]	hour of day routine cache purge occurs
[ProxyCacheDeviceCheckMinutes]	minutes between check of cache device usage
[ProxyCacheDeviceMaxPercent]	maximum percentage of cache device used before purge
[ProxyCacheDevicePurgePercent]	during purge reduce by this many percent
[ProxyCachePurgeList]	list of file ages used during purge
[ProxyCacheReloadList]	list of file ages before realod from source
[ProxyForwardedBy]	add ‘‘Forwarded-by’’ to requests
[ProxyHostCachePurgeHours]	period at which DNS information is purged
[ProxyHostLookupRetryCount]	DNS resolution retry count
[ProxyReportLog]	report failures to process log
[ProxyReportCacheLog]	report cache failures to process log
[ProxyServing]	enable proxy server

Request Timeout

[TimeoutInput]	minutes a connection can wait before sending request
[TimeoutKeepAlive]	seconds a connection is kept active after request conclusion
[TimeoutOutput]	minutes a response can continue to output
[TimeoutNoProgress]	minutes a response can continue without data transfer progress

Scripting

[DclCgiPlusLifeTime]	minutes of non-use before CGIplus subprocess is deleted
[DclFullRequest]	(backward compatibility only)

[DclHardLimit]	maximum number of concurrent subprocesses
[DclScriptRunTime]	script execution environment
[DclSoftLimit]	maximum number of subprocesses before proactive deletion begins
[DclSpawnAuthPriv]	spawn subprocesses with account’s authorized privileges
[DclZombieLifeTime]	minutes of non-use before a CGI/CLI subprocess is deleted
[DecnetReuseLifeTime]	minutes of non-use before a DECnet process is released
[DecnetConnectListMax]	maximum number of DECnet processes

Server Side Includes

[SSI]	enable Server Side Includes
[SSIaccesses]	allow access counting
[SSIexec]	allow DCL commands

7.2 Alphabetic Listing

1. **[Accept]** ***host/domain name*** *(default: all)*

   One or more (comma-separated if on the same line) internet host/domain names, with ‘‘*’’ wildcarding for host/subdomain matching, to be explicitly allowed access. If DNS lookup is not enabled hosts must be expressed using numeric addresses (see [DNSLookup] directive). Also see the [Reject] directive. Reject directives have precedence over Accept directives. The Accept directive may be used multiple times.

   Examples:

   ```
   [Accept]
   *.wasd.dsto.defence.gov.au
   131.185.250.*
   ```

2. **[ActivityDays]** ***integer*** *(default: 0)*

   Specifies the number of days to record activity statistics, available in report form from the server administration menu (see Section 14.2). Zero disables this data collection. The maximum is 28 days. 11520 bytes per day, and 80640 per week, is required to store the per-minute data.

3. **[AddIcon]** ***icon-URL ALT-text template*** *(no default)*

   Specifies a directory listing icon and alternative text for the mime content type specified in the template.

   Examples:

   ```
   [AddIcon]
   /icon/-/doc.gif    [HTM]  text/html
   /icon/-/text.gif   [TXT]  text/plain
   /icon/-/image.gif  [IMG]  image/gif
   ```

4. **[AddBlankIcon]** ***icon-URL***
   [AddDirIcon] ***icon-URL ALT-text***
   [AddParentIcon] ***icon-URL ALT-text***
   [AddUnknownIcon] ***icon-URL ALT-text*** *(no defaults)*

   Specifies a directory listing icon for these non-content-type parts of the listing.

   Examples:

   ```
   [AddBlankIcon]    /icon/-/blank.gif
   [AddDirIcon]      /icon/-/dir.gif      [DIR]
   [AddParentIcon]   /icon/-/back.gif     [<--]
   [AddUnknownIcon]  /icon/-/unknown.gif  [???]
   ```

5. **[AddType]** ***.suffix content-type script-name description*** *(no default)*

   Binds a file suffix (extension, type) to a mime content type. The script name is used to auto-script against a specified file type. Use a hyphen as a place-holder and to indicate no auto-script. The description is used as documentation for directory listings.

   ```
   [AddType]
   .html  text/html        -       HyperText Markup Language
   .txt   text/plain       -       plain text
   .gif   image/gif        -       image (GIF)
   .hlb   text/x-script /Conan  VMS Help library
   .decw$book   text/x-script   /HyperReader    Bookreader book
   *  internal/x-unknown   -  application/octet-stream
   #*  internal/x-unknown  -  text/plain
   ```

   The content-type string may include a specific character set. In this way non-default sets (which is usually ISO-8859-1) can be specified for any particular site or any particular file type. Enclose the content-type string with double-quotation marks.

   ```
   [AddType]
   .html    "text/html; charset=ISO-8859-1"   -  HTML (ISO-8859-1)
   .html_5  "text/html; charset=ISO-8859-5"   -  Cyrillic HTML (ISO-8859-5)
   .html_r  "text/html; charset=KOI8-R"       -  Cyrillic HTML (KOI8-R)
   .txt     "text/plain; charset=ISO-8859-1"  -  plain text (ISO-8859-1)
   .txt_5   "text/plain; charset=ISO-8859-5"  -  Cyrillic text (ISO-8859-5)
   .txt_r   "text/plain; charset=KOI8-R"      -  Cyrillic text (KOI8-R)
   ```

   (Versions prior to 3.2 used this configuration directive for the MIME content-type to determine whether a file was transfered record-by-record or in binary. This is no longer required and should be removed from existing configuration files.)

6. **[AuthBasic]** ENABLED|DISABLED *(default: DISABLED)*

   Enables or disables BASIC username authentication. See Chapter 11.

7. **[AuthCacheMinutes]** ***integer*** *(default: 60)*

   The number of minutes authentication information is cached before being revalidated from the authentication source. Zero disables caching (with a resultant impact on performance as each request requiring authentication is validated directly from the source).

8. **[AuthDigest]** ENABLED|DISABLED *(default: DISABLED)*

   Enables or disables Digest username authentication. See Chapter 11.

9. **[AuthDigestGetLife]** ***integer*** *(default: 0)*

The number of seconds a digest nonce for a GET request (read) can be used before becoming stale.

10. **[AuthDigestPutLife]** ***integer*** *(default: 0)*

    The number of seconds a digest nonce for a PUT (/POST/DELETE ... write) request can be used before becoming stale.

11. **[AuthFailureLimit]** ***integer*** *(default: 10)*

    The number of unsuccessful attempts at authentication before the username is disabled. Once disabled any subsequent attempt is automatically refused without further reference to the authentication source. A disabled username can be reenabled by simply purging the cache.

12. **[AuthRevalidateUserMinutes]** ***integer*** *(default: 60)*

    The number of minutes between authenticated requests that user authentication remains valid before the user is forced to reenter the authentication information (via browser dialog). Zero disables the requirement for revalidation.

13. **[AuthVMS]**

    (Retired in v4.4, server /SYSUAF qualifier provides this)

14. **[BufferSizeDclCgiPlusIn]** ***integer*** *(default: 2048)*

    The number of bytes (and hence BYTLM quota) permanently allocated to each scripting subprocess CGIPLUSIN mailbox.

15. **[BufferSizeDclCommand]** ***integer*** *(default: 3072)*

    The number of bytes (and hence BYTLM quota) permanently allocated to each scripting subprocess SYS$COMMAND mailbox.

16. **[BufferSizeDclOutput]** ***integer*** *(default: 4096)*

    The number of bytes (and hence BYTLM quota) permanently allocated to each scripting subprocess SYS$OUTPUT mailbox.

17. **[BufferSizeNetRead]** ***integer*** *(default: 2048)*

    The number of bytes allocated to the network read buffer (used for request header, POST body, etc.). Also the number of bytes (and hence BYTLM quota) permanently allocated to each scripting subprocess SYS$INPUT mailbox (allowing a script to read a request body).

18. **[BufferSizeNetWrite]** ***integer*** *(default: 4096)*

    Number of bytes allocated to the network write buffer. This buffer is used as the basic unit when transfering file contents (from cache or the file system), as an output buffer during SSI pocessing, directory listing, etc. During many activities multiple outputs are buffered into this storage before being written to the network.

19. **[Busy]** ***integer*** *(default: 10)*

    The maximum number of concurrent client connections before a ''*server too busy right now ... try again shortly*'' error is returned to the client.

20. **[Cache]** ENABLED|DISABLED *(default: DISABLED)*

21. **[CacheChunkKBytes]** ***integer*** *(default: 0)*

    Granularity of memory blocks allocated to file data, in kilobytes.

22. **[CacheEntriesMax]** ***integer*** *(default: 0)*

    Maximum number of files loaded into the cache before entries are reused removing the original contents from the cache.

23. **[CacheFileKBytesMax]** ***integer*** *(default: 0)*

    Maximum size of a file before it is not a candidate for being cached, in kilobytes.

24. **[CacheFrequentHits]** ***integer*** *(default: 0)*

    Minimum, total number of hits an entry must sustain before being a candidate for [CacheFrequentSeconds] assessment.

25. **[CacheFrequentSeconds]** ***integer*** *(default: 0)*

    If a file has been hit at least [CacheFrequentHits] times in total and the last was within the number of seconds here specified it will not be a candidate for reuse. See Chapter 16.

26. **[CacheHashTableEntries]** ***integer*** *(default: 0)*

    Size of the hash table.

27. **[CacheTotalKBytesMax]** ***integer*** *(default: 0)*

    Maximum memory allocated to the cache, in kilobytes.

28. **[CacheValidateSeconds]** ***integer*** *(default: 0)*

    The interval after which a cache entry's original, content revision time is revalidated against the file's current revision time. If not the same the contents are declared invalid and reloaded.

29. **[CharsetDefault]** ***string*** *(default: none)*

    The default character set sent in the response header for text documents (plain and HTML). English language sites should specify ISO-8859-1, other Latin alphabet sites, ISO-8859-2, 3, etc. Cyrillic sites might wish to specify ISO-8859-5 or KOI8-R, and so on.

30. **[CommentedInfo]** ENABLED|DISABLED *(default: DISABLED)*

    Includes, as META information, the software ID of the server and any relevant VMS file specification for directory listings, etc.

31. **[DclCgiPlusLifeTime]** ***integer*** *(default: 0)*

    This value represents time in minutes. If this value is zero CGIplus subprocess may persist indefinitely (excluding explicit and proactive server purging). If non-zero the CGIplus subprocess is terminated the specified number of minutes after it last processed a request. This helps prevent sporadically used scripts from clogging up a system.

32. **[DclFullRequest]** ENABLED|DISABLED *(default: DISABLED)*

Versions of the server prior to 4.3 supplied the full request (header then body) to the script. This was not fully CGI-compliant. Versions 4.3 and following supply only the body, although the previous behaviour may be explicitly selected by enabling this configuration parameter.

33. **[DclHardLimit]** ***integer*** *(default: 0)*

    The maximum number of DCL/CGI script processing subprocesses that may ever exist concurrently (works in conjunction with [DclSoftLimit].

34. **[DclScriptRunTime]** ***string*** *(default: none)*

    One or more file type (extension) specification and scripting verb pairs. See ''Scripting Overview, Runtime''.

35. **[DclSoftLimit]** ***integer*** *(default: 0)*

    The number of DCL/CGI script processing subprocesses after which idle subprocesses are deleted to make room for new ones. The [DclHardLimit] should be approximately 25% more than the [DclSoftLimit]. The margin exists to allow for occasional slow run-down of deleted/finishing subprocesses. If these limits are not set (i.e. zero) they are calculated with [Busy] using ''[DclSoftLimit] = [Busy]'' and ''[DclHardLimit] = [DclSoftLimit] + [DclSoftLimit] / 4''.

36. **[DclSpawnAuthPriv]** ENABLED|DISABLED *(default: DISABLED)*

    By default, when a DCL/scripting subprocess is spawned it inherits the server's currently enabled privileges, which are **none**, not even TMPMBX or NETMBX. If this parameter is enabled the subprocess is created with the server account's SYSUAF-authorized privileges (which should never be other than NETMBX and TMPMBX). Use with caution.

37. **[DclZombieLifeTime]** ***integer*** *(default: 0)*

    This value represents time in minutes. If this value is zero the use of persistant DCL subprocesses is disabled. If non-zero the *zombie* subprocess is terminated the specified number of minutes after it last processed a request. This helps prevent zombie processes from clogging up a system. See ''Scripting Environment'' document.

38. **[DECnetReuseLifeTime]** ***integer*** *(default: 0)*

    The number of minutes a DECnet scripting connection is maintained with the network task. Zero disables connection reuse.

39. **[DECnetConnectListMax]** ***integer*** *(default: 0)*

    The size of the list used to manage connections for DECnet scripting. Zero effectively allows the server to use as many DECnet scripting connections as demanded.

40. **[DirAccess]** ON|OFF|SELECTIVE *(default: DISABLED)*

    Controls directory listings. SELECTIVE allows access only to those directories containing a file .WWW_BROWSABLE. The WASD HTTPd directory access facility always ignores directories containing a file named .WWW_HIDDEN. Also see the [DirWildcard] directive.

41. **[DirDescription]** ON|OFF *(default: DISABLED)*

    Retired in v5.0, replaced by [DirDescriptionLines]

42. **[DirDescriptionLines]** ***integer*** *(default: 0)*

Non-Zero enables HTML file descriptions during listings. Generating HTML descriptions involves opening each HTML file and searching for <TITLE>...</TITLE> and <H1>...</H1> text to generate the description. This is an obviously resource-intensive activity and on busy servers or systems may be disabled. Any non-zero number specifies the number of lines to be searched before quitting. Set to a very high number to search all of files' contents (e.g. 999999).

43. **[DirLayout]** ***string*** *(default: I__L__R__S__D)*

Allows specification of the directory listing layout. This is a short, case-insensitive string that specifies the included fields, relative placement and optionally the width of the fields in a directory listing. Each field is controlled by a single letter and optional leading decimal number specifying its width. If a width is not specified an appropriate default applies. An underscore is used to indicate a single space and is used to separate the fields (two consecutive works well).

- **C** - creation date
- **D** - description (generally best specified last)
  - **D:L** - for files, make a link out of the description text
- **I** - icon (takes no field-width attribute)
- **L** - link (highlighted anchor using the name of the file)
- **N** - name (no link, why bother? who knows!)
- **O** - owner (can be disabled)
- **R** - revision date
- **S** - size
  - **S:B** - in bytes (comma-formatted)
  - **S:D** - decimal kilos (see below)
  - **S:F** - kilo and mega are displayed to one decimal place
  - **S:K** - in kilo-bytes (and fractions thereof)
  - **S:M** - in mega-bytes (and fractions thereof)
- **U** - upper-case file and directory names (must be the first character)

The following shows some examples:

```
[DirLayout]       I__L__R__S__D
[DirLayout]       I__L__R__S:b__D
[DirLayout]       I__15L__S__D
[DirLayout]       UI__15L__S__D
[DirLayout]       15L__9R__S
[DirLayout]       15N_9C_9R_S
[DirLayout]       I__L__R__S:d__D
[DirLayout]       25D:l__S:b__C__R
```

The size of files is displayed by default as 1024 byte kilos. When using the ‘‘S:k’’, ‘‘S:m’’ and ‘‘S:f’’ size modifiers the size is displayed as 1000 byte kilos. If it is prefered to have the default display in 1000 byte kilos then set the directory listing layout using:

```
[DirLayout]       I__L__R__S:d__D
```

If unsure of the kilo value being used check the ‘‘<META>’’ information in the directory listing.

44. **[DirNoImpliedWildcard]** ON|OFF *(default: DISABLED)*

    When a directory is accessed having no file or type component and there is no welcome page available a directory listing is generated. By default any other directory accessed from this listing has the implied wildcards "*.*" added, consequently forcing directory listings. If enabled, this directive ensures no wildcards are added, so subsequent directories accessed with welcome pages display the pages, not a forced listing.

45. **[DirNoPrivIgnore]** ON|OFF *(default: DISABLED)*

    To prevent browsing through directories (perhaps due to inadvertant mapping) that have file permissions allowing no WORLD access the server stops listing and reports the error the first time a protection violation occurs. This behaviour may be changed to ignore the violation, listing only those files to which it has access.

46. **[DirOwner]** ON|OFF *(default: DISABLED)*

    Allows specification and display of the RMS file owner information.

47. **[DirPreExpired]** ENABLED|DISABLED *(default: DISABLED)*

    Directory listings and trees may be *pre-expired*. That is, the listing is reloaded each time the page is referenced. This is convenient in some environments where directory contents change frequently, but adds considerable over-head and so is disabled by default. Individual directory listings may have the default behaviour over-ridden using syntax similar to the following examples:

    ```
    /dir1/dir2/*.*?httpd=index?expired=yes
    /dir1/dir2/*.*?httpd=index?expired=no
    /tree/dir2/?httpd=index?expired=yes
    /tree/dir1/dir2/?httpd=index?expired=no
    ```

48. **[DirReadme]** TOP|BOTTOM | OFF *(default: DISABLED)*

    If any of the files provided using the [DirReadMeFile] directive are located in the directory the contents are included at the top or bottom of the listing (or not at all). Plain-text are included as plain-text, HTML are included as HTML allowing markup tags to be employed.

49. **[DirReadMeFile]** FILE.SUFFIX *(no default)*

    Specifies the names and order in which a directory is checked for *read-me* files. This can be enabled or disabled using the [DirReadme] directive. Plain-text are included as plain-text, HTML are included as HTML allowing markup tags to be employed.

Examples:

```
[DirReadMeFile]
readme.html
readme.htm
readme.
readme.txt
readme.1st
```

50. **[DirWildcard]** OFF|ON *(default: DISABLED)*

    This enables the facility to *force* the server to provide a directory listing by providing a wildcard file specification, even if there is a home (welcome) document in the directory. This should not be confused with the [DirAccess] directive which controls directory listing itself.

51. **[DNSLookup]** ENABLED|DISABLED *(default: DISABLED)*

    Enables or disables connection request host name resolution. This functionality may be expensive (in terms of processing overhead) and make serving granularity coarser if DNS is involved. If not enabled and logging is, the entry is logged against the numeric internet address. If not enabled any [Accept], [Reject] directive, etc., must be expressed as numeric addresses.

    This functionality significantly slows request processing. See Chapter 17.

52. **[ErrorReportPath]** ***string*** *(default: none)*

    Specifies the **URL-format path** to an optional, error reporting SSI document or script. See Section 6.3. This path can subsequently be remapped during request processing.

53. **[ErrorRecommend]** ENABLED|DISABLED *(default: DISABLED)*

    Provides a short message recommending action when reporting an error to a client. For example, if a document cannot be found it may say:

    ```
    (document, or bookmark, requires revision)
    ```

54. **[ErrorSourceInfo]** ENABLED|DISABLED *(default: DISABLED)*

    When an error message is generated META information is included showing the server version with source code module and line reporting the error. This is useful information during development and often during general use. Only disable if this concerns you.

55. **[ErrorInfo]**

    (Retired in v4.4, message configuration provides this)

56. **[ErrorSysAdmin]**

    (Retired in v4.4, message configuration provides this)

57. **[Logging]** ENABLED|DISABLED *(default: DISABLED)*

    Enables or disables the request log. Logging can slow down request processing and adds overhead. The log file name must be specified using the /LOG qualifier or HTTPD$LOG logical name (see Logical Names.

58. **[LogExcludeHosts]** ***string*** *(default: none)*

One or more (comma-separated if on the same line) internet host/domain names, with ‘‘*’’ wildcarding for host/subdomain matching, requests from which are not placed in any log files. If DNS lookup is not enabled hosts must be expressed using numeric addresses (see [DNSLookup] directive). Use for excluding local or web-maintainer’s host from logs.

Example:

```
[LogExcludeHosts]
*.wasd.dsto.defence.gov.au
131.185.250.*
```

59. **[LogFormat]** ***string*** *(default: COMMON)*

    Specifies one of three pre-defined formats, or a user-definable format. See Section 6.4.1.

60. **[LogNaming]** ***string*** *(default: none)*

    When [LogPeriod] or [LogPerService] directives are used to generate multiple log files this directive may be used to modify the naming of the file. See Section 6.4.4.

61. **[LogPeriod]** ***string*** *(default: none)*

    Specifies a period at which the log file is changed. See Section 6.4.2.

62. **[LogPerService]** ENABLED|DISABLED *(default: DISABLED)*

    When multiple services are specified (see Section 6.2) a separate log file will be created for each if this is enabled. See Section 6.4.3.

63. **[Monitor]** ENABLED|DISABLED *(default: DISABLED)*

    Allows monitoring via the HTTPDMON utility (see Section 19.6. Adds slight request processing overhead.

64. **[Port]** ***integer*** *(default: 80)*

    IP port number for server to bind to. For anything other than a command-line server control (see Section 5.3.2) this parameter is overridden by anything supplied via the [Service] directive.

65. **[ProxyForwardedBy]** ENABLED|DISABLED *(default: DISABLED)*

    Enables and disables the addition of a proxy request header line providing information that the request has been forwarded by another agent. The added header line would look like ‘‘Forwarded: by http://host.name.domain (HTTPd-WASD/n.n.n OpenVMS/AXP Digital-TCPIP SSL)’’.

66. **[ProxyHostLookupRetryCount]** ***integer*** *(default: 0)*

    When the server is resolving the name of a remote host the request may timeout due to up-stream DNS server latencies. This parameter allows a number of retries, at five second intervals, to be enabled.

67. **[ProxyReportLog]** ENABLED|DISABLED *(default: DISABLED)*

    Enables or disables the server process log reporting siginificant proxy processing events, such as cache maintenance activity.

68. **[ProxyReportCacheLog]** ENABLED|DISABLED *(default: DISABLED)*

Enables or disables the server process log reporting of proxy caching activity.

69. **[ProxyServing]** ENABLED|DISABLED *(default: DISABLED)*

    Enables or disables proxy serving on a whole-of-server basis, irrespective of any proxy services that might be configured.

70. **[ProxyCache]** ENABLED|DISABLED *(default: DISABLED)*

    Enables or disables proxy caching on a whole-of-server basis, irrespective of any proxy services that might be configured for caching.

71. **[ProxyCacheFileKBytesMax]** ***integer*** *(default: 256)*

    Maximum size of a cache file in kilobytes before it will not be cached.

72. **[ProxyCacheRoutineHourOfDay]** ***integer*** *(default: 0)*

    Hour of day for *routine* cache purge (00-23).

73. **[ProxyCacheDeviceCheckMinutes]** ***integer*** *(default: 15)*

    Interval in minutes between checking space availablility on cache device. If space is not available a *reactive* purge is initiated.

74. **[ProxyCacheDeviceMaxPercent]** ***integer*** *(default: 85)*

    The maximum percentage in use on the cache device before a *reactive* purge is scheduled. If device usage exceeds this limit no more cache files are created.

75. **[ProxyCacheDevicePurgePercent]** ***integer*** *(default: 1)*

    The percentage by which the cache device usage is attempted to be reduced when a *reactive* purge is initiated.

76. **[ProxyCachePurgeList]** ***string*** *(default: 168,48,24,8,0)*

    A list of comma-separated integers representing the sequence of last accessed period in hours used during a progressive *reactive* purge.

77. **[ProxyCacheReloadList]** ***string*** *(default: 1,2,4,8,12,24,48,96,168)*

    A list of comma-separated integers representing the sequence of age in hours used when determining whether a cache file's contents should be reloaded.

78. **[ProxyHostCachePurgeHours]** ***integer*** *(default: 24)*

    The period at which the cache of host names to IP addresses is purged.

79. **[PutMaxKBytes]** ***integer*** *(default: 250)*

    Maximum size of an HTTP POST or PUT method request in Kilobytes.

80. **[PutVersionLimit]** ***integer*** *(default: 3)*

    File created using the POST or PUT methods have the specified version limit applied.

81. **[Reject]** ***host/domain name*** *(default: none)*

One or more (comma-separated if on the same line) internet host/domain names, with ‘‘*’’ wildcarding for host/subdomain matching, to be explicitly denied access. If DNS lookup is not enabled hosts must be expressed using numeric addresses (see [DNSLookup] directive). Also see the [Accept] directive. Reject directives have precedence of Accept directives. The Reject directive may be used multiple times.

Example:

```
[Reject]
*.wasd.dsto.defence.gov.au
131.185.250.*
```

82. **[RequestHistory]** ***integer*** *(default: 0)*

    The server can keep a list of the most recent requests accessable from the server administration menu. This value determines the number kept. Zero disables the facility. Each retained request consumes 256 bytes and adds a small amount of extra processing overhead.

83. **[SearchScript]** ***path*** *(no default)*

    Specifies the **URL-format path** to the default query-string keyword search script. This path can subsequently be remapped during request processing.

    Examples:

    ```
    Search /ht_root/script/query
    ```

84. **[Service]** ***string*** *(no default)*

    This parameter allows SSL, multi-homed hosts and multiple port serving to be specified, see Section 13.1 and Section 6.2.

85. **[ServiceNotFoundURL]** ***string*** *(no default)*

    Provides a default path for reporting a virtual host does not exist, see Unknown Virtual Server.

86. **[SSI]** ENABLED|DISABLED *(default: DISABLED)*

    Enables or disables Server Side Includes (HTML pre-processing).

87. **[SSIaccesses]** ENABLED|DISABLED *(default: DISABLED)*

    Enables or disables Server Side Includes (HTML pre-processing) file access counter.

88. **[SSIexec]** ENABLED|DISABLED *(default: DISABLED)*

    Enables or disables Server Side Includes (HTML pre-processing) DCL execution functionality.

89. **[StreamLF]** ***integer*** *(default: 0 (disabled))*

    Enables or disables automatic conversion of VARIABLE record format documents (files) to STREAM-LF, which are much more efficient with this server. The integer is the maximum size of a file in kilobytes that the server will attempt to convert. Zero disables any conversions. See File Record Format.

90. **[StreamLFpaths]** ***string*** *(no default)*

(Retired in v5.3, mapping SET rule provides this now, see Section 9.2.5).

91. **[TimeoutInput]** ***integer*** *(default: 1)*

    Number of minutes to allow a connection request to be in progress without submitting a complete request header before terminating it.

92. **[TimeoutKeepAlive]** ***integer*** *(default: 5)*

    Number of seconds a ‘‘Keep-Alive:’’ request connection is maintained after the conclusion of a request. *Keep-Alive* improves the overall performance of the server by reducing the number of discrete TCP/IP connections that need to be establiched.

93. **[TimeoutNoProgress]** ***integer*** *(default: 2)*

    Number of minutes to allow request output to continue without any increase in the number of bytes transfered. This directive is targeted at identifying and eliminating requests that have stalled.

94. **[TimeoutOutput]** ***integer*** *(default: 10)*

    Number of minutes to allow a request to be output before terminating it. This directive sets an absolute maximum time a request can continue to receive output.

95. **[Welcome]** ***file.suffix*** *(no default)*

    Specifies the names and order in which a directory is checked for home page files. If no home page is found a directory listing is generated.

    Examples:

    ```
     [Welcome]
    home.html
    home.htm
    home.menu
    home.mnu
    ```

Chapter 8

Message Configuration

Message configuration is provided for two purposes.

1. Some sites would prefer to customize or extend the basic information provided to clients when an error or other event occurs.
2. Sites that do not use English as a first language may wish to provide some or all of the defined messages using a prefered language.

Not all messages provided by the WASD server are customizable, only those generated for non-administrative content. As the WASD server can also report using information derived from the standard VMS message service (via *sys$getmsg( )*) it is assumed a language-local implementation of this is in use as well. Unfortunately for the non-first-language-English Web and system administrators, the menus and messages used for administration purposes, etc., are still only in English. The intent of this facility is to provide non-administration clients only with a more familiar language environment.

Also note that the message database only applies to messages generated by the server, not to any generated by scripts, etc.

8.1 Behaviour

When an error, or other message or string, needs to be provided for the client the message database is accesssed using the following algorithm.

1. If the client request has specified a list of prefered languages using the ‘‘Accept-Language:’’ HTTP header field the message database is checked for support of that/those languages. If one is found then that language is used to access the message.
2. If none is found, or the client has not specified a prefered language, the client host address is checked against any list of hosts/domains provided against the language (see below). If a match occurs the specified language is used.
3. If neither of the above results in a message language the base language is used (the highest numbered language). This **must** have a complete set of messages or the server will not start!

8.2 Message File Format

By default, the system-table logical name HTTPD$MSG locates a common message file, unless an individual message file is specified using a job-table logical name. Simple editing of the message file changes the messages (after a server restart, of course). Comment lines may be included by prefixing them with the hash character (‘‘#’’), and lines continued by ensuring the last character is a backslash (‘‘\’’). The server will concurrently support an additional 3 languages to the base English (although this can be increased by recompilation :^)

Care must be taken with the message file or the server may refuse to start

As illustrated below the message file comprises a series of sections. Directives enclosed by square-brackets provide information to the message loader.

```
# this is a comment

[version] 4.4
[language] 1 se *.se
[language] 2 de *.de,*.su
[language] 3 en

[general]

en 01 Sanity check failure.
de 01 (the same message but in German, which I cannot speak :^)
se 01 (the same message but in Swedish, which I also cannot speak :^)
en 02 String overflow.
de 02 (the same message but in German, which I cannot speak :^)
se 01 (the same message but in Swedish, \
which I still cannot speak :^) \
this has also been continued over two additional lines!
en 03 Heap allocation failed.
en 04 calloc() failed
en 05 Request calloc() failed.
en 06 Server too busy.
en 07 Server access denied.
en 08 Facility is disabled.
en 09 Wildcard not permitted.
en 10 Directory layout problem.

[next-section, etc.]
```

The square-bracketed section headings have the following functions.

1. **[version] -** Ensures the correct database version is available for the server version attempting to use it. The message file always needs checking for this version number being changed at server updates, although the version may remain fixed at a previous server version number if there have been no changes to the message database during subsequent server versions. This must be the first directive in the file.
2. **[language] -** Creates space for assigning the new language’s messages. The number specifies an order within the languages, each must be different, but only the lowest and highest (prefered and base respectively) have operational significance. The highest number should always be English to provide a fall-back message. A short string provides an identifier for the language. This identifier should be the same as the identifying string in the browser request ‘‘Accept-Language:’’ header field (e.g. ‘‘en’’, ‘‘se’’, ‘‘de’’, ‘‘fr’’, etc.) Following the language identifier is an optional host/domain list. Multiple hosts/domains may be specified by separating each with a comma. The specifications

may contain wildcards. All the [language] directives should be grouped at the start of the file immediately following the [version] directive. Setting the language's ordering number to zero disables the language completely. All messages associated with it will then be ignored.

3. **[group-name] -** The messages are divided into groupings to make them easier to manage. Each group begins with the group name directive.
4. **en 01 message -** Each message in a group is assigned using using this format. The string identifying the language, then the message number (the leading zero just improves the format, strictly it is not required), then the actual message itself. The message can be of arbitrary length. Long messages may be continued on following lines using the ''\'' continuation character.

The base language (the highest numbered, which should always be English) must have precisely the right number of messages required by the server, too few or too many and the server will not start! **Additional languages do not have to reassign every message!** The base language will supply any not assigned. A message number of zero is disabled and completely ignored.

If messages contain HTML tags that markup must not interfere with the general HTML page it is used within.

Some messages are a composite of multiple strings each of which is used on a different part of the one page (e.g. for the [upd] edit-page). Each of the strings is delimited by the vertical bar ''|''. Care must be taken when customizing these strings that the overall number stays the same and that the length of each does not become excessive. Although it will not disrupt the server it may significantly disrupt the page layout.

All message numbers must be included. To provide an empty string for any one message (not recommended) provide the line with nothing following the message number.

8.3 Supplied Message Files

Any non-English message files that are provided to the author will be included for general use (please take the time to support this endeavour) in the HT_ROOT:[EXAMPLE] directory.

online hypertext link

Note that message files can become out-of-date as server versions change, requiring modifications to the message database. Check the version information and/or comments at the top of candidate message files, however even slightly dated files may serve as a good starting point for a locale-specific message base.

Chapter 9

Mapping Rules

Mapping rules are used in five primary ways.

1. To map a request *path* onto the VMS file system.
2. To process a request path according to specified criteria resulting in an effective path that is different to that supplied with the request.
3. To identify requests requiring script activation and to parse the script from the path portion of that request. The path portion is then independently re-mapped.
4. To conditionally map to different end-results based on one or more criteria of the request.
5. To provide differing virtual sites depending on the actual service accessed by the client.

Mapping is basically for server-internal purposes only. The only time the path information of the request itself is modified is when a script component is removed. At all other times the path information remains unchanged.

Hence, path authorization is always applied to the path supplied with the request, not to the results of any rule mapping that may have occured! This means authorization paths may be administered without any ambiguity introduced by any rule mapping that may occur.

By default, the system-table logical name HTTPD$MAP locates a common mapping rule file, unless an individual rule file is specified using a job- table logical name. Simple editing of the mapping file changes the rules. Comment lines may be included by prefixing them with the hash ‘‘#’’ character. Although, there is no fixed limit on the number of rules there are the processing implications of scanning a large, linear database.

Rules are given a basic consistency check when loaded (i.e. server startup, map reload, etc.) If there is an obvious problem (unknown rule, missing component, etc., path not absolute) a warning message is generated and the rule is not loaded into the database. This will not cause the server startup to fail. These warning messages may be found in the server process log.

The server administration facility allows realm and arbitrary paths to be checked against the rule database in real-time using the WATCH facility. See Chapter 15. In this way the rule database may be checked against test or even live requests.

Any changes to the mapping file may be (re)loaded into the running HTTPd server using the following command on the server system:

```
$ HTTPD /DO=MAP
```

Also see Chapter 6 for daemon configuration.

Server Mapping Rules

A server's currently loaded mapping rules may be interrogated. See Chapter 14 for further information.

9.1 VMS File System Specifications

The VMS file system in mapping rules is always assumed to begin with a device or concealed device logical. Specifying a Master File Directory (MFD) component, the [000000] is completely optional, although always implied. The mapping functions will always insert one if required for correct file system syntax. That is, if the VMS file system mapping of a path results in a file in a top-level directory an MFD is inserted if not explicitly present in the mapping. For example, both of the following paths

```
/dka100/example.txt
/dka100/000000/example.txt
```

would result in a mapping to

```
DKA100:[000000]EXAMPLE.TXT
```

The MFD is completely optional when both specifying paths in mapping rules and when supplying paths in a request. Similarly, when supplying a path that includes directory components, as in

```
/dka100/dir1/dir2/example.txt
/dka100/000000/dir1/dir2/example.txt
```

both mapping to

```
DKA100:[DIR1.DIR2]EXAMPLE.TXT
```

Implication: When using logical names in file system mappings they must be able to be used as concealed devices and cannot be logical equivalents of directory specifications. Concealed device logicals are created using the following syntax:

```
$ DEFINE LOGICAL_NAME device:[dir1.dir2.]
$ DEFINE LOGICAL_NAME /TRANSLATION=CONCEALED physical_device:[dir1.dir2.]
```

When during rule mapping of a path to a VMS file specification an RMS-invalid character (e.g. ''+'') or syntax (e.g. multiple periods) is encountered a dollar symbol is substituted in an attempt to make it acceptable. This functionality is often useful for document collections imported to the local web originating from, for instance, a Unix site that utilizes non-RMS file system syntax. The default substitution character may be changed on a per-path basis using the SET rule, Section 9.2.5.

9.2 Rules

There are seven mapping rules.

- Rules that map paths to the file system, and to other paths:
  - MAP
  - PASS
  - FAIL
  - REDIRECT
  - USER
- Rules that provide access to scripting:
  - EXEC
  - SCRIPT
- A rule that sets characteristics against particular paths:
  - SET

9.2.1 MAP, PASS, FAIL Rules

1. **map *template result***

   If the URL path matches the template, substitute the *result* string for the path and use that for further rule processing. Both template and result paths must be absolute (i.e. begin with ''/'').

2. **pass *template***
   pass *template result*
   pass *template ''999 message text''*

   If the URL path matches the template, substitute the result if present (if not just use the original URL path), processing no further rules.

   The *result* should be a either a physical VMS file system specification in URL format or an *HTTP status-code message* (see below). If there is a direct correspondance between the *template* and *result* the result may be omitted.

   The PASS directive is also used to *reverse-map* VMS file specifications to the URL path format equivalent.

   An HTTP status-code message can be provided as a result. The server then generates a response corresponding to that status code containing the supplied message. Status-code results should be enclosed in one of single or double quotes, or curly braces. See examples. A 3*nn* status results in a redirection response with the message text comprising the location. Codes 4*nn* and 5*nn* result in an error message. Other code ranges (e.g. 0, 1*nn*, 2*nn*, etc.) simply cause the connection to be immediately dropped, and can be used for that purpose (i.e. no indication of why!)

3. **fail *template***

If the URL path matches the template, prohibit access, processing no further rules. The template path must be absolute (i.e. begin with ‘‘/’’).

9.2.2 REDIRECT Rule

1. **redirect *template result***

   If the URL path matches the template, substitute the *result* string for the path. Process no further rules. Redirection rules can provide result URLs in four formats, each with a slightly different behaviour.

   1. The *result* can be a full URL (‘‘http://host.domain/path/to/whatever’’). This is used to redirect requests to a specific service, usually on a another host.

   2. If the scheme (e.g. ‘‘http:’’) is omitted the scheme of the current request is substituted. This allows HTTP requests to be transparently redirected via HTTP and HTTPS (SSL) requests via HTTPS (e.g. ‘‘//host.domain/path/to/whatever’’, note the leading double-slash).

   3. In a similar fashion both the scheme and the host name may be omitted (e.g. ‘‘///path/to/whatever’’, note the leading triple-slash). The server then substitutes the appropriate request scheme and host name before returning the redirection to the client.

   4. Alternatively, it may be just a path (‘‘/path/to/whatever’’, a single leading slash), which will cause the server to internally generate an entire new request structure to process the new path (i.e. request redirection is not returned to the client).

9.2.3 USER Rule

The USER rule maps a VMS user account default device and directory (i.e. *home* directory) into a request path. That is, the base location for the request is obtained from the VMS systems SYSUAF file. This is usually invoked by a request path in the form ‘‘/~username/’’, see Section 9.7 for more detailed information.

1. **user *template result***

   If the path matches the template then the result is substituted, with the following conditions. At least one wildcard must be present. The first wildcard in the result substitutes the username’s home directory into the path (in place of the ‘‘~username’’). Any subsequent wildcard(s) substitute corresponding part(s) of the original path.

   If the user DANIEL’s default device and directory were

   ```
   USER$DISK:[DANIEL]
   ```

   the following rule

```
user /~*/* /*/www/*
```

would result in the following path being mapped and used

```
/user$disk/daniel/www/
```

Note

Accounts that possess SYSPRV, are CAPTIVE, have been DISUSERED or that have expired passwords will not be mapped. A ‘‘directory not found’’ error report is returned.

9.2.4 EXEC and SCRIPT, Script Mapping Rules

Also see ‘‘Scripting Environment’’ document for further information.

The EXEC and SCRIPT directives have the **variants EXEC+ and SCRIPT+**. These behave in exactly the same fashion and simply mark the rule as representing a CGIplus script environment.

The EXEC rules maps script **directories**.

The SCRIPT rules maps script **file names**. It behaves a little differently to the EXEC rule, essentially supplying in a single rule the effect of a MAP then an EXEC rule.

Both rules must have a *template* and *result*, and both must end in a wildcard asterisk. The placement of the wildcards and the subsequent functionality is slightly different however. Both template and result paths must be absolute (i.e. begin with ‘‘/’’).

1. **exec *template result***

   The EXEC rule requires the *template*’s asterisk to immediately follow the slash terminating the directory specification containing the scripts. The script name follows immediately as part of the wildcard-matched string. For example:

   ```
   exec /htbin/* /ht_root/script/*
   ```

   If the URL path matches the template, the result, including the first slash-terminated part of the wildcard-matched section, becomes the URL format physical VMS file specification the script to be executed. What remains of the original URL path is used to create the path information. Process no further rules.

   Hence, the EXEC rule will match multiple script specifications without further rules, the script name being supplied with the URL path. Hence any script (i.e. procedure, executable) in the specified directory is accessable, a possible security concern if script management is distributed.

2. **script *template result***

   The SCRIPT rule requires the *template*’s asterisk to immediately follow the *unique string* identifying the script in the URL path. The wildcard-matched string is the following path, and supplied to the script. For example:

   ```
   script /conan* /ht_root/script/conan*
   ```

   If the URL path matches the template, the result becomes the URL format physical VMS file specification for the DCL procedure of the script to be executed (the default file extension of ‘‘.COM’’ is not required). What remains of the original URL path is used to create the path information. Process no further rules.

Note

> The wildcard asterisk is best located immediately after the unique script identifier. In this way there does not need to be any path supplied with the script. If even a slash follows the script identifier it may be mapped into a file specification that may or may not be meaningful to the script.

Hence, the SCRIPT rule will match only the script specified in the *result*, making for finely-granular scripting at the expense of a rule for each script thus specified. It also implies that only the script name need precede any other path information.

It may be thought of as a more efficient implementation of the equivalent functionlity using two CERN rules, as illustrated in the following example:

```
map /conan* /script/conan*
exec /cgi-bin/* /cgi-bin/*
```

Script Location

It is conventional to locate script images in HT_ROOT:[AXP] or HT_ROOT:[VAX] (depending on the platform), and procedures, etc. in HT_ROOT:[SCRIPT] or HT_ROOT:[SCRIPT_LOCAL]. These multiple directories are accessable via the single search list logical CGI-BIN.

Script files can be located elsewhere, either subdirectories of HT_ROOT:[SCRIPT_LOCAL] or in an area completely outside of the HT_ROOT tree. Two approaches are available.

1. Modify the search list CGI-BIN to include the additional directories.
2. Use mapping rules to make the script accessable. This can be done by using the EXEC or SCRIPT rule to specify the directory directly as in these examples

   ```
   exec /mycgi-bin/* /site_local_scripts/bin/*
   script /myscript* /web/myscripts/bin/myscript.exe*
   ```

   or by using the MAP rules to make a hierarchy of script locations obvious and accessable, as in this example

   ```
   map /cgi-bin/myscripts/* /cgi-bin_myscripts/*
   exec /cgi-bin_myscripts/* /web/myscripts/bin/*
   ```

9.2.5 SET Rule

The SET rule does not change the mapping of a path, it just sets one or more characteristics against that path that affect the subsequent processing in some way. It is a general purpose rule that conveniently allows the administrator to tell the server to process requests with particular paths in some ad hoc and generally useful fashion. Most SET parameters are single keywords that act as boolean switches on the request, some require parameter strings.

1. **AUTHONCE, NOAUTHONCE -** If a request path contains both a script component and a resource component by default the WASD server makes sure both parts are authorized before allowing access (see Chapter 11). This can be disabled using this path setting. When this is done only the original request path undergoes authorization. ‘‘authONCE’’.

2. **CACHE, NOCACHE -** The default is to cache files (when caching is enabled). NOCACHE sets files in this path as not to be stored in the file cache. A subsequent CACHE will again permit caching, allowing smaller path hierarchies to be cached within larger ones that are not.
3. **CGIPREFIX -** CGI environment variable names are by default prefixed with ‘‘WWW_ ’’. This may be changed on a per-path basis using this SET rule. To remove the prefix altogether for selected scripts use ‘‘CGIprefix=’’.
4. **CHARSET -** This setting allows overriding of the server default ([CharsetDefault] configuration parameter) content-type character set (in the response header) for text files (plain and HTML). A string is required as in the following example, ‘‘charset=ISO-8859-5’’.
5. **CONTENT -** The content-type of a file is normally determined by the file’s type (extension). This setting allows files matching the template to be returned with the specified content-type. The content-type must be specified as a parameter, e.g. ‘‘content=application/binary’’.
6. **EXPIRED, NOEXPIRED -** This setting allows files in the specified paths to be sent pre-expired. The browser should always then reload them whenever accessed.
7. **INDEX -** This setting provides the ‘‘Index of’’ (directory listing) format string for directory paths matching the template. It uses the same formatting as can be supplied with a URL and overrides any query string passed via any URL.
8. **LOG, NOLOG -** When server access logging is enabled the default is to log all requests. The NOLOG setting suppresses logging for requests involving the specified path template.
9. **PROFILE, NOPROFILE -** When using the server /PROFILE qualifier enable or disable the authentication profile when assessing access for a specific path.
10. **RMSCHAR -** During rule mapping of a path to a VMS file specification, if an RMS-invalid character (e.g. ‘‘+’’) or syntax (e.g. multiple periods) is encountered a dollar symbol is substituted in an attempt to make it acceptable. This setting provides an alternate substitution character. Any general RMS-valid character may be specified (e.g. alpha-numeric, ’$’, ’-’ or ’_’, although the latter three are probably the only REAL choices). A single character is required as in the following example, ‘‘RMSchar=_’’.
11. **STMLF, NOSTMLF -** Specify files to be automatically converted to Stream-LF format. The default is to ignore conversion. STMLF allows selected paths to be converted. See File Record Format.

Of course, as with all mapping rules, paths containing file types (extensions) may be specified so it is quite easy to apply SETtings to particular groups of files. Multiple settings may be made against the one path, merely separate set directives from each other with white-space. If a setting string is required to contain white-space enclose the string with single or double quotes, or curly brackets. The following example gives a small selection of potential uses.

```
# examples of SET rule usage
# --------------------------
# disable caching for selected paths
set /ht_root/src/* NOcache
set /sys$common/* NOcache
# enable stream-LF conversion in selected directory trees
set /web/* stmlf
set /ht_root/* stmlf
# respond with Cyrillic character set(s) from relevant directories
set /*/8859-5/* charset=ISO-8859-5
set /*/koi8-r/* charset=KOI8-R
# the Sun Java tutorial when UNZIPped contains underscores for invalid characters
set /vms/java/tutorial/* RMSchar=_
# if a request has "/plain-text/" in it's path then ALWAYS return as plain-text!
set /*/plain-text/* content=text/plain
map /*/plain-text/* /*/*
# same for "/binary/"
set /*/binary/* content=text/plain
map /*/binary/* /*/*
```

9.3 Rule Interpretation

The rules are scanned from first towards last, until a PASS, EXEC, SCRIPT, FAIL or REDIRECT is encountered, when processing ceases and final substitution occurs. Mapped rules substitute the *template* with the *result* and continue to the next rule.

Use of wildcards in *template* and *result*:

- The **template** may contain one or more asterisk (‘‘*’’) wildcard symbols. These match zero or more characters up until the character following the wildcard (or end-of-string). If no wildcard is present then the path must match the *template* exactly.
- The **result** may contain one or more asterisk (‘‘*’’) wildcard symbols. It must not contain more wildcards than the *template*. The *result* wildcards are expanded to replace the matching characters of the respective *template* wildcards. Characters represented by wildcards in the *template* not represented by a corresponding wildcard in the *result* are ignored. Non-wildcard *result* characters are directly inserted in reconstructed path. Non-wildcard characters in the *template* are ignored. If the *result* contains no wildcards it completely replaces the URL path.

Virtual Servers

As described in Section 6.2 virtual service syntax may be used with mapping rules to selectively apply rules to one specific service. If virtual services are configured rule interpretation sees only rules common to all services and those specific to it’s own service (host address and port). In all other aspects rule interpretation applies as described above.

9.4 Mapping Examples

The example mapping rule file for the WASD HTTP server can be viewed.

online hypertext link

Example of Map Rule

The *result* string of these rules may or may not correspond to to a VMS physical file system path. Either way the resulting rule is further processed before passing or failing.

1. The following example shows a path ‘‘/web/unix/shells/c’’ being mapped to ‘‘/web/software/unix/scripts/c’’, with this being used to process further rules.

```
map /web/unix/* /web/software/unix/*
```

Examples of Pass Rule

The *result* string of these rules should correspond to to a VMS physical file path.

1. This example shows a path ‘‘/web/rts/home.html’’ being mapped to ‘‘/user$rts/web/home.html’’, and this returned as the mapped path.

```
pass /web/rts/* /user$rts/web/*
```

2. This maps a path ‘‘/icon/bhts/dir.gif’’ to ‘‘/web/icon/bhts/dir.gif’’, and this returned as the mapped path.

```
pass /icon/bhts/* /web/icon/bhts/*
```

3. This example illustrates HTTP status code mapping. Each of these does basically the same thing, just using one of the three possible delimiters according to the characters required in the message. The server generates a 403 response with has as it’s text the following message. (Also see the conditional mapping examples.)

```
pass /private/* "403 Can’t go in there!"
pass /private/* ’403 "/private/" is off-limits!’
pass /private/* {403 Can’t go into "/private/"}
```

Examples of Fail Rule

1. If a URL path ‘‘/web/private/home.html’’ is being mapped the path would immediately be failed.

```
fail /web/private/*
```

2. To ensure all access fails, other than that explicitly passed, this entry should be included the the rules.

```
fail /*
```

Examples of Exec and Script Rules

1. If a URL path ‘‘/htbin/ismap/web/example.conf’’ is being mapped the ‘‘/ht_root/script/’’ must be the URL format equivalent of the physical VMS specification for the directory locating the script DCL procedure. The ‘‘/web/example.conf’’ that followed the ‘‘/htbin/ismap’’ in the original URL becomes the translated path for the script.

```
exec /cgi-bin/* /cgi-bin/*
```

2. If a URL path ‘‘/conan/web/example.hlb’’ is being mapped the ‘‘/ht_root/script/conan’’ must be the URL format equivalent of the physical VMS specification for the DCL procedure. The ‘‘/web/example.hlb’’ that followed the ‘‘/conan/’’ in the original URL becomes the translated path for the script.

```
script /conan* /ht_root/script/conan*
```

Example of Redirect Rule

1. If a URL path ‘‘/AnotherGroup/this/that/other.html’’ is being mapped the URL would be redirected to ‘‘http://host/this/that/other.html’’

```
redirect /AnotherGroup/* http://host/group/*
```

9.5 Virtual Servers

As described in Section 6.2, virtual service syntax may be used with mapping rules to selectively apply rules to one specific service. This example provides the essentials of using this syntax. Note that service-specific and service-common rules may be mixed in any order allowing common mappings (e.g. for scripting) to be shared.

```
# a mapping rule example of virtual servers
[[alpha.wasd.dsto.defence.gov.au:80]]
# ALPHA is the only service allowing access to VMS help directory
pass /sys$common/syshlp/*
[[beta.wasd.dsto.defence.gov.au:80]]
# good stuff is only available from BETA
pass /good-stuff/*
# BETA has it’s own error report format, the others share one
pass /errorreport /ht_root/local/errorreportalpha.shtml
[[gamma.wasd.dsto.defence.gov.au:80]]
# gamma responds with documents using the Cyrillic character set
set /* charset=ISO-8859-5
[[*]]
# common file and script mappings
exec /cgi-bin/* /cgi-bin/*
exec+ /cgiplus-bin/* /cgi-bin/*
script+ /help/* /cgiplus-bin/conan/*
pass /errorreport /ht_root/local/errorreport.shtml
# now the base directories for all documents
[[alpha.wasd.dsto.defence.gov.au:80]]
/* /web/alpha/*
[[beta.wasd.dsto.defence.gov.au:80]]
/* /web/beta/*
[[gamma.wasd.dsto.defence.gov.au:80]]
/* /web/gamma/*
[[*]]
# catch-all rule (just in case :^)
pass /* /web/*
```

The server administration menu WATCH report (see Section 14.2) provides the capability to view the rule databse as well as rule mapping during actual request processing, using the WATCH facility.

9.6 Conditional Mapping

The purpose of *conditional mapping* is to **apply** rules only after certain criteria other than the initial path match are met. These criteria serve to create conditional mapping rules, and were introduced in version 4.4.

THIS OFFERS A POWERFUL TOOL TO THE SERVER ADMINISTRATOR!

Conditional mapping can be applied on the following criteria:

- authenticated remote user
- client internet address
- browser-accepted languages
- browser-accepted character sets
- browser-accepted content-types
- browser identification string
- cookie data
- host and port specified in request header
- HTTP method (GET, POST, etc.)
- proxy/gateway host(s) request forwarded by
- refering page
- request scheme (protocol . . . ‘‘http:’’ or ‘‘https:’’)
- query string
- server name
- server port

Note that path authorization is always applied to the path supplied with the request, not to the results of any rule mapping that may have occured! This means authorization paths may be administered without any ambiguity being implied by any rule mapping that may occur.

Conditionals must follow the rule and are delimited by ‘‘[’’ and ‘‘]’’. Multiple, space-separated conditions may be included within one ‘‘[...]’’. This behaves as a logical OR (i.e. the condition only needs one matched to be true). Multiple ‘‘[...]’’ conditionals may be included against a rule. These act as a logical AND (i.e. all must have at least one condition matched). If a condition begins with a ‘‘!’’ it acts as a negation operator (i.e. matched strings result in a false condition, unmatched strings in a true condition). The result of an entire conditional may also be negated by prefixing the ‘‘[’’ with a ‘‘!’’.

If a conditional, or set of conditionals, is not met the rule is completely ignored.

Matching is done by simple, case-insensitive, string comparison, using the wildcards ‘‘*’’, matching one or more characters, and ‘‘%’’, matching any single character.

White-space (spaces and TABs), wildcards and the delimiting ‘‘[’’ and ‘‘]’’, are forbidden characters and cannot be used within condition matching strings, nor can they be encoded for inclusion in any way (for simplicity and speed of processing). These characters are uncommon in the information being matched against, but if one does occur then ‘‘match’’ it using a single character wildcard (‘‘%’’).

While conditionals are powerful adjuncts to *smart* serving they do add significant overhead to rule mapping and should be used with this in mind.

Conditionals

- **ac: -** browser-accepted content types (‘‘Accept:’’ request header field)
- **al: -** browser-accepted languages (‘‘Accept-Language:’’ request header field)
- **as: -** browser-accepted character sets (‘‘Accept-Charset:’’ request header field)
- **ck: -** cookie data (‘‘Cookie:’’ request header field)
- **fo: -** request forwarded by proxy/gateway host(s) (‘‘Forwarded:’’ request header field)
- **hh: -** host and port request directed to (‘‘Host:’’ request header field)
- **ho: -** browser host internet name or address
- **me: -** request HTTP method
- **qs: -** query string
- **rf: -** refering page (‘‘Referer:’’ request header field)
- **ru: -** authenticated remote user name
- **sc: -** request *scheme* (protocol), ‘‘http’’, and if SSL is in use ‘‘https’’ (see Chapter 13)
- **sn: -** server name
- **sp: -** server port
- **ua: -** browser (‘‘User-Agent:’’ request header field)

Examples

NOTE: It is possible to *spoof* (impersonate) internet host addresses. Therefore any controls applied using host name/address information cannot be used for authorization purposes in the strictest sense of the term.

1. The following example shows a rule being applied only if the client host is within a particular subnet. This is being used to provide a ‘‘private’’ home page to those in the subnet while others get a ‘‘public’’ page by the second rule.

```
pass / /web/internal/ [ho:131.185.250.*]
pass / /web/
```

2. This is a similar example to the above, but showing multiple host specifications and specifically excluding one particular host using the negation operator ‘‘!’’. This could be read as *pass if ((host OR host) AND (not host)).*

   ```
   pass / /web/internal/ [ho:*.fred.com ho:*.george.com] [!ho:you.fred.com]
   pass / /web/
   ```

3. The next example shows how to prevent browsing of a particular tree except from specified host addresses.

   ```
   pass /web/internal/* /web/SorryNoAccess.html [!ho:131.185.250.*]
   pass /web/internal/*
   ```

   This could be used to prevent browsing of the server configuration files (an alternative to this sort of approach is to use the authorization file, see Chapter 11).

   ```
   pass /ht_root/local/* /web/SorryNoAccess.html [!ho:131.185.250.201]
   ```

4. This example performs much the same task as the previous one, but uses whole conditional negation to prevent browsing of a particular tree except from specified addresses (as well as using the continuation character to provide a more easily comprehended layout . . . note the trailing spaces as required). This could be read as *pass if not (host OR host OR host).*

   ```
   pass /web/internal/* /web/SorryNoAccess.html \
   ![\
   ho:131.185.250.* \
   ho:131.185.251.* \
   ho:131.185.45.1 \
   ho:ws2.wasd.dsto.gov.au\
   ]
   pass /web/internal/*
   ```

5. This example demonstrates mapping pages according to geography or language preference (it’s a bit contrived, but . . .)

   ```
   pass /doc/* /web/doc/french/* [ho:*.fr al:fr]
   pass /doc/* /web/doc/swedish/* [ho:*.se al:se]
   pass /doc/* /web/doc/english/*
   ```

6. How to exclude specific browsers from your site (how many times have we seen this!)

   ```
   # I had to pick on a well-known acronym, no offence Bill!
   pass /* /web/NoThankYou.html [ua:*MSIE*]
   ```

7. This example allows excluding certain requests from specific addresses. This could be read as *pass if ((method is POST) AND (not host)).*

   ```
   pass /* /web/NotAllowed.html [me:POST] [!ho:*.my.net]
   ```

8. The following illustrates using the server name and/or server port to conditionally map servers executing on clustered nodes using the same configuration file, or for multi-homed/multi-ported hosts. Distinct home pages are maintained for each system, and on BETA two servers execute, one on port 8000 that may only be used by those within the specified network address range.

```
pass / /web/welcome_to_Alpha.html [sn:alpha.*]
pass / /web/welcome_to_Beta.html [sn:beta.*] [sp:80]
pass /* /sorry_no_access.html [sn:beta.*] [sp:8000] [!ho:*.my.sub.net]
pass / /web/welcome_to_Beta_private.html [sn:beta.*] [sp:8000]
```

9. Each of these three do basically the same thing, just using the three possible delimiters according to the characters required in the message. The server generates a 403 response with has as it's text the following message.

```
pass /private/* "403 Can't go in there!" [!ho:my.host.name]
pass /private/* '403 "/private/" is off-limits!' [!ho:my.host.name]
pass /private/* {403 Can't go into "/private/"} [!ho:my.host.name]
```

Note that rule processing for any particular path may be checked using the WATCH facility from the server administration menu. See Chapter 15 for details.

9.7 Mapping User Directories (*tilde* character ("~"))

The convention for specifying user web areas is ''/~username/''. The basic idea is that the user's web-available file-space is mapped into the request in place of the circumflex and username.

9.7.1 Using The SYSUAF

The USER rule maps a VMS user account default device and directory (i.e. *home* directory) into a request path (see Section 9.2.3). That is, the base location for the request is obtained from the VMS systems SYSUAF file. A user's home directory information is cached, to reduce load on the authroization databases. As this information is usually quite static there is no timeout period on such information (although it may be flushed to make room for other user's). Cache contents is include in the Mapping Rules Report (see Section 14.2) and is implicitly flushed when the server's rules are reloaded (see Section 14.4).

The following is a typical usage of the rule.

```
USER  /~*/*  /*/www/*
```

Note the ''/www'' subdirectory component. It is **stongly recommended** that users never be mapped into their top-level, but into a web-specific subdirectory. This effectively ''sandboxes'' Web access to that subdirectory hierarchy, allowing the user privacy elsewhere in the home area.

To accomodate request user paths that do not incorporate a trailing delimiter after the username the following redirect may be used to cause the browser to re-request with a more appropriate path (make sure it follows the USER rule).

```
REDIRECT  /~*  ///*/www/
```

WASD also ''reverse maps'' VMS specifications into paths and so requires additional rules to provide these mappings. (Reverse mapping is required during directory listings and error reporting.) For the continuing example the following rules would be required (and in the stated order).

```
USER  /~*/*  /*/www/*
REDIRECT  /~*  ///*/www/
PASS  /~*/*  /user$disk/*/www/*
```

Where user home directories are spread over multiple devices (physical or concealed logical) a reverse-mapping rule would be required for each. Consider the following situation, where user directories are distributed across these devices (concealed logicals)

```
USER$GROUP1:
USER$GROUP2:
USER$GROUP2:
USER$OTHER:
```

This would require the following mapping rules (in the stated order).

```
USER  /~*/*  /*/www/
PASS  /~*/*  /user$group1/*/www/*
PASS  /~*/*  /user$group2/*/www/*
PASS  /~*/*  /user$group3/*/www/*
PASS  /~*/*  /user$other/*/www/*
```

Accounts with a search list as a default device (e.g. SYS$SYSROOT) present particular complications in this schema and should be avoided.

Note

> Accounts that possess SYSPRV, are CAPTIVE, have been DISUSERED or that have expired passwords will not be mapped. A ''directory not found'' error report is returned. This error was chosen to make it to make more difficult to *probe* the authorization environment, determining whether accounts exist or not.

Of course vanilla mapping rules may be used to provide for special cases. For instance, if there is requirement for a particular, privileged account to have a user mapping that could be provided as in the following (rather exagerated) example.

```
PASS  /~system/*  /sys$common/sysmgr/www/*
USER  /~*/*  /*/www/
PASS  /~*/*  /user$disk/*/www/*
```

9.7.2 Without Using The SYSUAF

The server is also able to map user directories using the same mechanisms as for any other. No reference needs to be made to the SYSUAF, user support can be accomplished via a combination of mapping rule and logical name. This approach relies on a correspondance between the username and the home directory name. Hence users are made known by the HTTPd using the name of their top-level directory. User scripts can also be supported using WASD's DECnet scripting environment.

The ''PASS'' rule provides a wildcard representation of users' directory paths. As part of this mapping a subdirectory specifically for the hypertext data should always be included. **Never** map users' top-level directories. For instance if a user's account home directory was located in the area USER$DISK:[DANIEL] the following rule would potentially allow the user DANIEL to provide web documents from the home subdirectory [.WWW] (if the user has created it) using the accompanying URL:

```
pass /~*/* /user$disk/*/www/*
```

```
http://host/~daniel/
```

It is **recommended** that a separate logical name be created for locating user directories. This helps hide the internal organisation of the file system. The following logical name definition and mapping rule illustrate this point.

```
$ DEFINE /SYSTEM /EXEC /TRANSLATION=CONCEALED WWW_USER device:[USER.]
```

```
pass /~*/* /www_user/*/www/*
```

Where users are grouped into different areas of the file system a logical search list may be defined.

```
$ DEFINE /SYSTEM /EXEC /TRANSLATION=CONCEALED -
         WWW_USER -
         DISK1:[GROUP1.], -
         DISK1:[GROUP2.], -
         DISK2:[GROUP3.], -
         DISK2:[GROUP4.]
```

```
pass /~*/* /www_user/*/www/*
```

As logical search lists have specific uses and some complications (e.g. when creating files) this is the only use for them recommended with this server, although it is specifically coded to allow for search lists in document specifications.

If only a subset of all users are to be provided with WWW publishing access either their account directories can be individually mapped (best used only with a small number) or a separate area of the file system be provided for this purpose and specifically mapped as user space.

Of course, user mapping is amenable to all other rule processing so it is a simple matter to redirect or otherwise process user paths. For instance, the published username does not need to, or need to continue to, correspond to any real user area, or the user's actual name or home area:

```
redirect /~doej/* http://a.nother.host/~doej/*
 pass /~doej/* /www/messages/deceased.html
 pass /~danielm/* /special$www$area/danielm/*
 pass /~Mark.Daniel/* /user$disk/danielm/www/*
 pass /~*/* /www_user/*/www/*
```

A user directory is always presented as a top-level directory (i.e. no parent directory is shown), although any subdirectory tree is accesssable by default.

Chapter 10

Authorization Quick Guide

WASD offers a comprehensive and versatile authentication and authorization environment. A little too comprehensive, often leaving the new administrator wondering where to begin. The role of this chapter is to provide a starting place, especially for sources of authentication, along with some basic configurations. Chapter 11 contains a detailed explanation of all aspects. All examples here assume a standard installation and environment.

Just to clarify. **Authentication** is the verification of a user's identity, usually through username/password credentials. **Authorization** is allowing a certain action to be applied to a particular path based on that identity.

10.1 Identifier Authentication

This setup allows any active account to authenticate using the local VMS username and password. By default not every account may authenticate this way, only those holding specified VMS rights identifiers. See Section 11.9.1.

- Define the following logical before calling the server startup procedure. This logical contains a startup qualifier that configures the server to allow authentication from the SYSUAF. See Section 11.1.

  ```
  $ DEFINE /SYSTEM HTTPD$SERVER_STARTUP "/SYSUAF=ID"
  $ @device:[HT_ROOT.LOCAL]STARTUP.COM
  ```

- Decide on an identifier name. This can be an existing identifier, or one created for the purpose. An identifier can be created using the following actions.

  ```
  $ SET DEFAULT SYS$SYSTEM
  $ MCR AUTHORIZE
  UAF> ADD /IDENTIFIER identifier_name
  ```

- Modify the authorization configuration file, accessed by the server using the system logical HTTPD$AUTH, to contain the following. This allows full access to the Administration Menu and [.LOCAL] directory (and no world access). Additional paths may be added as required, and of course multiple identifiers may be created and used for multiple realms and paths.

```
["Whatever you want to call it!"=identifier_name=id]
/ht_root/local/* r+w
/httpd/-/admin/* r+w
```

- The identifier must then be granted to those accounts to be allowed to authenticate in this way.

```
UAF> GRANT /IDENTIFIER identifier_name account
```

- Using this approach useful discrimination may be exercised. For instance, one identifier for Web administrators, another (or others) for different authentication requirements. Of course the one account may hold multiple identifiers.

```
["Web Administration"=web-identifier_name=id]
/ht_root/local/* r+w
/httpd/-/admin/* r+w
["Area Access"=area-identifier_name=id]
/web/area/* r+w ; r
```

10.2 SYSUAF Authentication

If the use of identifiers is impractical for some reason, this alternative setup allows any active, non-privileged account to authenticate using the server system VMS username and password.

- Define the following logical before calling the server startup procedure. This logical configures the server to allow authentication from the SYSUAF. See Section 11.1.

```
$ DEFINE /SYSTEM HTTPD$SERVER_STARTUP "/SYSUAF"
$ @device:[HT_ROOT.LOCAL]STARTUP.COM
```

- Modify the authorization configuration file, accessed by the server using the system logical HTTPD$AUTH, to contain the following. This allows full access to the Administration Menu and [.LOCAL] directory (and no world access at all). Additional paths may be configured as required.

```
["Whatever you want to call it!"=VMS]
/web/area/* r+w
```

  Update access (read+write) may be provided to the authorized and read-only access to the world using syntax similar to the following example. This applies to all authentication sources.

```
/web/area/* r+w ; r
```

- With no discrimination on which accounts can and cannot authenticate using this setup additional controls on what can be accessed must be supplied. This is done using either restriction lists (see Access Restriction List)

  ```
  ["Realm Name"=VMS]
  /web/area1/* r+w,~username1,~username2
  /web/area2/* r+w,#host1,#host2,~username3
  ```

  or using access groups (see Section 10.4)

  ```
  ["Realm Name"=VMS;list_name1=list]
  /web/area1/* r+w ;
  ["Realm Name"=VMS;list_name2=list]
  /web/area2/* r+w ; r
  ```

Note

To also allow privileged accounts to authenticate via the SYSUAF with this configuration change the logical definition as follows.

```
$ DEFINE /SYSTEM HTTPD$SERVER_STARTUP "/SYSUAF=RELAXED"
```

This is **not a recommended configuration**.
Using VMS identifiers is a far better solution. See previous section.

10.3 Other Authentication

Other sources of authentication are available, either by themselves or used in the same configuration file (different realms and paths) as those already discussed. See Section 11.4. Non-SYSUAF sources do not require any startup qualifier to be enabled.

- **Simple lists** contain usernames and unencrypted passwords. These are plain-text files, created and modified using any desired editor.

  ```
  ["Whatever you want to call it!"=list-name=list]
  /web/area/* r+w ; r
  ```

  This is a very simple arrangement, with little inherent security. Lists are more useful when grouping names together for specifying which group may do what to where.

- **HTA databases** are WASD-specific, binary repositories of usernames, encrypted passwords, capabilities, user and other detail.

  ```
  ["Whatever you want to call it!"=HTA-database-name=HTA]
  /web/area/* r+w ; r
  ```

  These databases may be administered using the Administration Menu, are quite secure and versatile.

- **External agents** are authentication and authorization scripts executed on demand, under the control-of but external to the server. It is possible for a site to write it's own, custom authorization agent.

```
["Whatever you want to call it!"=agent-name=agent]
/web/area/* r+w ; r
```

10.4 Read and Write Groupings

WASD allows separate sources for groups of usernames to control read and write access in a particular realm. See Section 11.5. These groups may be provided via simple lists, VMS identifiers, HTA databases and authorization agents. The following example shows an identifier authenticated realm with full and read-only access controlled by two simple lists. For the first path the world has no access, for the second read-only access (with the read-only grouping becoming basically redundant information).

```
["Realm Name"=identifier_name=id;full_access_name=list;read-only_name=list]
/web/area/* r+w ;
/web/another-area/* r+w ; r
```

10.5 Considerations

Multiple authentication sources (realms) may be configured in the one HTTPD$AUTH file.

Multiple paths may be mapped against a single authentication source.

Any path may be mapped only once (for any single virtual service).

Paths may have additional access restrictions placed on them, including client host name, username, etc. See Access Restriction List.

The configuration file is loaded and stored by the server at startup. If changed it must be reloaded to take effect. This can be done manually using

```
$ HTTPD/DO=AUTH=LOAD
```

Authentication information is cached. Access subsequently removed or modified will not take effect until the entry expires, or is manually purged using

```
$ HTTPD/DO=AUTH=PURGE
```

Failed attempts to authenticate against a particular source are limited. When this is exceeded access is always denied. If this has happened the cache must be manually purged before a user can successfully authenticate

```
$ HTTPD/DO=AUTH=PURGE
```

Chapter 11

Authentication and Authorization

Authentication is the verification of a user's identity, usually through username/password credentials. **Authorization** is allowing a certain action to be applied to a particular path based on authentication of the originator.

Generally, authorization is a two step process. First authentication, using a username/password database. Second authorization, determining what the username is allowed to do for this transaction.

Authentication environments can get complex very quickly, don't forget to ''keep it simple, stupid'', see Section 11.7.1.

Overview

Server authorization is performed using a configuration file, authentication source, and optional full-access and read-only authorization grouping sources, and is based on per-path directives. There is no user-configured authorization necessary, or possible! In the configuration file paths are associated with the authentication and authorization environments, and so become subject to the HTTPd authorization mechanism. Reiterating . . . WASD HTTPd authorization administration involves those two aspects, setting authorization against paths and administering the authentication and authorization sources.

AUTHORIZATION IS ALWAYS APPLIED TO THE PATH SUPPLIED WITH THE REQUEST, not to the results of any rule mapping that may have occured! This means authorization paths may be administered without ambiguity. With requests containing a script, authorization is performed on **both** script and path components. First script access is checked for any authorization, then the path component is independently authorized. Either may result in an authorization challenge/failure. This beahviour can be disbaled using a path SETting rule, see Section 9.2.5.

Authentication can be done via the server system's **SYSUAF database** (**not recommended except except when employing SSL or in the most secure of LAN environments**), see Section 11.9 for further information.

The **authentication source** name is refered to as the *realm*, and refers to a collection of usernames and passwords. It can be the system's SYSUAF database.

The **authorization source** is refered to as the *group*, and refers to a collection of usernames and associated *permissions*.

11.1 Authentication Policy

A *policy* regarding when and how authorization can be used may be established on a per-server basis. This can restrict authentication challenges to ‘‘https:’’ (SSL) requests (see Chapter 13), thereby ensuring that the authorization environment is not compromised by use in non-encrypted transactions. Two HTTPd qualifiers provide this.

- **/AUTHORIZE=**
    - **ALL** restricts **all** requests to authorized paths. If a path does not have authorization configured against it it is automatically denied access. This is an effective method of preventing inadvertant access to areas in a site. See Section 11.11.
    - **SSL** restricts **all** authentication/authorization transactions to the SSL environment.
    - **(SSL,ALL)** combines the above two.
- **/SYSUAF=**
    - Used without any keywords, this qualifier allows all current (non-expired, non-disusered, etc.), non-privileged accounts to be used for authentication purposes.
    - **ID** restricts SYSUAF authenticated account to those possessing a specific VMS resource identifier (see Section 11.9.1).
    - **RELAXED** allows **any** current account to be authorized via the SYSUAF. **This is not recommended,** use rights identifiers to allow some discrimination to be exercised.
    - **SSL** restricts only SYSUAF authenticated transactions to the SSL environment.
    - **WASD** enables the deprecated, "hard-wired" WASD identifier environment available to this server. See Section 11.9.2.
    - **(ID,SSL)** would allow two keywords to be applied, etc.

Note also that individual paths may be restricted to SSL requests using either the mapping conditional rule configuration or the authorization configuration files. See Conditionals and Access Restriction List.

In addition, two configuration parameters have a direct role in an established authorization policy.

- **[AuthFailureLimit]** sets the maximum unsuccessful attempts allowed before a username is then always refused authentication. After this limit is reached the cache must be purged to re-allow the user access, see Section 5.3.2.2.
- **[AuthRevalidateUserMinutes]** sets the number of minutes between successive authentication attempts before the user is forced to reenter the authentication data (via a browser dialog). Zero disables this function.

Authentication Failures

Details of authentication failures are logged to the HTTPd process log.

- **%HTTPD-W-AUTHFAIL** indicates a failure to authenticate (incorrect username/password). The number of failures, the realm name, the user name and the originating host are provided. Isolated instances of this are only of moderate interest. Consecutive instances may indicate a user thrashing about for the correct password, but they usually give up before a dozen attempts.
- **%HTTPD-I-AUTHFAILOK** advises that a previous failure to authenticate has now successfully done so. This is essentially informational.
- **%HTTPD-W-AUTHFAILIM** indicates the number of failures have exceeded the [AuthFailureLimit], after which automatic refusal begins. This message should be of concern and the circumstances investigated, especially if the number of attempts becomes excessive.

11.2 Permissions, Path and User

Both paths and usernames have permissions associated with them. A path may be specified as read-only, read and write, or write-only (yes, I'm sure someone will want this!) A username may be specified as read capable, read and write capable, or only write capable. For each transaction these two are combined to determine the maximum level of access allowed. The allowed action is the logical AND of the path and username permissions.

The permissions may be described using the HTTP method names, or using the more concise abbreviations R, W, and R+W.

Path/User	DELETE	GET	HEAD	POST	PUT
READ or R	no	yes	yes	no	no
WRITE or W	yes	no	no	yes	yes
R+W	yes	yes	yes	yes	yes
DELETE	yes	yes	no	no	no
GET	no	yes	no	no	no
HEAD	no	no	yes	no	no
POST	no	no	no	yes	no
PUT	no	yes	no	no	yes

11.3 Authorization Configuration File

Requiring a particular path to be authorized in the HTTP transaction is accomplished by applying authorization requirements against that path in a configuration file. This is an activity distinct from setting up and maintaining any authentication/authorization databases required for the environment.

By default, the system-table logical name HTTPD$AUTH locates a common authorization configuration file, unless an individual rule file is specified using a job-table logical name. Simple editing of the file changes the configuration. Comment lines may be included by prefixing them with the hash ‘‘#’’ character, and lines continued by placing the backslash character ‘‘\’’ as the last character on a line.

Configuration directives begin either with a ‘‘[realm]’’, ‘‘[realm;group]’’ or ‘‘[realm;group-r+w;group-r]’’ specification, or with the forward-slash of a path specification. Following the path specification are HTTP method keywords controlling group and world permissions to the path, and any **access-restricting** request scheme (‘‘https:’’) and/or host address(es) and/or username(s).

- **REALM**

  Square brackets are used to enclose a [realm;group;group] specification, introducing a new authentication grouping. Within these brackets is specified the realm name (authentication source), and then optional group (authorization source) names separated by semi-colons. All path specifications following this are authenticated against the specified realm database, and permissions obtained from the group d‘‘[realm;group]’’ database (or authentication database if group not specified), until the next [realm;group;group] specification.

  The following shows the format of an authentication source (realm) only directive.

  ```
  [authentication-source]
  ```

  This one, the format of a directive using both authentication and authorization sources (both realm and group).

  ```
  [authentication-source ; authorization-source]
  ```

  The third variation, using an authentication, full-access (read and write) and read-only authorization sources (realm and two grouping).

  ```
  [authentication-source ; full-access-source ; read-only-source]
  ```

  The authentication source may also be given a description. This is the text the browser dialog presents during password prompting. See Realm Description in Section 11.4.

- **PATH**

  Paths are usually specified terminated with an asterisk wildcard. This implies that any directory tree below this is included in the access control. Wildcards may be used to match any portion of the specified path, or not at all. Following the path specification are control keywords representing the HTTP methods or permissions that can be applied against the path, and optional access-restricting list of host address(es) and/or username(s), separated using commas. Access control is against either or both the group and the world. The group access is specified first followed by a semi-colon separated world specification. The following show the format of the path directive, see the examples below to further clarify the format.

  ```
  /root/path/  group-access-list,group-permissions ; \
               world-access-list,world-permissions
  ```

The same path cannot be specified against two different realms. The reason lies in the HTTP authentication schema, which allows for only one realm in an authentication dialog. How would the server decide which realm to use in the authentication challenge? Of course, different parts of a given tree may have different authorizations, however any tree ending in an asterisk results in the entire sub-tree being controlled by the specified authorization environment, unless a separate specification exists for some inferior portion of the tree.

There is a thirty-one character limit on authentication source names, and a fifteen character limit on usernames and passwords. All source and user names are case-insensitive. *BASIC* authentication scheme passwords are case insensitive, however *DIGEST* authentication scheme passwords should all be either upper or lower case, mixed case will not work.

Reserved Names

The following names are reserved and have special functionality.

1. **EXTERNAL -** Any authentication and authorization will be done in some way by an external CGI script. None is attempted by the server.
2. **NONE -** This refers to any request, is not authenticated in a any way, and just marks the path as having been authorized for access. See Section 11.11.
3. **PROMISCUOUS -** This realm is only available while the /PROMISCUOUS qualifier is in use. See Chapter 14.
4. **WORLD -** This refers to any request and is not authenticated in any way, only the permissions associated with the path are applied to the request. The reserved username ‘‘WORLD’’ becomes the authenticated username.
5. **VMS -** Use the server system’s SYSUAF database to authenticate the username. For ‘‘http:’’ requests the username/password pairs are transmitted encoded but not encrypted, **this is not recommended**. For ‘‘https:’’ requests, using the implicit security offered by SSL (see Chapter 13) the use of SYSUAF authentication is considered viable.

   By default accounts with SYSPRV authorized are always rejected to discourage the use of potentially significant usernames (e.g. SYSTEM). Accounts that are disusered, have passwords that have expired, or that are captive or restricted are also automatically rejected.

   The authentication source may be disguised by giving it a specific description. This will the text the browser dialog presents during password prompting. See Realm Description in Section 11.4.

   See Section 11.9 for further information on these topics.

Reserved Username

The following username is reserved.

1. **WORLD -** If a path is authorized using the WORLD realm the pseudo-authenticated username becomes ‘‘WORLD’’. Any log will reflect this username and scripts will access a WWW_REMOTE_USER containing this value. Although not forbidden, it is not recommended this string be used as a username in other realms.

Access Restriction List

If a host name, protocol identifier or username is included in the path configuration directive it acts to **further** limit access to matching clients (path and username permissions still apply). If more than one are included a request must match each. If multiple host names and/or usernames are included the client must match at least one of each. Host and username strings may contains the asterisk wildcard, matching one or more consecutive characters. This is most useful when restricting access to all hosts within a given domain, etc.

- **Host Names -** may be specified as either alphabetic (if DNS name resolution is enabled, see [DNSlookup] configuration directive) or numeric addresses. When a host restriction occurs there is never an attempt to authenticate any associated username. Hence applying host restrictions very effectively prevents an attack from outside the allowed addresses. The reserved word *#localhost* refers to the host name the server is executing on.
- **Request Scheme -** (protocol) either ‘‘http:’’ or secured via ‘‘https:’’ (SSL)
- **User Names -** are indicated by a leading circumflex, the ‘‘~’’ character (similar or username URL syntax).

For example

```
/web/secret/* *.three.stooges,~Moe,~Larry,~Curly,read
```

restricts read access to Curly, Larry and Moe accessing from within the three.stooges network, while

```
/web/secret/* https:,*.three.stooges,~Moe,~Larry,~Curly,read
```

applies the further restriction of access via ‘‘https:’’ (SSL) only.

Note that it more efficient to place *protocol* and *host* restrictions at the front of a list.

11.4 Authorization Sources

Username authorization information may be derived from several sources, each with different characteristics.

- **VMS Rights Identifier**

  An identifier is indicated by appending a ‘‘=ID’’ to the name of the realm or group. Also refer to Section 11.9.1.

  Whether or not any particular username is allowed to authenticate via the SYSUAF may be controlled by that account holding or not holding a particular rights identifier. Placing ‘‘=ID’’ against realm name implies the username must exist in the SYSUAF and hold the specified identifier name.

  ```
  [PROJECT_A=id]
  ```

  When (and only when) a username has been authenticated via the SYSUAF, rights identifiers associated with that account may be used to control the level-of-access within that realm. This is in addition to any identifier controlling authentication itself.

  ```
  [PROJECT_A=id;PROJECT_A_LIBRARIAN=id;PROJECT_A_USER=id]
  ```

In this example a username would need to hold the PROJECT_A identifier to be able to authenticate, PROJECT_A_LIBRARIAN to write the path(s) (via POST, PUT) and PROJECT_A_USER to be able to read the path(s).

- **VMS Authentication**

  The server system SYSUAF may be used to authenticate usernames using the VMS account name and password. The realm being VMS may be indicated by using the name ‘‘VMS’’, by appending ‘‘=VMS’’ to another name making it a *VMS synonym*, or by giving it a specific description (see Realm Description in Section 11.4). Further information on SYSUAF authentication may be found in Section 11.9. These examples illustrate the general idea.

  ```
  [VMS]
  [LOCAL=vms]
  [ANY_NAME_AT_ALL=vms]
  ```

- **Simple List**

  A plain-text list may be used to provide usernames for group membership. The format is one username per line, at the start of the line, with optional, white-space delimited text continuing along the line (which could be used as documentation). Blank lines and comment lines are ignored. A line may be continued by ending it with a ‘‘\’’ character. These files may, of course, be created and maintained using any plain text editor. They must exist in the HT_AUTH: directory, have an extension of ‘‘.HTL’’, and do not need to be world accessable.

  ```
  # the stooges
  curley     Jerome Horwitz
  larry      Louis Feinberg
  moe        Moses Horwitz
  shemp      Samuel Horwitz
  JoeBesser
  JoeDeRita
  ```

  Simple lists are indicated in the configuration by appending a ‘‘=LIST’’ to the name.

  ```
  [VMS;STOOGES=list]
  ```

  It also possible to use a simple list for authentication purposes. The plain-text password is appended to the username with a trailing equate symbol. Although in general this is not recommended as everything is stored as plain-text it may be suitable as an ad hoc solution in some circumstances. The following example shows the format.

  ```
  # silly example
  fred=dancesalittle  Guess who?
  ginger=rogers       No second prizes!
  ```

- **HTA Database**

  These are binary, fixed 512 byte record files, containing authentication and authorization information. HTA databases may be used for authentication and group membership purposes. The content is much the same, the role differs according to the location in the realm directive. The contents these files must be accessed and modified via the server administration menu. See Section 14.3. They are located in the HT_AUTH: directory and have an extension of ‘‘.HTA’’.

(Essentially for historical reasons) HTA databases are the default sources for authorization information. Therefore, using just a name, with no trailing ‘‘=*something*’’, will configure an HTA source. Also, and recommended for clearly showing the intention, appending the ‘‘=HTA’’ qualifier specifies an HTA database. The following example show some of the variations.

```
[VMS;PROJECT_A=hta]
[DEVELOPERS=hta;PROJECT_A=hta]
```

- **Authorization Agent**

  An authorization agent is a CGI-compliant script that is specially activated during the authorization processing. Using CGI environment variables it gets details of the request, makes an assessment based on it’s own internal authentication/authorization processing, and using the script *callout* mechanism returns it’s results to the server, which then acting on these, allows or denies access.

  Such agents allows a site to develop local authentication/authorization mechnisms relatively easily, based on CGI principles. A discussion of such a development is not within the scope of this section, see the ‘‘WASD Scripting Environment’’ document for information on the use of callouts, and the example and working authorization agents provided in the HT_ROOT:[SRC.AGENT] directory. The description at the beginning of these programs covers these topics in some detail.

  An authorization agent would be configured using something like the following, where the ‘‘AUTHAGENT’’ is the actual script name doing the authorization. This has the the path ‘‘/cgiauth-bin/’’ prepended to it.

  ```
  ["Example Agent"=AUTHAGENT_EXAMPLE=agent]
  /some/path/or/other/* r+w
  ```

  It is possible to supply additional, per-path information to an agent. This can be any free-form text (up to a maximum length of 63 characters). This might be a configuration file location, as used in the example CEL authenticator. For example,

  ```
  ["CEL Authenticator"=AUTHAGENT_CEL=agent]
  /some/path/or/other/* r+w,param=HT_ROOT:[LOCAL]CEL1.LIS
  /a/nother/path/* r+w,param=HT_ROOT:[LOCAL]CEL2.LIS
  ```

  It is necessary to have the following entry in the HTTPD$MAP configuration file:

  ```
  exec+ /cgiauth-bin/* /cgi-bin/*
  ```

  This implies that authentication agents are always CGIplus scripts (technically it’s possble for vanilla CGI script to be agents, but for efficiency reasons this is discouraged). It also allows authentication scripts to be located outside of the general server tree if desired.

Multiple Source Types

A realm directive may contain one or more different types of authorization information source, with the following restrictions.

- Rights identifiers may only be used with SYSUAF authenticated requests. The following combinations would therefore not be allowed.

  ```
  [DEVELOPERS;PROJECT_A=id]
  [DEVELOPERS=hta;LIBRARIAN=id;PROJECT_A=list]
  [STOOGES=list;MOE_HOWARD=id]
  ```

- WASD rights identifiers (deprecated) may only be used for group membership when the /AUTHORIZE=WASD server qualifier has been specified at startup, and the username has been authenticated using a WASD identifier. See Section 11.9.2.

Realm Description

It is possible to supply text describing the authentication realm to the browser user that differs from the actual source name. This may be used to disguise the actual source or to provide a more informative description than the source name conveys.

Prefixing the actual realm source name with a double-quote delimited string (of up to 31 characters) and an equate symbol will result in the string being sent to a browser as the realm description during an authentication challenge. Here are some examples.

```
["the local host"=VMS]
["Social Club"=SOCIAL_CLUB_RW=id]
["Finance Staff"=FINANCE=list]
["Just Another Database"=DBACCESS=hta]
```

Note

> The *Digest* authentication scheme uses the realm description at both server and browser in the encrypted password challenge and response. When passwords are stored in an HTA file this realm synonym cannot be changed without causing these passwords to be rendered invalid.

11.5 Realm, Full-Access, Read-Only

WASD authorization offers a number of combinations of access control. This is a summary. Please note that when refering to the *level-of-access* a particular username may be allowed (read-only or full, read-write access), that it is always moderated by the level-of-access provided with a path configured within that realm. See Section 11.2.

- **Authentication Only**

  When a path is controlled by a realm that comprises an authentication source only, as in this example

  ```
  [authentication-source]
  ```

  usernames authenticated using that are granted full (read and write) access.

- **Authentication and Group**

Where a group membership source is provided following the authentication source, as illustrated in this example

```
[authentication-source;group-source]
```

the level-of-access depends on the source of the group membership. If from a *simple-list* of usernames or via a *VMS rights identifier* the username receives full (read and write) access. If from an HTA database the access is dependent on what is set against that user in the database. It can be either full or read-only.

- **Authentication and Two Groups**

  When a second group is specified, as in

  ```
  [authentication-source;group-source;group-source]
  ```

  the authentication is interpreted in a fixed fashion. The first group specified contains usernames to be granted full (read and write) access. The second group read-only access. Should a username occur in both groups full access takes precedence.

11.6 Virtual Servers

As described in Section 6.2, virtual service syntax may be used with authorization mapping to selectively apply rules to one specific service. This example provides the essentials of using this syntax. Note that service-specific and service-common rules may be mixed in any order allowing common authorization environments to be shared.

```
# authorization rules example for virtual servers
[[alpha.wasd.dsto.defence.gov.au:443]]
# ALPHA SSL is the only service permitting VMS (SYSUAF) authentication
[LOCAL=vms]
/web/* https:,r+w ; r
/httpd/-/admin/* ~daniel,https:,r+w
[[beta.wasd.dsto.defence.gov.au:80]]
# BETA has it's own HTA database
[BETA_USER=hta]
/web/* r+w ; r
[[gamma.wasd.dsto.defence.gov.au:80]]
# GAMMA likewise
[GAMMA_DEVELOPER=id;PROJECT-A=list]
/web/project/a/* r+w ; r
[GAMMA_DEVELOPER=id;PROJECT-B=list]
/web/project/b/* r+w ; r
[[*]]
# allow anyone from the local subnet to upload to here
[WORLD]
/web/unload/* 131.185.200.*,r+w
```

The server administration menu path authorization report (see Section 14.2) provides a selector allowing the viewing and checking of rules showing all services or only one particular virtual server, making it simpler to see exactly what any particular service is authorizing against.

11.7 Authorization Configuration Examples

Mixed case is used in the configuration examples (and should be in configuration files) to assist in readability. Rule interpretation however is completely case-insensitive.

1. In the following example the authentication realm is ‘‘WASD’’, a synonym for SYSUAF authentication, and the permissions group ‘‘SOCIALCLUB’’, a simple list of usernames. The directive allows those authenticated from the WASD realm and in the SOCIALCLUB group full access (read and write), and the world read-only.

```
[WASD=vms;SOCIALCLUB=list]
/web/socialclub/* r+w ; read
```

2. This example illustrates restricting access according internet address. Both the group and world restriction is identical, but the group address is being specified numerically, while the world access is being specified alphabetically (just for the purposes of illustration). This access check is done doing simple wildcard comparison, and makes numerical specifications potentially more efficient because they are usually shorter. The second line restricts that path’s write access even further, to one username, ‘‘BLOGGS’’.

```
[WASD=vms;SOCIALCLUB=list]
/web/socialclub/* 131.185.45.*,get,post; *.dsto.defence.gov.au,get
/web/socialclub/accounts/* 131.185.45.*,~BLOGGS,get,post; *.dsto.defence.gov.au,get
```

3. Three sources for authorization are specified in the following example. As the authentication source is VMS (by rights identifier), the full-access group and read-only group can also be determined by possessing the specified identifiers. The first path can only be written to by those holding the full-access identifier (librarian), the second path can only be read by both. The world has no access to these paths.

```
[DEVELOPER=id;PROJECT_A_LIBRARIAN=id;PROJECT_A_USER=id]
/web/projects/a/*  r+w
/web/projects/*    r
```

4. In the following example the authentication realm and group are a single HTA database, ‘‘ADMIN’’. The first directive allows those in the ADMIN group to read and write, and the world to read (‘‘get,post;get’’). The second line restricts write and even read access to ADMIN group, no world access at all (‘‘get,post’’).

```
[ADMIN=hta]
/web/everyone/* get,post;get
/web/select/few/* get,post
```

5. With this example usernames are used to control access to the specified paths. These usernames are authenticated from the COMPANY database. The world has read access in both cases. Note the realm description, ‘‘The Company’’.

```
["The Company"=COMPANY=hta]
/web/docs/* ~Howard,~George,~Fred,r+w ; r
/web/accounts/* ~George,r+w ; r
```

6. The following example shows a path specifying the local system’s SYSUAF being used to authenticate any usernames. Whenever using SYSUAF authentication it is **strongly recommended to limit the potential hosts** that can authenticate in this way by always using a host-limiting access restriction list. The world gets read access.

```
[VMS]
/web/local/area/* 131.185.250.*,r+w ; r
```

7. To restrict server administration to browsers executing on the server system itself and the SYSUAF-authenticated username DANIEL use a restriction list similar to the following. It also shows the use of SYSUAF-authentication being hidden by using a realm description.

   ```
   ["not the VMS SYSUAF"=VMS]
   /httpd/-/admin/*  #localhost,~daniel,r+w
   ```

8. The following example illustrates providing a read and writable area (GET, POST and PUTable) to hosts in the local network **without username authentication** (careful!).

   ```
   [WORLD]
   /web/scratch/*  *.local.hosts.only,r+w
   ```

11.7.1 KISS

WASD authorization allows for very simple authorization environments and provides the scope for quite complex ones. The path authentication scheme allows for multiple, individually-maintained authentication and authorization databases that can then be administered by autonomous managers, applying to widely diverse paths, all under the ultimate control of the overall Web administrator.

Fortunately great complexity is not generally necessary.

Most sites would be expected to require only an elementary setup allowing a few selected Web information managers the ability to write to selected paths. This can best be provided with the one authentication database containing read and write permissions against each user, with and access-restriction list against individual paths.

For example. Consider a site with three departments, each of which wishes to have three representatives capable of administering the departmental Web information. Authentication is via the SYSUAF. Web administrators hold an approriate VMS rights identifier, ‘‘WEBADMIN’’. Department groupings are provided by three simple lists of names, including the Web administrators (whose rights identifier would not be applied if access control is via a simple list), a fourth lists those with read-only access into the Finance area. The four grouping files would look like:

```
# Department 1              # Department 2
WEB1                        WEB1
WEB2                        WEB2
JOHN                        RINGO
PAUL                        CURLY
GEORGE                      LARRY

# Department 3              # Finance (read access)
WEB1                        PAUL
WEB2                        GEORGE
MOE                         JOHN
SHEMP                       RINGO
MAC
```

The authorization configuration file then contains:

```
######################################################################
# allow web masters (!) to use the administration menu
#                       to revise web configuration files
# world has no access (read or write)
# access is only allowed from a browser in the same subnet as the HTTPd
["Hypo Thetical Corp."=HYPOTHETICAL=vms;WEBADMIN=id]
/httpd/-/admin/*  #150.15.30.*,r+w
/ht_root/local/*  #150.15.30.*,r+w

# allows Department 1 representatives to maintain their web
# this may only be done from within the company subnet
# world has read access
["Hypo Thetical Corp."=HYPOTHETICAL=vms;DEPARTMENT1=list]
/web/dept/general/*   150.15.30.*,r+w ; r

# and so on for the rest of the departments

["Hypo Thetical Corp."=HYPOTHETICAL=vms;DEPARTMENT2=list;FINANCE=list]
# no world read access into finance, only those in the FINANCE list
/web/dept/finance/*    150.15.30.*,r+w

["Hypo Thetical Corp."=HYPOTHETICAL=vms;DEPARTMENT3=list]
/web/dept/inventory/*       150.15.30.*,r+w ; r
/web/dept/production/*      150.15.30.*,r+w ; r
# (the next uses line continuation just for illustration)
/web/dept/marketing/*       150.15.30.*,\
                            r+w ;\
                            read

# we need an area for general POSTing (just for illustration :^)
[WORLD]
/web/world/*  r+w

######################################################################
```

11.8 Authorization Cache

Access to authentication sources, SYSUAF, simple lists and HTA databases, are relatively expensive operations. To reduce the impact of this activity on request latency and general server performance authentication and realm-associated permissions for each authenticated username are stored in a cache. This means that only the initial request needs to be checked from appropriate databases, subsequent ones are resolved more quickly and efficiently from cache.

Such cached entries have a finite lifetime associated with them. This ensures that authorization information associated with that user is regularly refreshed. This period, in minutes, is set using the [AuthCacheMinutes] configuration parameter. Zero disables caching with a consequent impact on performance.

Implication

Where-ever a cache is employed there arises the problem of keeping the contents current. The simple lifetime on entries in the authentication cache means they will only be checked for currency whenever it expires. Changes may have occured to the databases in the meantime.

Generally there is are other considerations when adding user access. Previously the user attempt failed (and was evaluated each time), now the user is allowed access and the result is cached.

When removing or modifying access for a user the cached contents must be taken into account. The user will continue to experience the previous level of access until the cache lifetime expires on the entry. When making such changes it is recommended to explicitly purge the authentication cache either from the command line using /DO=AUTH=PURGE (see Section 5.3.2) or via the server administration menu (see Chapter 14). Of course the other solution is just to disable caching, which is a less than optimal solution.

11.9 SYSUAF-Authenticated Users

The ability to authenticate using the system's SYSUAF is controlled by the server /SYSUAF[=keyword] qualifier. By default it is disabled. Also see Section 11.9.3.

By default accounts with SYSPRV authorized are always rejected to discourage the use of potentially significant usernames (e.g. SYSTEM). This behaviour can be changed through the use of specific identifiers, see Section 11.9.1 immediately below. Accounts that are disusered, have passwords that have expired or that are captive or restricted are always rejected. Accounts that have access time restricting REMOTE or NETWORK access will have those restrictions honoured (see Section 11.9.1 for a workaround for this.)

11.9.1 Rights Identifiers

Whether or not any particular username is allowed to authenticate via the SYSUAF may be controlled by that account holding or not holding a particular VMS rights identifier. When a username has been authenticated via the SYSUAF, rights identifiers associated with that account may be used to control the level-of-access within that realm.

Use of identifiers for these purposes are enabled using the /SYSUAF=ID server startup qualifier.

Three reserved identifier names are optional. A warning will be reported during startup if these are not found.

1. **WASD_HTTPS_ONLY -** restricts accounts holding it to authenticating using SSL (https:). Authentication via a standard ''http:'' will always be denied.
2. **WASD_NIL_ACCESS -** allows accounts with access time restrictions to authenticate via the SYSUAF. This is particularly intended to support the use of nil-access accounts, see Section 11.9.3.
3. **WASD_PASSWORD_CHANGE -** allows an account to modify it's SYSUAF password, if this is configured for the server, see Section 11.12.

Identifiers may be managed using the following commands. If unsure of the security implications of this action consult the relevant VMS system management security documentation.

```
$ SET DEFAULT SYS$SYSTEM
$ MCR AUTHORIZE
UAF> ADD /IDENTIFIER WASD_HTTPS_ONLY
UAF> ADD /IDENTIFIER PROJECT_USER
UAF> ADD /IDENTIFIER PROJECT_DEVELOPER
UAF> ADD /IDENTIFIER PROJECT_LIBRARIAN
```

They can then be provided to desired accounts using commands similar to the following:

```
UAF> GRANT /IDENTIFIER PROJECT_USER <account>
```

and removed using:

```
UAF> REVOKE /IDENTIFIER PROJECT_USER <account>
```

Be aware that, as with all successful authentications, and due to the WASD internal authentication cache, changing database contents does not immediately affect access. Any change in the RIGHTSLIST won't be reflected until the cache entry expires or it is explicitly flushed (see Section 11.8).

11.9.2 WASD "Hard-Wired" Identifiers

Deprecated and Discouraged

This description is included for reasons of backward compatibility. As of version 6.0 the WASD identifiers must be enabled using the /SYSUAF=WASD qualifier.

When SYSUAF authentication is enabled, by default all non-privileged, active accounts are capable of authentication. Restriction of this to those actually requiring such a capability is provided using VMS rights identifiers. When the /SYSUAF=WASD qualifier is employed a VMS account must possess one of two specific identifiers before it is allowed to be used for server authentication. Note that this mechanism can also allow privileged accounts to be so used . . . deploy with discretion! Note also that the use of access identifiers with this facility allows a much finer control of which accounts may be used for authentication and so is also the prefered mechanism for deploying SYSUAF authentication.

1. **WASD_VMS_R**

   This identifier provides at most read access.

2. **WASD_VMS_RW**

   This identifier provides read and write access (path protections still apply of course).

Other identifiers provide further control for the way in which the authenticated account may be used.

1. **WASD_VMS_HTTPS**

   Use of the authenticated account is restricted to ''https:'' (SSL) requests.

2. **WASD_VMS_PWD**

   The account is allowed to change it's SYSUAF password (!!). It is recommended this facility only be employed with SSL in place.

   Password modification is enabled by including a mapping rule to the internal change script.

   ```
   map /httpd/-/change/* /httpd/-/change/*
   ```

The authorization configuration file must provide authenticated access.

```
[VMS]
/httpd/-/change/vms/* https:,r+w
```

Also see Section 11.12.

3. **WASD_VMS__<group-name>**

   This form allows a suitably named identifier to be created for use in providing group-membership via the SYSUAF. Note the double-underscore separating the fixed from the locally specified portion. Using these identifiers it is possible to limit paths to SYSUAF-authenticated accounts possessing the requisite identifier in manner similar to non-SYSUAF-authentication groups. An account possessing the WASD_VMS__TESTING identifier is allowed write access to the path in the following example:

   ```
   [VMS;TESTING]
   /web/project/testing/* r+w ; r
   ```

All four rights identifiers must exist for the /SYSUAF=ID facility to be used (even though none may be granted to any account). The identifiers may be created using the AUTHORIZE utility with following commands:

```
$ SET DEFAULT SYS$SYSTEM
$ MCR AUTHORIZE
UAF> ADD /IDENTIFIER WASD_VMS_R
UAF> ADD /IDENTIFIER WASD_VMS_RW
UAF> ADD /IDENTIFIER WASD_VMS_HTTPS
UAF> ADD /IDENTIFIER WASD_VMS_PWD
```

They can then be provided to desired accounts using commands similar to the following:

```
UAF> GRANT /IDENTIFIER WASD_VMS_RW <account>
```

and removed using:

```
UAF> REVOKE /IDENTIFIER WASD_VMS_RW <account>
```

11.9.3 Nil-Access VMS Accounts

It is possible, and may be quite effective for some environments, to have a SYSUAF account or accounts strictly for HTTP authorization, with no actual interactive or other access allowed to the VMS system itself. This would relax the caution on the use of SYSUAF authentication outside of SSL transactions. An obvious use would be for the HTTP server administrator. Additional accounts could be provided for other authorization requirements, all without compromising the system's security.

In setting up such an environment it is vital to ensure the HTTPd server is started using the /SYSUAF=ID qualifier (see Section 11.1). This will require all SYSUAF-authenticated accounts to possess a specific VMS resource identifier, accounts that do not possess the identifier cannot be used for HTTP authentication. In addition the identifier WASD_NIL_ACCESS will need to be held (see Section 11.9.1), allowing the account to authenticate despite being restricted by REMOTE and NETWORK time restrictions.

To provide such an account select a group number that is currently unused for any other purpose. Create the desired account using whatever local utility is used then activate VMS AUTHORIZE and effectively disable access to that account from all sources and grant the appropriate access identifier (see Section 11.9.1 above).

```
$ SET DEFAULT SYS$SYSTEM
$ MCR AUTHORIZE
UAF> MODIFY <account> /NOINTERACTIVE /NONETWORK /NOBATCH /FLAG=DISMAIL
UAF> GRANT /IDENTIFIER WASD_VMS_RW <account>
```

11.9.4 SYSUAF and SSL

When SSL is in use (see Chapter 13) the username/password authentication information is inherently secured via the encrypted communications of SSL. To enforce access to be via SSL add the following to the HTTPD$MAP configuration file:

```
/whatever/path/you/like/*  "403 Access denied."  ![sc:https]
```

or alternatively the following to the HTTPD$AUTH configuration file:

```
[REALM]
/whatever/path/you/like/*  https:
```

Note that this mechanism is applied **after** any path and method assessment made by the server's authentication schema.

The qualifier /SYSUAF=SSL provides a powerful mechanism for protecting SYSUAF authentication, restricting SYSUAF authenticated transactions to the SSL environment. The combination /SYSUAF=(SSL,ID) is particularly effective.

Also see Section 11.1.

11.9.5 Security Profile

It is possible to control access to files and directories based on the VMS security profile of a SYSUAF-authenticated remote user. This functionality is implemented using VMS security system services involving SYSUAF and RIGHTSLIST information. The feature must be explicitly allowed using the server /PROFILE qualifier. By default it is disabled.

NOTE: Use caution when deploying the /PROFILE qualifier. It was really designed with a very specific environment in mind, that of an Intranet where the sole purpose was to provide VMS users access to their normal VMS resources via a Web interface.

When a SYSUAF-authenticated user (i.e. the VMS realm) is first authenticated a VMS security-profile is created and stored in the authentication cache (see Section 11.8). A cached profile is an efficient method of implementing this as it obviously removes the need of creating a user profile each time a resource is assessed. If this profile exists in the cache it is attached to each request authenticated for that user. As it is cached for a period, any change to a user's security profile in the SYSUAF or RIGHTSLIST won't be reflected in the cached profile until the cache entry expires or it is explicitly flushed (see Section 14.4).

When a request has this security profile all accesses to files and directories are assessed against it. When a file or directory access is requested the security-profile is employed by a VMS security system service to assess the access. If allowed, it is provided via the SYSTEM file protection field. Hence it is possible to be eligible for access via the OWNER field but not actually be able to access it because of SYSTEM field protections! If not allowed, a ‘‘no privilege’’ error is generated.

Of course, this functionality only provides access for the server, IT DOES NOT PROPAGATE TO ANY SCRIPT ACCESS. If scripts must have a similar ability they should implement their own scheme (which is not too difficult,) see HT_ROOT:[SRC.MISC]CHKACC.C based on the CGI variable WWW_AUTH_REALM which would be ‘‘VMS’’ indicating SYSUAF-authentication, and the authenticated name in WWW_REMOTE_USER.

Performance Impact

If the /PROFILE qualifier has enabled SYSUAF-authenticated security profiles, whenever a file or directory is assessed for access an explicit VMS security system service call is made. This call builds a security profile of the object being assessed, compares the cached user security profile and returns an indication whether access is permitted or forbidden. This is addition to any such assessments made by the file system as it is accessed.

This extra security assessment is not done for non-SYSUAF-authenticated accesses within the same server.

For file access this extra overhead is negligible but becomes more significant with directory listings (‘‘Index of’’) where each file in the directory is independently assessed for access.

11.10 Controlling Server Write Access

The server account should have no direct write access to into any directory structure. Files in these areas should be owned by SYSTEM ([1,4]). Write access for the server into VMS directories (using the POST or PUT HTTP methods) should be controlled using VMS ACLs. **This is in addition to the path authorization of the server itself of course!** The recommendation to have no ownership of files and provide an ACE on required directories prevents inadvertant mapping/authorization of a path resulting in the ability to write somewhere not intended.

Two different ACEs implement two grades of access.

1. If the ACE grants **CONTROL** access to the server account then only VMS-authenticated usernames with security profiles can potentially write to the directory. Only potentially, because a further check is made to assess whether that VMS account in particular has write access.

   This example shows a suitable ACE that applies only to the original directory:

   ```
   $ SET SECURITY directory.DIR -
     /ACL=(IDENT=HTTP$SERVER,ACCESS=READ+WRITE+EXECUTE+DELETE+CONTROL)
   ```

   This example shows setting an ACE that will propagate to created files and importantly, subdirectories:

   ```
   $ SET SECURITY directory.DIR -
     /ACL=((IDENT=HTTP$SERVER,OPTIONS=DEFAULT,ACCESS=READ+WRITE+EXECUTE+DELETE+CONTROL), -
           (IDENT=HTTP$SERVER,ACCESS=READ+WRITE+EXECUTE+DELETE+CONTROL))
   ```

2. If the ACE grants **WRITE** access then the directory can be written into by any authenticated username for the authorized path.

This example shows a suitable ACE that applies only to the original directory:

```
$ SET SECURITY directory.DIR -
  /ACL=(IDENT=HTTP$SERVER,ACCESS=READ+WRITE+EXECUTE+DELETE)
```

This example shows setting an ACE that will propagate to created files and importantly, subdirectories:

```
$ SET SECURITY directory.DIR -
  /ACL=((IDENT=HTTP$SERVER,OPTIONS=DEFAULT,ACCESS=READ+WRITE+EXECUTE+DELETE), -
        (IDENT=HTTP$SERVER,ACCESS=READ+WRITE+EXECUTE+DELETE))
```

To assist with the setting of the required ACEs an example, general-purpose DCL procedure is provided, HT_ROOT:[EXAMPLE]AUTHACE.COM.

11.11 Securing All Requests

Some sites may be sensitive enough about Web resources that the possibility of providing inadvertant access to some area or another is of major concern. WASD provides a facility that will automatically deny access to any path that does not appear in the authorization configuration file. This does mean that all paths requiring access must have authorization rules associated with them, but if something is missed some resource does not unexpectedly become visible.

At server startup the /AUTHORIZE=ALL qualifier enables this facility.

For paths that require authentication and authorization the standard realms and rules apply. To indicate that a particular path should be allowed access, but that no authorization applies the ‘‘NONE’’ realm may be used. The following example provides some indication of how it should be used.

```
# allow the librarian to update this area, world to read it
[VMS;LIBRARIAN=id]
/web/library/* r+w ; read
# indicate there is no authorization to be applied
[NONE]
# allow access to general web areas
/web/*
# allow access to the HT_ROOT tree
/ht_root/*
```

11.12 User Password Modification

The server provides for users to be able to change their own HTA passwords (and SYSUAF if required). This functionality, though desirable from the administrator’s viewpoint, is not mandatory if the administrator is content to field any password changes, forgotten passwords, etc. Keep in mind that passwords, though not visible during entry, are passed to the server using clear-text form fields.

Password modification is enabled by including a mapping rule to the internal change script. For example:

```
map /httpd/-/change/* /httpd/-/change/*
```

Any database to be enabled for password modification must have a writable authorization path associated with it. For example:

```
[GROUP]
/httpd/-/change/group/* r+w

[ANOTHER_GROUP]
/httpd/-/change/another_group/* r+w
```

Use some form of cautionary wrapper if providing this functionality:

```
<H2>Change Your Authentication</H2>

<BLOCKQUOTE><I>
Change the password used to identify yourself to the REALM Web environment for
some actions.  Note that this <U>not</U> an operating system password, nor has
it anything to do with it.  Due to the inherent weaknesses of using
non-encrypted password transmissions on networks <FONT COLOR="#ff0000"><U>DO
NOT</U> use a password you have in use anywhere else, especially an operating
system password!</FONT> You need your current password to make the change.  If
you have forgotten what it is contact <A HREF="/web/webadmin.html">WebAdmin</A>,
preferably via e-mail, for the change to be made on your behalf.
</I></BLOCKQUOTE>

<UL>
<LI><A HREF="/httpd/-/change/REALM/">REALM</A> realm.
</UL>
```

11.13 Cancelling Authorization

The reason authorization information is not required to be reentered on subsequent accesses to controlled paths is cached information the browser maintains. It is sometimes desirable to be able to access the same path using different authentication credentials, and correspondingly it would be useful if a browser had a *purge authorization cache* button, but this is commonly not the case. To provide this functionality the server must be used to ‘‘trick’’ the browser into cancelling its authorization information for a particular path.

This is achieved by adding a specific query string to the path requiring cancellation. The server detects this and returns an authorization failure status (401) regardless of the contents of request ‘‘Authorization:’’ field. This results in the browser flushing that path from it’s authorization cache, effectively requiring new authorization information the next time that path is accessed.

The basic procedure is as follows.

1. Add the query string ‘‘?httpd=cancel’’ to the path in question (if there is an existing query then replace it), as in the following example.

   ```
   /the/current/path?httpd=cancel
   ```

2. The browser will respond with an authorization failure, and prompting to retry or reenter the username and password.
3. It is necessary to clear at least the password (i.e. remove any password from the appropriate field) and reenter.
4. The browser again responds with an authorization failure.

5. At this stage the authorization dialog can be cancelled, resulting in a server authorization failure message.
6. The original path can now be returned to and reaccessed. The browser should again prompt for authorization information at which point different credentials may be supplied.

Chapter 12

Proxy Services

A proxy server acts as an intermediary between Web clients and Web servers. It listens for requests from the clients and forwards these to remote servers. The proxy server then receives the responses from the servers and returns them to the clients. Why go to this trouble? There are several reasons, the most common being:

- Using the proxy server to allow internal clients access to the Internet from behind a firewall. Browsers behind the firewall have full Web access via the proxy system.
- Many proxy servers provide caching, or local storage, of responses. For frequent or commonly accessed resources this can not only significantly reduce apparent network latency but also greatly reduce the total traffic downloaded by a site.
- For anonymity. Although often related directly to firewall security considerations, it can also sometimes be an advantage to just not reveal the exact source of Web transactions from within your local network.

Proxy Serving Quick-Start

No additional software needs to be installed to provide proxy serving. The following steps provide a brief outline of proxy configuration.

1. Enable proxy serving and specify which particular services are to be proxies (Section 12.1.2).
2. If proxy caching is required (most probably, see Section 12.2)
    - Decide on a cache device, create the cache root directory, modify server startup procedures to include the HT_CACHE_ROOT logical name (Section 12.2.1).
    - Enable caching on required services (Section 12.2.2).
    - Adjust relevant cache management configuration parameters if required (Section 12.2.3).
    - If required adjust cache retention parameter (Section 12.2.5).

3. If providing SSL tunnelling (proxy of Secure Sockets Layer transactions) add/modify a service for that (Section 12.4).
4. Add HTTPD$MAP mapping rules for controlling this/these services (Section 12.1.3, Section 12.4.2, and Section 12.3).
5. Restart server (HTTPD/DO=RESTART).

Configuration Parameter Summary

- **[ProxyForwardedBy]** ENABLED|DISABLED *(default: DISABLED)*

  Enables and disables the addition of a proxy request header line providing information that the request has been forwarded by another agent. The added header line would look like ‘‘Forwarded: by http://host.name.domain (HTTPd-WASD/n.n.n OpenVMS/AXP Digital-TCPIP SSL)’’.

- **[ProxyHostLookupRetryCount]** ***integer*** *(default: 0)*

  When the server is resolving the name of a remote host the request may timeout due to up-stream DNS server latencies. For sites where this is commonplace (report as *unknown host* when you know quite well the host exists) this parameter allows a number of retries, at five second intervals, to be enabled. A value of two or three should be quite sufficient for most purposes.

- **[ProxyReportLog]** ENABLED|DISABLED *(default: DISABLED)*

  Enables or disables the server process log reporting siginificant proxy processing events, such as cache maintenance activity.

- **[ProxyReportCacheLog]** ENABLED|DISABLED *(default: DISABLED)*

  Enables or disables the server process log reporting of proxy caching activity.

- **[ProxyServing]** ENABLED|DISABLED *(default: DISABLED)*

  Enables or disables proxy serving on a whole-of-server basis, irrespective of any proxy services that might be configured.

- **[ProxyCache]** ENABLED|DISABLED *(default: DISABLED)*

  Enables or disables proxy caching on a whole-of-server basis, irrespective of any proxy services that might be configured for caching.

- **[ProxyCacheFileKBytesMax]** ***integer*** *(default: 256)*

  Maximum size of a cache file in kilobytes before it will not cached.

- **[ProxyCacheRoutineHourOfDay]** ***integer*** *(default: 0)*

  Hour of day for *routine* cache purge (00-23).

- **[ProxyCacheDeviceCheckMinutes]** ***integer*** *(default: 15)*

  Interval in minutes between checking space availablility on cache device. If space is not available a *reactive* purge is initiated.

- **[ProxyCacheDeviceMaxPercent]** ***integer*** *(default: 85)*

The maximum percentage in use on the cache device before a *reactive* purge is scheduled. If device usage exceeds this limit no more cache files are created.

- **[ProxyCacheDevicePurgePercent]** ***integer*** *(default: 1)*

  The percentage by which the cache device usage is attempted to be reduced when a *reactive* purge is initiated.

- **[ProxyCachePurgeList]** ***string*** *(default: 168,48,24,8,0)*

  A list of comma-separated integers representing the sequence of last accessed period in hours used during a progressive *reactive* purge.

- **[ProxyCacheReloadList]** ***string*** *(default: 1,2,4,8,12,24,48,96,168)*

  A list of comma-separated integers representing the sequence of age in hours used when determining whether a cache file's contents should be reloaded.

- **[ProxyHostCachePurgeHours]** ***integer*** *(default: 24)*

  The period at which the cache of host names to IP addresses is purged.

12.1 HTTP Proxy Serving

WASD (currently) provides a proxy service for the HTTP scheme (prototcol).

Proxy serving generally relies on DNS resolution of the requested host name. DNS lookup can introduce significant latency to transactions. To help ameliorate this WASD incorporates a host name cache. To ensure cache consistency the contents are regularly flushed, after which host names must use DNS lookup again, refreshing the information in the cache. The period of this cache purge is contolled with the [ProxyHostCachePurgeHours] configuration parameter.

When a request is made by a proxy server is is common for it to add a line to the request header stating that it is a forwarded request and the agent doing the forwarding. With WASD proxying this line would look something like this:

```
Forwarded: by http://host.name.domain (HTTPd-WASD/6.0.0 OpenVMS/AXP Digital-TCPIP SSL)
```

It is enabled using the [ProxyForwardedBy] configuration parameter.

12.1.1 Reverse Proxy

The use of WASD proxy serving as a firewall component assumes two configured network interfaces on the system, one of which is connected to the internal network, the other to the external network. (Firewalling could also be accomplished using a single network interface with router blocking external access to all but the server system.) Outgoing (internal to external) proxying is the most common configuration, however a proxy server can also be used to provide controlled external access to selected internal resources. This is sometimes known as *reverse proxy*.

In this configuration the proxy server is contacted by an external browser with a standard HTTP request. Proxy server rules map this request onto a proxy-request format result. For example:

```
pass /sales/* http://sales.corporate.server.com/*
```

The server recognises the result format and performs a proxy request to a system on the internal network. Note that the mappings required could become quite complex, but it is possible. See example 7 in Section 12.1.3.

12.1.2 Enabling Proxy Serving

Proxy serving is enabled on a per-server basis using the [ProxyServing] configuration parameter.

The actual services providing the proxy serving (i.e. the host and port) are specified on a per-service basis. This means it is possible to have proxy and non-proxy services deployed on the one server (on different ports of course). Proxying is enabled by appending the *proxy* keyword to the particular service specification. The following example shows a non-proxy and proxy service.

```
[Service]
http://alpha.wasd.dsto.defence.gov.au:80
http://alpha.wasd.dsto.defence.gov.au:8080;proxy
```

Proxy Chaining

Some sites may already be firewalled and have corporate proxy servers providing Internet access. It is quite possible to use WASD proxying in this environment, where the WASD server makes it's proxied requests via the next proxy server in the hierarchy. This is known as *proxy chaining*. Using the *chain* keyword specify the host name of the next server when enabling the proxy service, as in this example:

```
[Service]
http://alpha.wasd.dsto.defence.gov.au:8080;proxy;chain=next.proxy.host
```

12.1.3 Controlling Proxy Serving

Extensive control of how and by whom a proxy service is used may be exercised using WASD general and conditional mapping, (see Chapter 9 and Section 9.6), possibly in the context of a virtual service specification for the particular connect service host and port (see Section 9.5). The following examples provide a small indication of how mapping could be used in a proxy service context.

1. It is possible, though more often not practical, to regulate which hosts are connected to via the proxy service. For example, the following rule forbids accessing any site with the string ‘‘hacker’’ in it (for the proxy service ‘‘alpha . . . :8080’’.

   ```
   [[alpha.wasd.dsto.defence.gov.au:8080]]
   pass http://*hacker*/* "403 Proxy access to this host is forbidden."
   pass http://*
   ```

2. Or as in the following example, only allow access to specific sites.

   ```
   [[alpha.wasd.dsto.defence.gov.au:8080]]
   pass http://*.org/*
   pass http://*.digital.com/*
   pass http://* "403 Proxy access to this host is forbidden."
   ```

3. It is also possible to restrict access via the proxy service to selected hosts on the internal subnet. Here only a range of numeric addresses plus a single host in another subnet are allowed access to the service.

```
[[alpha.wasd.dsto.defence.gov.au:8080]]
pass http://* "403 Restricted access." ![ho:131.185.250.* ho:131.185.200.10]
pass http://*
```

4. In the following example POSTing to a particular proxied servers is not allowed (why I can't imagine, but hey, this is an example!)

```
[[alpha.wasd.dsto.defence.gov.au:8080]]
pass http://subscribe.sexy.com/* "403 POSTing not allowed." [me:POST]
pass http://*
```

5. It is possible to redirect proxied requests to other sites.

```
[[alpha.wasd.dsto.defence.gov.au:8080]]
redirect http://www.sexy.com/* http://www.disney.com/
pass http://*
```

6. A proxy service is just a specialized capability of a general HTTP service. Therefore it is quite in order for the one service to respond to standard HTTP requests as well as proxy-format HTTP requests. To enforce the use of a particular service as proxy-only, add a final rule to a virtual service's mapping restricting non-proxy requests.

```
[[alpha.wasd.dsto.defence.gov.au:8080]]
pass http://*
pass /* "403 This is a proxy-only service."
```

7. This example provides the essentials when supporting *reverse proxying*. Note that mappings may become quite complex when supporting access to resources across multiple internal systems (e.g. access to directory icons).

```
[[main.corporate.server.com:80]]
pass /sales/* http://sales.corporate.server.com/*
pass /shipping/* http://shipping.corporate.server.com/*
pass /support/* http://support.corporate.server.com/*
pass * "403 Nothing to access here!"
```

Note

To expedite proxy mapping is it recommended to have a final rule for the proxy virtual service that explicitly *pass*es the request. This would most commonly be a permissive pass as in example 1, could quite easily be an restrictive pass as in example 2, or a combination as in example 6.

12.2 Caching

Caching involves using the local file-system for storage of responses that can be reused when a request for the same URL is made. The WASD server does not have to be configured for caching, it will provide proxied access without any caching taking place.

When a proxied request is processed, and it's characteristics would allow the response to be cached, a unique identifier generated from the URL is used to create a corresponding file name. The response header and any body are stored in this file. This may be the data of an HTML page, a graphic, etc.

When a proxied request is being processed, and it's characteristics would allow the request to be cached, the unique identifier generated allows for a previously created cache file to be checked for. If it exists, and is current enough, the response is returned from it, instead of from the remote server. If it exists and is no longer current the request is re-made to the remote server, and the response if still cacheable is re-cached, keeping the contents current. If it does not exist the response is delivered from the remote server.

Not all responses can be cached!

The main critera are for the response to be successful (200 status), general (i.e. one not in response to a specialized query or action), and not too volatile (i.e. the same page may be expected to be returned more than once, preferably over an extended period).

Proxied *requests* can only be cached if . . .

- uses the GET method
- does not contain a query string

Proxied *responses* will only be cached if . . .

- status code begins with ''2'' (success)
- contains a *Last-Modified:* header field
- one or more hours since the last modification
- any *Expires:* date/time is still in the future
- does not contain a *Pragma: no-cache* field
- does not exceed a configuration parameter in size

The [ProxyCacheFileKbytesMax] configuration parameter controls the maximum size of a response before it will not be cached. This can be determined from any ''Content-Length:'' response header field, in which case it will proactively not be cached, or if during cache load the maximum size of the file increases beyond the specified limit the load is aborted.

Not all sites may benefit from cache!

As many transactions on today's Web contain query strings, cookies, etc., and therefore cannot be meaningfully cached, it should not be assumed the cost/benefit of having a proxy cache enabled is a forgone conclusion. Each site should monitor the proxy traffic reports and decide on a local policy.

The facilities described in Section 12.5 allow a reasonably informed decision to be made. Items to be considered.

- The ratio of cache reads to network accesses.
- The number of non-cacheable requests and responses, particularly as a percentage of total proxy traffic.
- The ratio of network to cache traffic, although this may be skewed by having a high ratio of 304 (not-modified) responses from cache (which contain few bytes). Check the cache 304 reporting item.

Last, but by no means least, understanding the characteristics of local usage. For example, are there a small number of requests generating lots of non-cacheable traffic? For instance, a few users accessing streaming content.

12.2.1 Cache Device

Selection of a disk device for supporting the proxy cache should not be made without careful consideration, doubly so if significant traffic is experienced. Here are some common-sense suggestions.

- **avoid** locating it as a subdirectory of HT_ROOT:[000000]
- use a disk with as little other activity as possible (both I/O and space usage)
- use a disk with as much free space as possible
- use the fastest disk available

Initially the directory will need to be created. This can be done manually as described below, or if using the supplied server startup procedures (see STARTUP.COM) it is checked for and if it does not exist is automatically created during startup. The directory must be owned by the HTTP$SERVER account and have full read+write+execute+delete access. It is suggested to name it [HT_CACHE] and may be created manually using the following command.

```
$ CREATE /DIR /OWN=HTTP$SERVER /PROT=(O:RWED,W) device:[HT_CACHE.]
```

It is a relatively simple matter to relocate the cache at any stage. Simply create the required directory in the new location, modify the startup procedures to reflect this, shut the server down completely then restart it using the procedures (**not** a /DO=RESTART!). It is probably not worth copying the cache files from the old to the new location, just start afresh.

HT_CACHE_ROOT Logical

It is required to define the logical name HT_CACHE_ROOT if any proxy services are specified in the server configuration. The server will not start unless it is correctly defined. The logical should be a *concealed device* logical specifying the top level directory of the cache tree. The following example shows how to define such a logical name.

```
$ DEFINE /SYSTEM /EXEC /TRANSLATION=CONCEALED HT_CACHE_ROOT device:[HT_CACHE.]
```

If example startup procedure is in use then it is quite straight-forward to have the logical created during server startup (see STARTUP.COM).

12.2.2 Enabling Caching

Caching may enabled on a per-service basis. This means it is possible to have a caching proxy service and a non-caching service active on the one server. Caching is enabled by appending the *cache* keyword to the particular service specification. The following example shows a non-proxy and a caching proxy service.

```
[Service]
http://alpha.wasd.dsto.defence.gov.au:80
http://alpha.wasd.dsto.defence.gov.au:8080;proxy;cache
```

Proxy caching may be selectively disabled for a particular site, sites or paths within sites using the *set nocache* mapping rule. This rule, used to disable caching for local requests, also disables proxy file caching for that subset of requests. This example shows a couple of variations.

```
[[alpha.wasd.dsto.defence.gov.au:8080]]
# disable caching for local site’s servers that respond fairly quickly
set http://*.local.domain/* nocache
# disable caching of log files
set http://*.log nocache
pass http://*
```

Note

It is also recommended to place the cache directory under some authorization control to prevent casual browsing and access of the cache contents. Something local, similar in intention to

```
[VMS]
/ht_cache_root/* ~webadmin,131.185.250.*,r+w ;
```

12.2.3 Cache Management

As the proxy cache is implemented using the local file system, management of the cache implies controlling the number of, and exactly which files remain in cache. Essentially then, management means when and which to delete. The [ProxyReportLog] configuration parameter enables the server process log reporting of cache management activities.

Cache file deletion takes two forms.

1. **ROUTINE**

   This ensures files that have not been accessed within specified limits are periodically and regularly deleted. The [ProxyCacheRoutineHourOfDay] configuration parameter controls this activity.

   The ROUTINE form occurs once per day at the specified hour. The cache files are scanned looking for those that exceed the configuration parameter for maximum period since last access, which are then deleted (the largest number of [ProxyCachePurgeList], as described below).

2. **REACTIVE**

   This is a remedial action, when cache device usage is reaching it’s configuration limit and files need to be deleted to free up space. The following parameters control this behaviour.

   [ProxyCacheDeviceCheckMinutes]
   [ProxyCacheDeviceMaxPercent]
   [ProxyCacheDevicePurgePercent]
   [ProxyCachePurgeList]

   The cache device space usage is checked at the specified interval.

   If the device reaches the specified percentage used a cache purge is initiated and by deleting files until the specified reduction is attained, the total space in use on the disk is reduced.

The cache files are scanned using the [ProxyCachePurgeList] parameter described below, working from the greatest to least number of hours in the steps provided. At each scan files not accessed within that period are deleted. At each few files deleted the device free space is checked as having reached the lower purge percentage limit, at which point the scan terminates.

This parameter has as it's input a series of comma-separated integers representing a series of hours since files were last accessed. In this way the cache can be progressively reduced until percentage usage targets are realized. Such a parameter would be specified as follows,

```
[ProxyCachePurgeList] 168,48,24,8,0
```

meaning the purge would first delete files not accessed in the last week, then not for the last two days, then the last twenty-four hours, then eight, then finally all files. The largest of the specified periods (in this case 168) is also used as the limit for the ROUTINE scan and file delete.

Once the target reduction percentage is reached the purge stops. During the purge operation further cache files are not created. Even when cache files cannot be created for any reason proxy serving still continues transparently to the clients.

Note

Cache files can be manually deleted at any time (from the command line) without disturbing the proxy-caching server and without rebuilding any databases. When deleting, the /BEFORE=*date/time* qualifier can be used, with /CREATED being the document's last-modified date, /REVISED being the last time it was loaded, and /EXPIRED the last time the file was accessed (used to supply a request). Be aware that on an active server it is quite possible some files may be locked at time of attempted deletion.

12.2.4 Cache Invalidation

For the purposes of this document, cache invalidation is defined as the determination when a cache file's data is no longer valid and needs to be reloaded.

The method used for cache validation is deliberately quite simple in algorithm and implementation. In this first attempt at a proxy server the overriding criteria have been efficiency, simplicity of implementation, and reliability. Wishing to avoid complicated revalidation using behind-the-scenes HEAD requests the basic approach has been to just invalidate the cache item upon exiry of a period related to it's ''Last-Modified:'' age or upon a *no-cache* request, both described further below.

- If a ''Pragma: no-cache'' request header field is present (as is generated by Netscape Navigator when using the *reload* function) then the server should completely reload the response from the remote server. (Too often the author seems to have received incomplete responses where the proxy server caches only part of a response and has seemed to refuse to explicitly re-request.) OK it's a a bit more expensive but who's to say the proxy server is right all the time! The response is still cached ... the next request may not have the *no-cache* parameter.

- When a response is cached the file *creation* date/time is set to the local equivalent of the ‘‘Last-Modified:’’ GMT date and time supplied with the response. In this manner the file’s absolute age can be determined quickly and easily from the file header. This is used as described in Section 12.2.5.
- When a file is cached, the *revision* and *expires* date/times are set to current. The revision date/time is used when assessing when the file was last loaded/validated/reloaded. Once a file is cached the RMS *expires* date/time is updated every time it is subsequently accessed. In this way recency of usage of the item can be easily tracked, allowing the routine and reactive purges to operate by merely checking the file header.

The *revision count* (automatically updated by VMS) tracks the absolute number of accesses since the file was created (actually a maximum of 65535, or an unsigned short, but that should be enough for informational purposes).

12.2.5 Cache Retention

The [ProxyCaheReloadList] configuration parameter is used to control when a file being accessed is reloaded from source.

This parameter supplies a series of integers representing the hours after which an access to a cache file causes the file to be invalidated and reloaded from it’s source during the proxied request. Each number in the series represents the lower boundary of the range between it and the next number of hours. A file with a last-loaded age falling within a range is reloaded at the lower boundary of that particular range. The following example

```
[ProxyCacheReloadList] 1,2,4,8,12,24,48,96,168
```

would result in a file 1.5 hours old being reloaded every hour, 3.25 hours old every 2 hours, 7 hours old every 4 hours, etc. Here "old" means since last (or of course first) loaded. Files not reloaded since the final integer, in this example 168 (one week), are always reloaded.

12.3 FTP Proxy Serving

WASD provides a proxy service for the FTP scheme (prototcol). This is not (as yet) integrated into the HTTPd or cached being provided using the *proxy agent* script HT_ROOT:[SRC.MISC]FETCH.C.

The (probable) file system of the FTP server host is determined by examining the results of an FTP PWD command. If it returns a current working directory specification containing a ‘‘/’’ then it’s assumes it to be Unix(-like), if ‘‘:[’’ then VMS, if a ‘‘\’’ then DOS. Anything else is unknown and it tries to do it’s best with an uninterpreted listing.

Note that the content-type of the transfer is determined by the way the proxy server interprets the FTP request path’s ‘‘file’’ extension. This may or may not correspond with what the remote system might consider the file type to be. The default content-type for unknown files is ‘‘application/octet-stream’’ (binary). In addition, a directory listing contains three links indicated by the italicised characters, ‘‘*aid*’’. These allow the user to specify the transfer mode, text (‘‘a’’ for ASCII), binary (‘‘i’’ for image) and ‘‘d’’ for directory listing, for files with a content-type not correctly interpreted by the agent.

Supports the FTP URL ‘‘;type=a’’ (return document as plain text), ‘‘;type=i’’ (return document as binary) and ‘‘;type=d’’ (return directory listing) modifiers. If a particular site is giving problems then a ‘‘;type=debug’’ may be added, revealing the client-server FTP dialog. This may provide some insight into the problem.

Rules required in HTTPD$MAP for acting as an agent of (script for) proxy:

```
redirect ftp://* /fetch/ftp://*
pass /ftp://*
script+ /fetch/* /cgi-bin/fetch/*
```

12.4 CONNECT Serving

The *connect* service provides firewall proxying for any connection-oriented TCP/IP access. Essentially it provides the ability to tunnel any other protocol via a Web proxy server. In the context of Web services it is most commonly used to provide firewall-transparent access for Secure Sockets Layer (SSL) transactions.

The WASD connect service implements the de facto standard HTTP **CONNECT** method, described in a number of Internet Drafts.

12.4.1 Enabling CONNECT Serving

As with proxy serving in general, CONNECT serving may enabled on a per-service basis. Connect serving is enabled by appending the *connect* keyword to the particular service specification. The following example shows a non-proxy, a proxy without connect service, a connect service, and finally a proxy with connect service.

```
[Service]
http://alpha.wasd.dsto.defence.gov.au:80
http://alpha.wasd.dsto.defence.gov.au:8080;proxy
http://alpha.wasd.dsto.defence.gov.au:8081;connect
http://alpha.wasd.dsto.defence.gov.au:8082;proxy;connect
```

12.4.2 Controlling CONNECT Serving

The connect service poses a significant security dilemma when in use in a firewalled environment. Once a CONNECT service connection has been accepted and established it essentially acts as a relay to whatever data is passed through it. Therefore **any transaction whatsoever** can occur via the connect service, which in many environments may be considered undesirable.

In the context of the Web and the use of the connect service for proxying SSL transactions it may be well considered to restrict possible connections to the well-known SSL port, 443. This may be done using conditional mapping rules, as in the following example:

```
[[alpha.wasd.dsto.defence.gov.au:8080]]
pass *:443 [me:connect]
pass * "403 CONNECT only allowed to port 443." [me:connect]
```

All of the comments on the use of general and conditional mapping made in Section 12.1.3 can also be applied to the connect service.

12.5 Reporting and Maintenance

The HTTPDMON utility allows real-time monitoring of proxy serving activity. See Section 19.6.

Proxy reports and some administrative control may be exercised from the server administration menu, see Chapter 14. The information reported includes:

- some proxy serving statistics
- current cache device status
- whether cache space is available
- if a purge is in progress
- the results from the last routine and reactive purges
- the results from the last scan of the cache
- contents of the host name/address cache

The following actions can be initiated from this menu. Note that three of these relate to proxy file cache and so may take varying periods to complete, depending on the number of files. If the cache is particularly large the scan/purge **may take some considerable time**.

- generate proxy cache statistics by scanning the entire cache
- perform a routine purge
- perform a reactive purge
- purge the proxy host name/address cache

Also available from the administration menu is a dialog allowing the proxy characteristics of the *running* server to be adjusted on an ad hoc basis. This only affects the executing server, to make changes to permenant configuration the HTTPD$CONFIG configuration file must be changed.

This dialog can be used to modify the device free space percentages according to recent changes in device usage, alter the reload or purge hour list characteristics, etc. After making these changes a routine or reactive purge will automatically be initiated to reduce the space in use by the proxy cache if implied by the new settings.

12.5.1 PCACHE Utility

It is often useful to be able to list the contents of the proxy cache directory or the characteristics or contents of a particular cache file. Cache files have a specific internal format and so require a tool capable of dealing with this. The HT_ROOT:[SRC.UTILS]PCACHE.C program provides a versatile command-line utility as well as CGI(plus) script, making cache file information accessable from a browser. It also allows cache files to be selected by wildcard filtering on the basis of the contents of the associated URL or response header. For detailed information on the various command-line options and CGI query-string options see the description at the start of the source code file.

Command-Line Use

Make the HT_EXE:PCACHE.EXE executable a foreign verb. It is then possible to

- list the basic characteristics of all/selected files in the cache directory tree
- list the characteristics plus the HTTP response header of a single file
- extract the response header
- extract the response body (text, graphic, file, etc.)
- do all of the above while filtering on URL or response header contents, number of hits, when last accessed, last loaded, and last modified (in hours)

Script Use

To make the PCACHE script available to the server ensure the following line exists in the HTTP$CONFIG configuration file in the [AddType] section.

```
.HTC  application/x-script  /cgiplus-bin/pcache  WASD proxy cache file
```

The following rule needs to be in the HTTPD$MAP configuration file.

```
pass /ht_cache_root/*
```

Note

> It is also recommended to place the utility and the cache directory under some authorization control to prevent casual browsing and access of the cache contents. Something local, similar in intention to
>
> ```
> [VMS]
> /pcache/* ~webadmin,131.185.250.*,r+w ;
> /ht_cache_root/* ~webadmin,131.185.250.*,r+w ;
> ```

Once available the following is then possible.

- From a directory listing (‘‘Index Of’’) access a cache file and be presented with the following information:
  - blocks used/allocated
  - last modification date/time of the response
  - date/time the response was (re)loaded into cache
  - date/time the cache file was last accessed
  - number of time since first created the cache file has been accessed
  - the URL the cache file represents (as a link)
  - the full response header (as received from the proxied server)
  - a series of ‘‘buttons’’ allowing
    1. the cache content (response body) to be viewed (note that self-relative embedded graphics, etc., probably will not be displayed in such documents)
    2. the cache file to be VMS DUMPed

3. the cache file to be VMS ANALYZE/RMSed
4. the cache file to be VMS DELETEd

If the configuration changes described above have been made the following link will return such an index.

online hypertext link

- Have the utility generate a form providing a convenient interface to the various capabilities and filters available. If the configuration changes described above have been made the following link will return this form.

online hypertext link

- The utility's form does not have to be used. By supplying the appropriate query string components, either from a custom form or forms, or directly embedded into links, profiles, listings, deletion may be generated.

Note

Cache directory trees have the potential to become heavily populated, so the use of the script to generate listings of the cache contents could return extremely large listing documents.

12.6 Browser Configuration

The browser needs to be configured to access URLs via the proxy server. This is done using two basic approaches, manual and automatic.

- **Manual**

  Most browsers allow the configuration for access via a proxy server. This commonly consists of an entry for each of the common Web protocol schemes (''http:'', ''ftp:'', ''gopher:'', etc.). Supply the configured WASD proxy service host name and port for the HTTP scheme. This is currently the only one available. This would be similar to the following example:

```
http: www.wasd.dsto.defence.gov.au 8080
```

  To exclude local hosts, and other servers that do not require proxy access, there is usually a field that allows a list of hosts and/or domain names for which the browser should not use proxy access. This might be something like:

```
wasd.dsto.defence.gov.au,dsto.defence.gov.au,defence.gov.au
```

- **Automatic**

  At least Netscape Navigator/Communicator and Microsoft Internet Explorer (4.*n* and following) provide the facility to download a small JavaScript function for establishing proxy policy. Information on this function and it's deployment may be found at

  http://home.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html

The following is a very simple proxy configuration JavaScript function. This specifies that all URL host names that aren't full qualified, or that are in the ''defence.gov.au'' domain will be connected to directly, with all other being accessed via the specified proxy server.

```
function FindProxyForURL(url,host)
{
   if (isPlainHostName(host) ||
       dnsDomainIs(host, ".defence.gov.au"))
      return "DIRECT";
   else
      return "PROXY www.wasd.dsto.defence.gov.au:8080; DIRECT";
}
```

This JavaScript is contained in a file with a specific, associated MIME file type, ''application/x-ns-proxy-autoconfig''. For WASD it is recommended the file be placed in HT_ROOT:[LOCAL] and have a file extension of .PAC (which follows Netscape naming convention).

The following HTTPD$CONFIG directive would map the file extension to the required MIME type:

```
[AddType]
.PAC  application/x-ns-proxy-autoconfig  -  proxy autoconfig
```

This file is commonly made the default document available from the proxy service. The following example shows the HTTP$MAP rules required to do this:

```
[www.wasd.dsto.defence.gov.au:8080]
pass http://* http://*
pass / /ht_root/local/proxy.pac
pass *
```

All that remains is to provide the browser with the location from which load this *automatic proxy configuration* file. In the case of the above set-up this would be:

```
http://www.wasd.dsto.defence.gov.au:8080/
```

A template for a proxy auto-configuration file may be found at HT_ROOT:[EXAMPLE]PROXY_AUTOCONFIG.TXT.

Chapter 13

Secure Sockets Layer

This section is not a tutorial on SSL. It contains only information relating to WASD's use of it. Refer to the listed references, Section 13.4, for further information on SSL technology.

The *Secure Sockets Layer* protocol (SSL) is designed to provide privacy between two communicating applications, in the case of HTTP between the browser (client) and the HTTPd (server). It provides other capabilities, such as client authentication, but these are not considered here. SSL operates by establishing an authenticated, encrypted communication path between the two applications, ''wrapping'' the entire application protocol, such as HTTP, in a secure link, providing complete privacy for the entire transaction. In this way security-related data such as user identification and password, as well as sensitive transaction information can be effectively protected from unauthorized access.

WASD implements SSL using a freely available software toolkit known as ''OpenSSL'', see Section 13.3.

SSL functionality is not supplied with the basic WASD package. In part this is due to the relative bulk of this component, but also considers patent issues and USA export restrictions on some cryptographic technology that would impact the basic package's potential for download out of the USA. For further comment on this aspect check the OpenSSL Project's Web site (see Section 13.4). No recommendations are made by this author on the use of the **OpenSSL package** in any particular locality, except to comment that it **is in commercial and non-commercial use in many applications across across North America, Europe, Asia and Australia**, in both OpenSSL and SSLeay incarnations.

The WASD SSL component is available as a separate, easily integrated package, with OpenSSL object libraries, OpenSSL utility object modules for building executables and WASD support files. It requires no compilation, only linking. Obtain these from the same source as the main package.

SSL Overhead

SSL **adds a significant overhead** to an HTTP transaction for the following reasons.

- An initial *connection establishment*, where the client and server exchange cryptographic data and authorize the transaction.
- The transaction transfer, where all *application data* must be processed by the CPU into an encrypted stream, and back . . . very expensive processes.
- The encrypted data contains more bytes than the raw data, increasing network transfer time.
- Other miscellaneous SSL handshaking for the life of the transaction.

For these reasons **SSL HTTP is slower and has far greater impact on the server system CPU** than standard HTTP and therefore should only be used when transaction privacy is required, not as a general HTTP service. Also, if a general HTTP and an SSL HTTP service is provided on a multi-processor system, with one or other or both experiencing significant traffic, then the two services should be run in separate processes.

Interoperability

WASD SSL has been used against a wide variety of browsers and certificates. Most combinations work. Some do not, usually related to the level of encryption required by some certificates precluding export-grade browsers from connecting.

To date OpenSSL certificates, and those from Thawte and VeriSign have been deployed on WASD servers.

WASD supports both 40bit, USA export-grade encryption, as well as 128-bit, **full-strength, USA domestic-grade** encryption.

13.1 SSL Configuration

WASD SSL installation creates an OpenSSL directory in the source area, HT_ROOT:[SRC.OPENSSL-n_n_n], containing the OpenSSL copyright notice, object libraries, object modules for building executables, example certificates, and some other support files and documentation.

The example HTTPd startup procedure already contains support for the SSL executable. If this has been used as the basis for startup then only the respective boolean needs to be altered to use the SSL rather than the standard executable. The SSL executable supports both standard HTTP services (ports) and HTTPS services (ports). These must be configured using the [service] parameter. SSL services are distinguished by specifying ‘‘https:’’ in the parameter. The default port for an SSL service is 443.

The following example illustrates creating two standard HTTP services, one on the default port 80, another on port 8000, and two SSL services, the first on the default port 443, the second on port 8443.

```
[Service]
alpha.wasd.dsto.defence.gov.au
alpha.wasd.dsto.defence.gov.au:8000
https://alpha.wasd.dsto.defence.gov.au
https://alpha.wasd.dsto.defence.gov.au:8443
```

The ‘‘/SSL’’ qualifier controls which version(s) of the SSL protocol the server will support; ‘‘2’’, ‘‘3’’ or ‘‘23’’ (i.e. versions 2 and 3, also the default). Using /NOSSL disables the SSL functionality of an SSL executable.

The one further requirement of an SSL server is a *certificate*. By default this is located using the HTTPD$SSL_CERT logical name during startup, however if required, each SSL service can have an individual certificate configured against it using the syntax shown in this example.

```
[Service]
https://alpha.wasd.dsto.defence.gov.au;cert=ht_root:[local]alpha.pem
https://beta.wasd.dsto.defence.gov.au;cert=ht_root:[local]beta.pem
https://gamma.wasd.dsto.defence.gov.au;cert=ht_root:[local]gamma.pem
```

SSL Access Control

When authorization is in place (see Chapter 11) access to username/password controlled data/functionality benefits enormously from the privacy of an authorization environment inherently secured via the encrypted communications of SSL. SSL may be used as part of the site’s access control policy, as whole-of-site, see Section 11.1, or on a per-path basis, see Conditionals and Access Restriction List.

SSL Quick-Start

If the basic WASD v6.*n* package is installed, configured and performing satisfactorily the following steps give a quick outline for support of SSL.

1. The ZIP archive will contain brief installation instructions. Use the following command to read this and any other information provided.

   ```
   $ UNZIP -z device:[dir]archive.ZIP
   ```

2. UNZIP the WASD SSL package.

   ```
   $ SET DEFAULT HT_ROOT:[000000]
   $ UNZIP "-V" device:[dir]archive.ZIP
   ```

3. Once UNZIPped it will be necessary to link the executables. This should be performed using the UPDATE.COM procedure.

   It assumes a *vanilla* setup, using the standard directories and account environment described in this document. All sections prompt before performing any action and default to ‘‘no’’. The SSL parameter indicates the SSL components should be processed.

   ```
   $ @UPDATE SSL
   ```

4. Once linked the UPDATE.COM procedure will prompt for permission to execute the demonstration/check procedure.

   It is also possible to check the SSL package at any other time using the FREEWARE_DEMO.COM procedure. It is necessary to specify that it is to use the SSL executable. Follow the displayed instructions.

   ```
   $ @HT_ROOT:[000000]FREEWARE_DEMO SSL
   ```

5. Modification of server startup procedures should not be necessary. If an SSL image is detected during startup it will be used in preference to the standard image.
6. Modify the HTTPD$CONFIG configuration file to specify an SSL service. For example the following creates both a standard HTTP service on the default port 80 and an SSL service on the default port 443

   ```
   [Service]
   alpha.dsto.defence.gov.au
   https://alpha.dsto.defence.gov.au
   ```

7. Shutdown the server completely, then restart.

   ```
   $ HTTPD /DO=EXIT
   $ @HT_ROOT:[LOCAL]STARTUP
   ```

8. To check the functionality (on default ports) access the server via
   - **Standard HTTP**

     ```
     http://host.domain/
     ```

   - **SSL HTTP**

     ```
     https://host.domain/
     ```

9. Once the server has been proved functional with the example certificate it is recommended that a server-specific certificate be created using the WASD support procedures described in Section 13.2. This may then be used by placing it in the appropriate local directory, assigning the WASD_SSL_CERT symbol appropriately before startup.

13.2 Certificate Management

This is not a tutorial on certificates and their management. Refer to the listed references, Section 13.4, for further information on this aspect. It does provide some basic guidelines.

Certificates associate a public cryptographic key with the identity of the certificate holder, usually an individual or server. It includes a distinguishing name, identification and signature of the certificate authority (CA, the issuer and guarantor of the certificate), and the period for which the certificate is valid, possibly with other, additional information.

The two types of certifcates of interest here should not be confused.

1. **CA -** The Certificate Authority identifies the *authority*, or organization, that issues a server certificate.
2. **Server -** Identifies a particular end-service. It's value as an guarantee of identity is founded in the *authority* of the organization that issues the certificate. It is the certificate specified to the server at startup.

The server uses a certificate to establish it's identity during the initial phase of the SSL protocol exchange. Each server should have a unique certificate. An example certificate is provided with the WASD SSL package. This is the default used by the example startup procedure and will suffice to demonstrate the server's functionality. If a ‘‘live’’ SSL site is required a unique certificate issued by a third-party Certificate Authority is desirable.

A working alternative to obtaining one of these certificates is provided by the WASD support DCL procedures, which are quick hacks to ease the production of certificates on an ad hoc basis. In all cases it is preferable to directly use the utilities provided with OpenSSL, but the documentation tends to be rather sparse.

The first requirement may be a tailored ‘‘Certificate Authority’’ certificate. As the Certificate Authority is non-authoritative (not trying to be too oxymoronic, i.e. not a well-known CA) these certificates have little value except to allow SSL transactions to be established with trusting clients. More commonly ‘‘Server Certificates’’ for specific host names are required. Procedures and brief notes in the HT_ROOT:[SRC.OPENSSL-n_n_n.WASD] directory provide some support for these activities.

Loading Authority Certificates

CA certificates can be loaded into browsers to allow sites using that CA to be accessed by that browser without further dialog. Both Netscape Navigator (v3.*n* & v4.*n*) and MS Internet Explorer (v4.*n*) automatically invokes a server certificate load dialog when it encounters a site using a valid but unknown server certificate.

A manual load is accomplished by requesting the certificate in a format appropriate to the particular browser. This triggers a browser dialog with the user to confirm or refuse the loading of that certificate into the browser Certificate Authority database.

To facilitate loading CA certificates into a browser ensure the following entries are contained in the HTTP$CONFIG configuration file:

```
[AddIcon]
/httpd/-/binary.gif  [BIN]  application/x-x509-ca-cert

[AddType]
.CRT  application/x-x509-ca-cert  -  DER certifcate (MSIE)
.PEM  application/x-x509-ca-cert  -  Privacy Enhanced Mail certificate
```

Then just provide a link to the required certificate file(s), and click.

Navigator should be able to load using either certificate format. MSIE v3.*n* will load and report on the ‘‘.CRT’’ certificate quite contentedly, but then will not allow it to be used because it does not represent a well-known Certficate Authority. MSIE v4.*n* seems able to use the ‘‘.CRT’’ certificate.

Changing Server Certificates

If a site’s server (or CA certificate) is changed and the server restarted any executing browsers will probably complain (Netscape Navigator reports an I/O error). In this case open the browser’s certificate database and delete any relevant, permanently stored certificate entry, then close and restart the browser. The next access should initiate the server certificate dialog, or the CA certificate may be explicitly reloaded.

13.3 OpenSSL

WASD implements SSL using a freely available software toolkit supported by the OpenSSL Project, and **in particular the VMS port package** supported by Richard Levitte (levitte@lp.se) and Robert Byer (ByerRA@aol.com). OpenSSL is a continuing development of the ‘‘SSLeay’’ toolkit (pronounced ‘‘S-S-L-E-A-Y’’, i.e. all letters spelt), authored by Eric Young and Tim Hudson. OpenSSL is not a GNU-licensed package, but does makes unrestricted commercial and non-commercial use available. See references below.

If the OpenSSL component of WASD is installed it be found in HT_ROOT:[SRC.OPENSSL-n_n_n].

It is always preferable to move to the latest support release of OpenSSL as known bugs in previous versions are progressively addressed (ignoring the issue of new bugs being introduced ;^).

13.4 SSL References

The following provide a starting-point for investigating SSL and OpenSSL further (verified available at time of publication).

- http://www.openssl.org/
  OpenSSL Project. This site is the prime source for the full toolkit, documentation, related links, news and support via mailing lists, etc.
- http://www.free.lp.se/openssl/
  OpenSSL on the VMS platform *(Richard Levitte)*.
- http://www.free.lp.se/openssl/docs/
  Certificate management, building the full package, etc., *(Robert Byer)*.
- http://www.mikom.csir.co.za/SSLeay/
  Enabling Network Security with SSLeay.
- http://www.ultranet.com/~fhirsch/Papers/cook/ssl_cook.html
  An excellent overview of managing certification with OpenSSL (SSLeay).
- http://www.consensus.com/security/ssl-talk-faq.html
  SSL-Talk discussion list FAQ. This is an excellent resource, although it indicates it will be superceded during 1999. Most probably links or redirections will be provided to any replacement.
- http://www.fortify.net/
  Information on restoring *export* Netscape Navigator and Communicator to **full-strength** encryption (Unix and Win32).

Chapter 14

Server Administration

All server configuration files, with the execption of the authentication databases, are plain text and may be modified with any prefered editor. However the majority of these can also be administered on-line through a browser.

A menu of server administration functions is provided by the server.

online hypertext link

In addition a Web **Upd**ate facility allows some administration of file system portions of the Web. See Chapter 18.

If authorization is in use (see Chapter 11) it is recommended that the path to these reports be controlled via authentication and authorization, using both host and username restrictions, similar to the following:

```
[WHATEVER-REALM]
/httpd/-/admin/*  host.ip.addr,~WebMaster,~WhoEverElse,r+w
```

If a full authorization environment is not required but administration via browser is still desired restrict access to browsers executing on the server system itself, using an appropriate SYSUAF-authenticated username. Provision of a VMS account for server administration only is quite feasable, see Section 11.9.3.

```
[VMS]
/httpd/-/admin/*  #localhost,~username,r+w
```

If SSL is in use (see Chapter 13) then username/password privacy is inherently secured via the encrypted communications. To restrict server administration functions to this secure environment add the following to the HTTPD$MAP configuration file:

```
/httpd/-/admin/*  "403 Access denied."  ![sc:https]
```

When using the *revise* capability of the administration menu it is necessary to comply with all the requirements for Web update of files. This is discussed in general terms in Chapter 18. Revision of server configuration files requires path permissions allowing write access for the username(s) doing the administration, as well as the required ACL on the target directory (in the following example HT_ROOT:[LOCAL]).

```
[VMS]
/httpd/-/admin/*  #localhost,~username,r+w
/ht_root/local/*  #localhost,~username,r+w
```

It is possible to allow general access to the administration menu and reports while restricting the ability to intiate server actions such as a restart! Using the WORLD realm against the path is necessary, for the obvious security reason, the server administration module will not allow itself to be used without an authenticated username, provided as a pseudo-authenticated ‘‘WORLD’’.

```
[VMS]
/httpd/-/admin/control/*  #localhost,~username,r+w
[WORLD]
/httpd/-/admin/* r
```

14.1 Breaking-in To The Server!

The server’s authentication/authorization environment can be circumvented under special circumstances. **Note that this is not possible without administrator action and persists only as long as the administrator actually performs the action!**

This provision exists for only four foreseeable situations:

1. ad hoc server configuration
2. initial configuration of the authentication environment
3. loss of the administrator’s password
4. demonstration/experimentation purposes

Method

If performing the initial authentication configuration

make sure the HT_AUTH logical is correctly defined when planning to use HTA databases.

Manually start a new instance of the server on a non-standard port using the /PROMISCUOUS qualifier. This can be done at the command line. There is now a security hole. Example:

```
$ HTTPD = "$HT_EXE:HTTPD.EXE"
$ SPAWN /WAIT HTTPD /NOLOG /SERVICE=8088 /PROMISCUOUS
```

This server with then allow access using any username/password combination. Even if not completely paranoid it’s perhaps a good idea to append a specific password to the qualifier, the server will then only authenticate to that.

```
$ SPAWN /WAIT HTTPD /NOLOG /SERVICE=8088 /PROMISCUOUS=VERYCAUTIOUS
```

Access this instance of the server with a browser and use the server administration menu.

```
http://host:port/httpd/-/admin/
```

It is now possible to review server-generated reports, check rule mappings, create databases, enter username details, change the administrator’s password to something known, etc., etc. Shutdown that server again (ctrl-Y) and the security hole disappears.

Note that if a site authorization rule already maps the administration menu path or configuration file path the server may report duplicate path errors. This is due to the /PROMISCUOUS startup effectively introducing the following two rules, ensuring the menu and configuration files can be accessed regardless of anything already in or missing from the authorization file.

```
[PROMISCUOUS]
# access to the server administration menu
/httpd/-/admin/* r+w
# access to write into the "usual" location for configuration files
/ht_root/local/* r+w
```

Other rules may be placed in the local configuration file for the [PROMISCUOUS] realm specifically for use during this mode of access. They will not apply during normal server operation. Be sure they are placed before other rules in the file.

14.2 HTTPd Server Reports

The server provides a number of internally generated reports.

It is possible to use this facility standalone, without configuring authorization, see Section 14.1.

14.2.1 Statistics

Server process up-time, CPU-time and other resources consumed, number of connections processed, number of requests of each HTTP method, type of processing involved (HTTPd module used), number of bytes processed, etc.

14.2.2 Configuration

A tabular summary of the server's current configuration. This is a convenient method for viewing the information from the HTTPD$CONFIG file.

14.2.3 Message

A tabular report of the server's current message database, multiple languages shown if configured that way.

14.2.4 Mapping

All loaded mapping rules and any cached USER rule paths. A selector allows rules applying only to one particular virtual server to be displayed.

14.2.5 Path Authorization

If authorization is in use (see Chapter 11) this report lists the paths with associated authorization and access control.

14.2.6 User Authentication

List any users that have been authorized since the server was last started, the realm authorized from, the group it applies to (if any), and what the user's capabilities are (allowed HTTP methods). A time-stamp and counters provide additional information.

14.2.7 Cache

Allows monitoring of cache behaviour and performance, as well as the files currently in the cache. For further information see Chapter 16.

14.2.8 DCL Scripting

Provides some DCL, CGI and CGIplus scripting information.

- DCL module statistics (same information as displayed in the server statistics report). These are cumulative for the entire life of the system (unless zeroed).
- Subprocess information shows how many actual subprocesses exist at the time of the report, as indicated by the PID and bolded, non-zero liftime (in minutes). The *soft-limit* specifies how many CGIplus scripts are allowed to continue existing before the least used is deleted and the *hard-limit* show how many subprocesses may actually exist at any one time (the margin allows for subprocess deletion latency). A count of how many times the CGIplus subprocesses have been explicitly purged (button available on this report page). The *life-time* of zombie processes (in minutes, zero implying use of zombies is disabled) and the number that have been purged due to expiry. CGIplus subprocess life-time (in minutes, zero implying indefinite), the number purged due to life-time expiry and the number of CGIplus subprocesses that the server has actually purged (deleted) to maintain the soft-limit margin specified above.
- Each of the allocated subprocess data structures is listed. There may be zero up to hard-limit items listed here depending on demand for DCL activities and the life of the server. Items with a PID shown indicate an actual subprocess existing. This can be a zombie subprocess or a CGIplus subprocess. If no subprocess is indicated then the other information represents the state the last time the item's associated subprocess completed. Information includes the script (URL-style path) or DCL command, total count of times the item has been used and the last time it was. The zombie count indicates the number of time the same subprocess finished a request and entered the *zombie* state. The CGIplus column indicates it is/was a CGIplus script and shows the total number of times that particular script has been/was used. If the subprocess is currently in use the client information show the client host name.
- If any subprocesses are associated with any data structure a *purge* button is provided that forces all subprocesses to be deleted. This can be useful if a new script image is compiled and it is required all scripts now use this. If a script is currently processing a request the subprocess deletion occurs when that processing is complete. The purge button **does not force** a subprocess to delete, so a second button **forces** all subprocesses to delete immediately. This can be used to forceably clear errant scripts, etc., but be warned script processing is indiscrimately stopped!

14.2.9 DECnet Scripting

DECnet module information shows totals for DECnet scripting usage and the DECnet connection list.

This list will grow, up to the specified configuration maximum, as conconurrent scripting demand occurs. Maintained connections are indicated by the bolded, non-zero lifetime (in minutes). When this reaches zero the task is disconnected. The current/last task for that connection is indicated, along with the number of times the connection was reused and a total number of uses for that list item.

Purge and *force* buttons allow current links to be broken after request completion or forcibly disconnected.

14.2.10 Memory

Provides a report and does an integrity check on each of the Virtual Memory (VM) zones employed by the WASD HTTPd.

14.2.11 Process

Lists all processes on the current system owned by the server account. From this list a process can be selected to have a ‘‘SHOW PROCESS /ALL’’ performed on it, displayed on a report page.

14.2.12 Proxy

If proxy serving is enabled a report providing statistics on the various HTTP methods used, network and cache traffic, cache reads and writes, requests not cachable, and host name lookup are provided. This may used to help guage the effectiveness of the cache.

14.2.13 Request

Lists current requests (always shows at least your own connection accessing this report :^) and if enabled by configuration a history list of the most recent requests (enabled by the configuration parameter [RequestHistory]). Current requests may selected for *one-shot* WATCHing from this page (see Chapter 15).

Another diagnostic tool is available from the same link, the *Peek Report*. A space leading the request number (immediately preceding the WATCH link) provides a page reporting selected internal fields and data structures of the request. This is primarily intended as a problem investiagtion and development tool, and will be of limited value without an understanding of server internals.

14.2.14 SSL

The SSL report lists counts of the number of SSL transactions initiated and completed, along with session cache statistics for the currently connected SSL service. It also lists the ciphers available and current session information.

14.2.15 Activity

Provide a graphical *snapshot* of server activity of a given period.

This is dynamic data, held in memory only, and so is current only from the latest server startup. It is enabled by the configuration parameter [ActivityDays]. The administration menu provides several, represented as a period of hours before the present time. Number of requests and bytes sent to the client are represented by a histogram with respective means for each during the period by a line graph. A *greyed* area indicates no data available for that time (i.e. before the latest server startup, or in the future).

Activity data is accumulated on a per-minute basis. This is the maximum granularity of any report. When reports are selected that can display less than this one minute granularity (i.e. with periods greater than four hours) the value shown is the **peak** of the number of minutes sampled for display. This better represents the load on the server than would a mean of those samples.

For browsers recognised as capable (with v4.3 of the server these are Netscape Navigator 3.0ff and Microsoft Internet Explorer 3.02ff) this report is JavaScript-enabled, and if in focus the browser refreshes itself at an interval appropriate to the reporting period. If not in focus, the report is automatically refreshed when the browser is brought back into focus.

The graph is an image map, various regions of which allow the selection of other reports with different periods or durations. This allows previous periods to be examined at various levels of detail using the graph for navigation. Various sections may have no mapping as appropriate to the current report.

The following example shows the layout for a four hour report. The upper and lower sections have distinct functions. The middle 50% of the upper section allows the same end time (most commonly the current hour) to be examined over twice the current period, in this case it would be over eight hours. The left 25% allows the previous fours hours to be viewed (if such data exists), and for non-current reports the right 25% allows the next four hours to be viewed. The lower half can be divided into sections representing hours or days depending on the period of the current report. This allows that period to be viewed in greater detail. For single hour reports this section, of course, is not mapped.

Remember that the URL of the mapped section will be displayed in the status bar of the browser. As the URL contains time components it is not a difficult task to decipher the URL displayed to see the exact time and period being selected.

```
+-----------+-----------------------+-----------+
| previous  |     same end time     | next      |
| 4 hours   |   double the period   | 4 hours   |
|           |                       |           |
|           |                       |           |
+-----------+-----------+-----------+-----------+
| hour 1    | hour 2    | hour 3    | hour 4    |
| period: 1 | period: 1 | period: 1 | period: 1 |
|           |           |           |           |
|           |           |           |           |
+-----------+-----------+-----------+-----------+
```

14.2.16 WATCH

This report provides an online, real-time, in-browser-window view of request processing on the **running server**. See Chapter 15 for details.

14.3 HTTPd Server Revise

The server provides a limited configuration revision facility.

1. **Configuration -** A form-driven interface allows the current configuration of the server to be altered on-line. This configuration may then be saved to the on-disk file and then the server could be restarted using the new parameters. The source of the current configuration can be either the server itself (from it's volatile, in-memory parameters) or from the on-disk configuration file. In addition it is possible to directly edit and update the on-disk file.
2. **Message -** No form-driven interface is currently available for changing the messages. However it is possible to directly edit and update the on-disk file. The server can then be restarted to use the modified database (see Section 14.4).
3. **Mapping -** No form-driven interface is currently available for changing the mapping rules. However it is possible to directly edit and update the on-disk file. The mapping rules could then be reloaded, changing the current server rules (see Section 14.4).
4. **Path Authorization -** No form-driven interface is currently available for changing the path authorization configuration. However it is possible to directly edit and update the on-disk file. The path authorization directives could the be reloaded, changing the current server authorization (see Section 14.4).
5. **User Authentication -** User authentication comprises a number of dialogues that allow the authentication databases to be administered. These include:
    - creating databases
    - deleting databases
    - accessing databases for administering usernames
    - listing usernames within databases
    - adding usernames
    - deleting usernames
    - modifying username permissions and other data
    - reseting in-server (cached) authentication information

    Chapter Chapter 11 covers authentication detail.

14.4 HTTPd Server Action

The server allows certain run-time actions to be initiated. Many of these functions can also be initiated from the command line, see Section 5.3.

Control Section

1. **Server Restart/restartNOW/Exit/exitNOW - Caution ... no confirmation!** The difference between restart/exit and restartNOW/exitNOW is the former waits for any current requests to be completed, while the latter does it immediately regardless of any current connections.
2. **Logging On/Off/Flush -** The HTTPD$LOG logical must be configured to allow access logging to be enabled and disabled from this menu.
3. **Caching On/Off/Purge -** Caching may be enabled and disabled in an ad hoc fashion using these controls. When being disabled after being enabled all previous data is retained. If subsequently reenabled that data is then again available for use. This allows convenient assessment of the subject or even object benefits on the cahing. If purged all entries in the cache are removed.

Configuration *Action* Section

1. **Statistics Zeroed -** All counters are zeroed (except the *number-of-times-zeroed* counter!)
2. **Mapping Rules Reload -** Reloads the path mapping rules from the on-disk file into the running server, clears the user SYSUAF mapping cache.

   Caution! If changing CGIplus script mapping it is advised to restart the server rather than reload. Some conflict is possible when using new rules while existing CGIplus scripts are executing.
3. **Path Authorization Reload -** Reloads the path authorization directives from the on-disk file into the running server.
4. **User Authentication Cache Purge -** For efficiency reasons authenticated user information is cached for a limited period within the running server. All this cached information may be completely purged using this action, forcing subsequent requests to be reauthenticated from the on-disk database.

Chapter 15

WATCH Facility

The WATCH facility is a powerful adjunct in server administration. From the administration menu it provides an **online, real-time, in-browser-window view of request processing in the running server**. The ability to observe live request processing on an ad hoc basis, without changing server configuration or shutting-down/restarting the server process, makes this facility a great configuration and problem resolution tool. It allows (amongst other uses)

- assessment of mapping rules
- assessment of authorization rules
- investigation of request processing problems
- general observation of server behaviour

A single client per server process can access the WATCH facility at any one time. It can be used in one of two modes.

- As a *one-shot*, one-off WATCH of a particular request. This is available from the *Request Report* page of the Administration Menu. In this case the single indicated request is tagged to be WATCHed in all categories (see below) for the duration of the request (or until the client stops WATCHing).
- As described in the following chapter the server and all new requests being processed are candidates for being WATCHed. Categories are selected before intiating the WATCH and the report can be generated for a user-specified number of seconds or aborted at any time using the browser's *stop* button.

15.1 Event Categories

An *event* is considered any significant point for which the server code has a reporting call provided. These have been selected to provide maximum information with minimum clutter and impact on server performance. Obvious examples are connection acceptance and closure, request path resolution, error report generation, network reads and writes, etc. Events are collected together into groupings to allow clearly defined areas of interest to be selected for reporting. The report menu provides for the inclusion of any combination of the following categories.

Request

- **Processing -** Each major step in a request's progress. For example, path resolution and final response status.
- **Header -** Provides the HTTP request header as a section of blank-line terminated text. If an ''Authorization:'' line is included any non-encrypted password information is masked.
- **Body -** In the case of HTTP POST or PUT methods any request body is displayed. It will generally be URL-encoded but otherwise plain-text.

Response

- **Processing -** Each major step in generating a response to the request. These generally reflect calls to a major server module such as file CACHE, FILE access, INDEX-OF, SSI processing, etc. One or more of these events may occur for each request. For instance a directory listing will show an INDEX-OF call and then usually a FILE call as any read-me file is accessed.
- **Header -** The blank-line terminated HTTP header to the response. Only server-generated headers are included. Scripts that provide a full HTTP stream do not have the header explicitly reported. The response body category must be enabled to observe these (indicated by a STREAM notation).
- **Body -** The content of the response. This is provided as a hexadecimal dump on the left and with printable characters rendered on the right, 32 bytes per line. Some requests also generate very large responses which will clutter output. Generally this category would be used when investigating specific request response body problems.

General

- **Connection -** Each TCP/IP connection acceptance and closure. The connect shows which service the request is using (scheme, host name and port).
- **Path Mapping -** This, along with the authorization report, provides one of the most useful aspects of the WATCH facility. It comprises an event line indicating the path to be mapped (it can also show a VMS file specification if a *reverse-mapping* has been requested). Then as each rule is processed a summary showing current path, match ''Y''/''N'' for each path template and any conditional, then the result and conditional. Finally an event entry shows the resulting path, VMS file specification, any script name and specification resolved. The path mapping category allows the administrator to directly assess mapping rule processing with live or generated traffic.
- **Authorization -** When authorization is deployed this category shows the rules examined to determine if a path is controlled, any authentication events in assessing username and password, and the consequent group, user and request capabilities (read and/or write) for that path. No password information is displayed.
- **Error -** The essential elements of a request error report are displayed. This may include a VMS status value and associated system message.
- **CGI -** This category displays the generated CGI variable names and values as used by subprocess and DECnet scripting, and by SSI documents.

- **DCL -** Debugging scripts can sometimes present particular difficulties. This category may help. It reports on all input/output streams with the subprocess (SYS$INPUT, SYS$OUTPUT, SYS$COMMAND, CGIPLUSIN).
- **DECnet -** For the same reason as above this category reports all DECnet scripting input/output of the DECnet link. In particular, it allows the observation of the OSU scripting protocol.

Network

- **Activity -** For each raw network read and write the VMS status code and size of the I/O is recorded.
- **Data -** For each raw network read or write the contents are provided as a hexadecimal dump on the left and with printable characters rendered on the right, 32 bytes per line.

Other

- **Timer -** Timers control the maximum periods each stage in request processing may procede before the server aborts processing. Input, output and keep-alive timers are reported.
- **Logging -** Access logging events include log open, close and flush, as well as request entries.
- **SSL -** If the Secure Sockets Layer image is in use this category provides a indication of high-level activity.

Proxy

- **Processing -** Each major step during the serving of a proxied request.
- **Request Header -** The proxy server rebuilds the request originally received from the client. This category shows that rebuilt request, the one that is sent to the remote server. If an ‘‘Authorization:’’ line is included any non-encrypted password information is masked.
- **Request Body -** In the case of HTTP POST or PUT methods any request body is displayed. This is provided as a hexadecimal dump on the left and with printable characters rendered on the right, 32 bytes per line.
- **Response Header -** The blank-line terminated HTTP header to the response from the remote, proxied server.
- **Response Body -** The content of the response sent from the remote server. This is provided as a hexadecimal dump on the left and with printable characters rendered on the right, 32 bytes per line.
- **Cache -** When proxy caching is enabled this category provides information on cache reading (serving a request from cache) and cache loading (writing a cache file using the response from a remote server). It will provide a reason for any request or response it does not cache, as well as report errors during file processing.

- **Cache Maintenance -** This category is not related to request processing. It allows routine and reactive cache purging activities to be watched.

15.2 Request Filtering

By default all requests to all services are WATCHed. Fine control may be exercised over exactly which requests are reported, allowing only a selected portion of all requests being processed to be concentrated on, even on a live and busy server. This is done by *filtering* requests according the following criteria.

- **Client -** The originating host name or address. Unless server DNS host name resolution is enabled this must be expressed in dotted-decimal notation.
- **Service -** The service connected to. This includes the *scheme* of the service (i.e. ''http:'', ''https:''), the host name (real or virtual), and the port. The host name is the *official* name of the service as reported during server startup. As the port number is a essential part of the service specification it must always be explicitly supplied or wildcarded.
- **Path -** The request path.

These filters are controlled using fully-specified or wildcarded strings. Requests that do not match the filter are not reported. In the case of originating client and destination service, requests are eliminated before ever appearing in the report. Path filtering is slightly different, requiring some request processing before the path can be determined. Depending on the categories selected this may result in some events begin displayed. It will be eliminated, with an accompanying explanatory message, as soon the path has been determined.

The following examples are grouped in the same order as the categories listed above; client, service and path.

```
alpha.wasd.dsto.defence.gov.au
*.wasd.dsto.gov.au
131.185.250.202
131.185.250.*

beta.wasd.dsto.defence.gov.au:8000
beta.wasd.dsto.defence.gov.au:*
http://*
https:*
*:80

/ht_root/src/*
/cgi-bin/*
/web/*/cyrillic/*
```

15.3 Report Format

The following example illustrates the format of the WATCH report. It begins with a six-line heading. The first two record the date, time and official server name, with underline. The third provides the WASD server version. The fourth and fifth provide a list of the categories being recorded and the filters in use. The sixth, the column headings described as follows:

- **time** the event was recorded
- the **module** name of the originating source code

- the **line** in the code module
- a unique **item** number for each thread being WATCHED
- event **category** name
- free-form, but generally interpretable **event** data

Note that some items also include a block of data. The request header category does this, providing the blank-line terminated text comprising the HTTP header. Rule mapping also provides a block of information representing each rule as it is interpreted. Generally WATCH-generated information can be distinguished from other data by it's uniform format and delimiting vertical bars. Initiative and imagination is sometimes required to interpret the free-form data but a basic understanding of HTTP serving and a little consideration is generally all that is required to deduce the essentials of any report.

```
14-OCT-1999 18:19:45  WATCH REPORT  delta.wasd.dsto.defence.gov.au:80
---------------------------------------------------------------------
HTTPd-WASD/6.1.0 OpenVMS/AXP Digital-TCPIP SSL (13-OCT-1999 17:45:10.17 VMS 7.1-2 DECC 5069
UCX-TCP/IP UCX$IPC_SHR UCX V4.2-21 (18-NOV-1997 13:00:37.31)
OpenSSL 0.9.4 09 Aug 1999 (8-SEP-1999 14:25:01.01)
AlphaServer DS20 500 MHz VMS V7.1-2
$ HTTPD /PRIORITY=4 /SYSUAF=ID
Watching: connect, request, request header, response, response header, mapping (219)
Client: "*" Service: "*" Path: "*"
|Time_______|Module__|Line|Item|Category|Event...|
|18:20:46.91 NET      1626 0001 CONNECT  ACCEPT 131.185.250.108,47111 on https://131.185.25
|18:20:47.00 REQUEST  1211 0001 REQUEST  HEADER 285 bytes|
GET /ht_root/ HTTP/1.0
If-Modified-Since: Thursday, 07-Oct-99 03:51:23 GMT; length=4644
Connection: Keep-Alive
User-Agent: Mozilla/3.0Gold (X11; I; OSF1 V4.0 alpha)
Pragma: no-cache
Host: delta.wasd.dsto.defence.gov.au
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*

|18:20:47.00 NET      1148 0001 CONNECT  VIRTUAL delta.wasd.dsto.defence.gov.au:443|
|18:20:47.00 REQUEST  2270 0001 REQUEST  GET /ht_root/|
|18:20:47.00 MAPURL   0312 0001 MAPPING  PATH /ht_root/|
/ht_root/  ..  REDIRECT  /*.*.htmlx  /*.htmlx?httpd=ssi&__part=*
/ht_root/  ..  PASS      /httpd/-/change/*  /httpd/-/change/*
/ht_root/  ..  MAP       /htroot/*  /ht_root/*
/ht_root/  ..  SET       /web/*   stmLF
/ht_root/  ..  SET       /ht_root/*   stmLF
/ht_root/  ..  MAP       /httpd-internal-icons/*  /httpd/-/*
/ht_root/  ..  PASS      /ht_root/runtime/*  /ht_root/runtime/*
/ht_root/  ..  PASS      /*/-/*  /ht_root/runtime/*/*
/ht_root/  Y-  PASS      /ht_root/*
|18:20:47.00 MAPURL   0335 0001 MAPPING  RESULT /ht_root/ /ht_root/ HT_ROOT:[000000] - -|
|18:20:47.00 CACHE    0604 0001 RESPONSE CACHE DSA811:[HT_ROOT.][000000]HOME.HTML|
|18:20:47.00 NET      2427 0001 RESPONSE HEADER 231 bytes|
HTTP/1.0 200 Success
Server: HTTPd-WASD/6.1.0 OpenVMS/AXP Digital-TCPIP SSL
Date: Thu, 14 Oct 1999 01:42:46 GMT
Last-Modified: Thu, 07 Oct 1999 03:51:23 GMT
Content-Type: text/html; charset=ISO-8859-1
Content-Length: 4644
```

```
|18:20:47.00 REQUEST  0407 0001 REQUEST  STATUS 200 rx:457 tx:5084 bytes 0.0000 seconds|
|18:20:47.00 REQUEST  0252 0001 CONNECT  KEEP-ALIVE 131.185.250.108,47111|
|18:20:52.06 NET      1502 0001 CONNECT  CLOSE 131.185.250.108,47111|
```

15.4 Usage Suggestions

The following provides a brief explanation on the way WATCH operates and any usage implications.

A single client may be connected to the WATCH facility at any given time. When connecting the client is sent an HTTP response header and the WATCH report heading lines. The request then remains connected until the WATCH duration expires or the client overtly aborts the connection. During this period the browser behaves as if receiving a sometimes very slow, sometimes stalled, plain-text document. As the server processes WATCHable events the text generated is sent to the WATCH-connected client.

If the connection is aborted by the user some browsers will consider document retrieval to be incomplete and attempt to reconnect to the service if an attempt is made to print or save the resulting document. As the printing of WATCH information is often quite valuable during problem resolution this behaviour can result in loss of information and generally be quite annoying. Appropriate use of the duration selector when requesting a report can work around this, as at expiry the *server* disconnects, browsers generally interpreting this as legitimate end-of-document (when no content-length has been specified).

During report processing some browsers may not immediately update the on-screen information to reflect received data without some application activity. If scroll-bars are present on the document window manipulating either the horizonal or vertical slider will often accomplish this. Failing that minimizing then restoring the application will usually result in the most recent information being visible.

Browser *reload/refresh* may be used to restart the report. A browser will quite commonly attempt to remain at the current position in the document, which with a WATCH report's sustained but largely indeterminate data stream may take some time to reach. It is suggested the user ensure that any vertical scroll-bar is at the beginning of the current report, then refresh the report.

Selecting a large number of categories, those that generate copious output for a single event (e.g. response body) or collecting for extended periods can all result in the receipt of massive reports. Some browsers do not cope well with documents megabytes in size.

WATCH reports are written using blocking I/O. This means when large bursts of data are being generated (e.g. when WATCHing network data, response bodies, etc.) significant granularity may be introduced to server processing. Also if the WATCH client fails or blocks completely server processing could halt completely! (This has been seen when WATCHing through a firewall.)

When supplying WATCH output as part of a problem report please ZIP the file and include it an an e-mail attachment. Mailers often mangle the report format making it difficult to interpret.

15.5 Command-Line Use

Although intended primarily as a tool for online use WATCH can be deployed at server startup with a command-line qualifier and provide report output to the server process log. This is slightly more cumbersome than the Web interface but may still be useful in some circumstances. Full control over event categories and filters is possible.

- **/NOWATCH** Disables the use of the online WATCH facility.
- **/WATCH=** Enables the server WATCH facility, dumping to standard output (and the server process log if detached). When in effect the online facility is unavailable. The string supplied to the qualifier may comprise four comma-separated components. Only the first is manadatory. Stated order is essential. It will probably be necessary to enclose the complete string in quotation marks.

  1. integer representing the categories to be reported
  2. client filter
  3. service filter
  4. path filter

  The category integer must be the bitwise-OR of the constants found in the ADMIN.H source code header file. The end of the category list in the on-line WATCH report header contains a parenthesized integer. This is the number representing the enabled categories for that report and can be used for those same categories from the command line.

  The following examples illustrate the command-line WATCH specification.

  ```
  /NOWATCH
  /WATCH=88
  /WATCH="88,*,*,/cgi-bin/*"
  ```

Chapter 16

Cache

WASD HTTPd provides an optional, configurable, monitorable file data and revision time cache. File data, so that requests for documents can be fulfilled without reference to the underlying file system, potentially reducing request latency and more importantly improving overall server performance and system impact, and file revision time, so that requests specifying an ''If-Modified-Since:'' header can also benefit from the above.

Note that it is a file-system cache. Only documents generated from the file system are cached, not from any potentially dynamic sources, such as scripts, directory listings, SSI documents, etc. The reason should be obvious. Only the file system provides a reliable mechansim for ensuring the validity of the cached data (i.e. has the original changed in some way since loaded?)

Files are cached according to **mapped** path (not necessarily the same path supplied with the request) and not by the file name represented by any path. This is a design decision targeted at avoiding any access to RMS before searching the cache. For example, the ambiguous reference to the directory

```
/ht_root/
```

may result in the following file being accessed (due to home page resolution)

```
HT_ROOT:[000000]HOME.HTML
```

and the contents returned to the client and consequently cached. Each time the path ''/ht_root/'' is subsequently requested it will be path hit and serviced from the cache entry without any recourse to RMS.

Of course the same file may be requested with the unambigious path

```
/ht_root/home.html
```

which is completely different to the first instance, although ultimately accessing the same file. Hence one file may be cached multiple times against distinct paths. Although isolated instances of this are possible, the likelihood of significant resources being consumed in practice should be quite low.

Why Implement Caching?

Caching, in concept, attempts to improve performance by keeping data in storage that is faster to access than it's usual location. The performance improvement can be assessed in three basic ways; reduction of

- response when accessing the data (latency and transfer time)
- processing involved (CPU cycles)
- impact on the usual storage location (file system I/O)

This cache is provided to address all three. Where networks are particularly responsive a reduction in request latency can often be noticable. It is also suggested a cache ''hit'' may consume less CPU cycles than the equivalent access to the (notoriously expensive) VMS file system. Where servers are particularly busy or where disk subsystems particularly loaded a reduction in the need to access the file system can significantly improve performance while simultaneously reducing the impact of the server on other system activities. The author's feeling is though, that for most VMS sites high levels of hits are not a great concern, and for these caching can easily be left disabled, particularly if the file system's virtual I/O cache is enabled.

A comparison between cached and non-cached performance is provided in Chapter 17.

Why take so long to implement caching? (introduced in version 4.5) Well, WASD's intranet services are not particularly busy sites. This coupled with powerful hardware has meant performance has not been an overriding concern. However, this cache module came about because I felt like creating it and it was an obvious lack of functionality within the server, not because WASD (the organisation) needed it.

Terminology

This is what is meant when used.

- ''hit'' refers to a request path being found in cache. If the data is still valid the request can be supplied from cache.
- ''load''ing the cache refers to reading the contents of a file into cache memory.
- ''valid'' means that the file from which the cached data was originally read has not had it's revision date changed (the implication being is the file contents have not changed.

16.1 Cache Suitability Considerations

A cache is not always of benefit! It's cost may outweigh it's return.

Any cache's efficiencies can only occur where subsets of data are consistently being demanded. Although these subsets may change slowly over time a consistent and rapidly changing aggregate of requests lose the benefit of more readily accessable data to the overhead of cache management, due to the constant and continuous flushing and reloading of cache data. This server's cache is no different, it will only improve performance if the site experiences some consistency in the files requested. **For sites that have only a small percentage of files being repeatedly requested it is probably better that the cache be disabled.** The other major consideration is available system memory. On a system where memory demand is high there is little value in having cache memory sitting in page space, trading disk I/O and latency for paging I/O and latency. **On memory-challenged systems cache is probably best disabled.**

To help assessment of the cache's efficiency for any given site monitor the administration menu's cache report.

Two sets of data provide complementary information, cache activity and file request profile.

- **Activity information** summarizes the cache search behaviour, in particular that of the hash table.

  The ''searched'' item, indicates the number of times the cache has been searched. Most importantly, this may include paths that can never be cached because they represent non-file requests (e.g. directory listings). Requests involving scripts, and some others, never attempt a cache search.

  The ''hit'' item, indicates the number of times the hash table directly provided a cached path. This is very efficient.

  The ''miss'' item, indicates the number of times the hash table directly indicated a path was not cached. This is decisive and is also very efficient.

  The ''collision'' item, indicates the number of times multiple paths resolved to the same hash table entry. Collisions require further processing and are far less efficient. The sub-items, ''collision hits'' and ''collision misses'' indicate the number of times that further processing resulted in a found or not-found cache item.

  A large number of cache misses compared to searches may only indicate a large number of non-cacheable requests and so depending on that further datum is not of great concern. A large proportion of collisions (say greater than 12.5%) is however, indicating either the hash table size needs increasing (1024 should be considered a minimum) or the hashing algorithm in the software need reviewing :^)

- **Files information** summarizes the site's file request profile.

  With the ''loads not hit'' item, the count represents the cumulative number of files loaded but never subsequently hit. If this percentage is high it means most files loaded are never hit, indicating the site's request profile is possibly unsuitable for caching.

  The item ''hits'' respresents the cumulative, total number of hits against the cumulative, total number of loads. The percentage here can range from zero to many thousands of percent :^) with less than 100% indicating poor cache performance and from 200% upwards better and good performance. The items ''1-9'', ''10-99'' and ''100+'' show the count and percentage of total hits that occured when a given entry had experienced hits within that range (e.g. if an entry has had 8 previous hits, the ninth increments the ''1-9'' item whereas the tenth and eleventh increments the ''10-99'' item, etc.)

  Other considerations also apply when assessing the benefit of having a cache. For example, a high number and percentage of hits can be generated while the percentage of ''loads not hit'' could be in the also be very high. The explanation for this would be one or two frequently requested files being hit while most others are loaded, never hit, and flushed as other files request cache space. In situations such as this it is difficult to judge whether cache processing is improving performance or just adding overhead.

Recommendation

Monitor the site's cache behaviour and adjust parameters from an assessment based on the guidelines above.

Again, the author's suggestion is, that for most VMS sites, high levels of access are not a great concern, and for these caching can easily be left disabled.

16.2 Cache Content Validation

The cache will automatically revalidate the file data after a specified number of seconds ([CacheValidateSeconds] configuration parameter), by comparing the original file revision time to the current revision time. If different the file contents have changed and the cache contents declared invalid. If found invalid the file transfer then continues outside of the cache with the new contents being concurrently reloaded into the cache.

Cache validation is also always performed if the request uses ''Pragma: no-cache'' (i.e. as with the Netscape Navigator *reload* function). Hence there is no need for any explicit flushing of the cache under normal operation. If a document does not immediately reflect any changes made to it (i.e. validation time has not been reached) validation (and consequent reload) can be ''forced'' with a browser reload.

If a site's contents are relatively static the validation seconds could be set to an extended period (say 3600 seconds, one hour) and then rely on an explicit ''reload'' to force validation of a changed file.

The entire cache may be purged of cached data either from the server administration menu or using command line server control, as in the following example

```
$ HTTPD /DO=CACHE=PURGE
```

16.3 Cache Configuration

The cache is controlled using HTTPD$CONFIG file configuration directives. A number of parameters control the cache's behaviour. See the example configuration file HT_ROOT:[EXAMPLE]HTTPD$CONFIG.CONF.

- **Cache** enables and disables caching.
- **CacheChunkKBytes** sets the granularity of memory blocks allocated to file data (in kilobytes). To reduce the number of, and possible blocks sizes of memory, being requested (potentially improving memory allocation performance) data allocated to cache is done in multiples of this chunk size.
- **CacheEntriesMax** and **CacheTotalKBytesMax** provide growth limits to cache expansion. Maximum entries limits the number of files loaded into the cache before entries begin to be reused (flushing the original contents). Maximum total kilobytes allocated to the cache provides a ceiling on the memory consumed. These parameters operate to limit each other (i.e. if one reaches it's limit before the other, the other will not grow further either).
- **CacheFileKBytesMax** provides a limit on file size (in kilobytes). Files larger than the specified limit will never be cached.

- **CacheFrequentHits** and **CacheFrequentSeconds** attempt to reduce unproductive reuse of cache entries by providing the cache with some indication of what constitutes a frequently hit entry. If it is frequently hit then it should not be immediately reused when there is a demand for cache space. The first parameter sets the number of hits an entry must sustain before being a candidate for ‘‘CacheFrequentSeconds’’ assessment. If a file has been hit at least ‘‘CacheFrequentHits’’ times in total and the last hit was within the number of seconds set by ‘‘CacheFrequentSeconds’’ it will not be flushed and reused. If it has not been hit within the specified period it will be reused.
- **CacheHashTableEntries** sets the size of the hash table, used to rapidly index into the cached paths. Generally bigger is better (each entry consumes four bytes, hence 1024 entries consume 4096 bytes). Due to the hashing algorithm employed the size should be in magnitudes of two (e.g. 1024, 2048, etc.), if not, the specified amount is *rounded down* to the nearest (e.g. 1000 to 512).
- **CacheValidateSeconds** The interval after which a cache entry’s original, content revision time is revalidated against the file’s current revision time. If not the same the contents are declared invalid and reloaded. Setting this to a greater period reduces disk I/O but revised files may not be obvious within an acceptable timer unless a re-validation is forced with a *reload*.

16.4 Cache Control

The cache may be enabled, disabled and purged from the server administration menu (see Chapter 14). In addition the same control may be exercised from the command line (see Section 5.3.2) using

```
$ HTTPD /DO=CACHE=ON
$ HTTPD /DO=CACHE=OFF
$ HTTPD /DO=CACHE=PURGE
```

If cache parameters are altered in the configuration file the server must be restarted to put these into effect. Disabling the cache on an ad hoc basis (from menu or command line) does not alter the contents in any way so it can merely be reenabled with use of the cache’s previous contents resuming. In this way comparisions between the two environments may more easily be made.

Chapter 17

Server Performance

The server has a single-process, multi-threaded, asynchronous I/O design. On a single-processor system this is the most efficient approach. On a multi-processor system it is limited by the single process context (with scripts executing within their own context).

The server has been tested with up to 30 concurrent requests originating from 6 different systems and continues to provide an even distribution of data flow to each client (albeit more slowly :^)

The test system was a lightly-loaded AlphaServer 2100 4/275, VMS v7.1 and DEC TCP/IP 4.2. No *Keep-Alive:* functionality was employed so each request required a complete TCP/IP connection and disposal, although the WWWRKOUT utility (see Section 19.7) was used on the same system as the HTTP server, eliminating actual network transport. DNS (name resolution) and access logging were disabled.

As of v6.0 the performance data is collected and collated automatically using the WWWRKOUT metric functionality, which reads and interprets directive input via the /METRIC= qualifier, running batches of requests then averaging the results. The raw performance data is available in HT_ROOT:[EXERCISE], and the summaries compiled below. **These results are indicative only! On a clustered, multi-user system too many things vary slightly all the time. Hence the batching of accesses, interleaved between servers, attempting provide a representative result.**

OSU Comparison

Until v5.3 a direct comparison of performance between OSU and WASd had not been made (even to satisfy the author’s own occasional curiosity). After a number of users with experience in both environments commented . . . WASD *seemed* faster, was it? . . . it was decided to make and provide comparisons using the same metrics used on WASD for some time.

Every endeavour has been made to ensure the comparison is as equitable as possible (e.g. each server executes at the same process priority, has a suitable cache enabled, runs on the same machine in the same relatively quiescent environment, 2 CPUs to allow OSU to take advantage of VMS V7.1’s kernel threads - checked to be enabled via the THREADCP utility and ‘‘$ SHOW SYSTEM’’). Each test run was interleaved between each server to try and distribute any environment variations. Tests showing /PORT=7777 were to the OSU server, any others to WASD. Both servers were configured ‘‘out-of-the-box’’, minimal changes (generally just path mappings), WASD executing via the FREEWARE_DEMO.COM procedure.

Of course performance is just one of a number of considerations in any software environment (otherwise we wouldn’t be using VMS now would we? ;^) No specific conclusions are promoted by the author. Readers may draw their own from the results recorded below.

For this document the results were derived using the WASD v6.1 and OSU 3.6 servers.

17.1 Simple File Request Turn-Around

A series of tests using batches of 200 accesses were made and the results averaged. The first test returned an empty file measuring response and file access time, without any actual transfer. The second requested a file of 64K characters, testing performance with a more realistic load. All were done using one and ten concurrent requests.

Cache Disabled - Requests/Second

Response	Concurrent	WASD	OSU
0K	1	86	49
0K	10	115	62
64K	1	30	13
64K	10	41	17

Cache Enabled - Requests/Second

Response	Concurrent	WASD	OSU
0K	1	268	130
0K	10	324	169
64K	1	47	22
64K	10	51	25

Configuration and result files:

- HT_ROOT:[EXERCISE]PERF_FILES.CONF
- HT_ROOT:[EXERCISE]PERF_FILES_NOCACHE_V61.TXT
- HT_ROOT:[EXERCISE]PERF_FILES_CACHE_V61.TXT

Significantly, with both WASD cached and non-cached (as with OSU), **throughput actually improves** at ten concurrent requests (probably due to the latency of the serial TCP/IP connection/disconnection in one-by-one, compared to several happening concurrently).

Note that the response and transfer benefits decline noticably with file size (transfer time). The difference between cached and non-cached with the zero file size (no actual data transfer involved) gives some indication of the raw difference in response latency, some 200-300% improvement. This is a fairly crude analysis, but does give some indication of cache efficiencies.

Just one other indicative metric of the two servers, CPU time consumed during the file measurement runs.

CPU Time Consumed (Seconds)

Cache	WASD	OSU
Disabled	95	368
Enabled	42	180

File Transfer Rate

Under similar conditions results indicate a potential transfer rate **well in excess of 1 Mbyte per second**. (Remember, both client and server are on the same system, so the data, although being transported by TCP/IP networking, is not actually ending up out on a physical network.) This serves to demonstrate that server architecture should not be the limiting factor in file throughput.

Transfer Rate - MBytes/Second

Response	Concurrent	WASD	OSU
2.4MB (4700 blocks)	1	3.4	1.6
2.4MB (4700 blocks)	10	3.7	1.9

Configuration and result files are the same as for file request turn-around metrics above.

Significantly, there were no dramatic drops in transfer rate between one and ten concurrent requests! In fact an small increase in throughput!

File Record Format

The server can handle STREAM, STREAM_LF, STREAM_CR, FIXED and UNDEFINED record formats very much more efficiently than VARIABLE or VFC files.

With STREAM, FIXED and UNDEFINED files the assumption is that HTTP carriage-control is within the file itself (i.e. at least the newline (LF), all that is required required by browsers), and does not require additional processing. With VARIABLE record files the carriage-control is implied and therefore each record requires additional processing by the server to supply it. Even with variable record files having multiple records buffered by the HTTPd before writing them collectively to the network improving efficiency, stream and binary file reads are by

Virtual Block and are written to the network immediately making the transfer of these very efficient indeed!

So significant is this efficiency improvement a module exists to automatically convert VARIABLE record files to STREAM-LF when detected by the file transfer module. This is disabled by default but the user is strongly encouraged to enable it and to ensure that stream format files are provided to the server by other hypertext generating and processing utilitites.

17.2 Scripting

Persistant-subprocesses are probably the most efficient solution for child-process scripting under VMS. See ‘‘Scripting Environment’’ document. The I/O still needs to be on-served to the client by the server.

A simple performance evaluation shows the relative merits of the four WASD scripting environments available, plus a comparison with OSU. These were obtained using the WWWRKOUT utility (see Section 19.7) accessing the same CGI test utility script, HT_ROOT:[SRC.CGIPLUS]CGIPLUSTEST.C, which executes in both standard CGI and CGIplus environments, and an ISAPI example DLL, HT_ROOT:[SRC.CGIPLUS]ISAPIEXAMPLE.C, which provides equivalent output. A series of 200 accesses were made and the results averaged. The first test returned only the HTTP header, evaluating raw request turn-around time. The second test requested a body of 64K characters, again testing performance with a more realistic load.

DECnet-based scripting was tested using essentially the same environment as subprocess-based CGI, assessing the performance of the same script being executed using DECnet to manage the processes. Three separate environments have been evaluated, WASD-DECnet-CGI, WASD-OSU-emulation and OSU. The OSU script used the WASD CGISYM.C utility to generate the required CGI symbols (also see WASD/OSU Comparison). DECnet Phase-IV was in use.

CGI Scripting - Requests/Second

Response	Concurrent	CGI	CGIplus	ISAPI	DECnet-CGI	OSU-emul	OSU
0KB	1	16	74	71	10	8	7
0KB	10	26	96	98	16	12	10
64KB	1	11	24	23	8	7	5
64KB	10	17	30	32	11	9	6

Configuration and result files:

- HT_ROOT:[EXERCISE]PERF_SCRIPTS.CONF
- HT_ROOT:[EXERCISE]PERF_SCRIPTS_V61.TXT

Scripting Observations

Although these results are indicative only, they do show CGIplus and ISAPI to have a potential for improvement over standard CGI of up to 400%, a not inconsiderable improvement. Of course this test generates the output stream very simply and efficiently and so excludes any actual processing time that may be required by a ‘‘real’’ application. **If the script/application has a large activation time the reduction in response latency could be even more significant** (e.g. Perl scripts and RDMS access languages).

DECnet Observations

This section comments on non-persistant scripts (i.e. those that must run-up and run-down with each request - general CGI behaviour). Although not shown here measurements of connection reuse show significant benefits in reduced response times, consistency of response times and overall throughput, showing a difference of some 200% over non-reuse (similar improvements were reported with the OSU 3.3a server).

With ten simultaneous and back-to-back scripts and no connection reuse many more network processes are generated than just ten. This is due to the NETSERVER maintenance tasks such as log creation and purging, activating and deactivating the task, etc., adding latency into this script environment. The throughput was generally still lower than with subprocess-based scripting.

While earlier versions cautioned on the use of DECnet-based scripting this has been relaxed somewhat through connection reuse.

WASD/OSU Comparison

A direct comparison of CGI performance between WASD and OSU scripting is biased in favour of WASD, as OSU scripting is based on it’s own protocol with CGI behaviour layered-in above scripts that require it. Therefore a non-CGI comparison was devised. The script, HT_ROOT:[SCRIPT]FACE2FACE.COM, is designed to favour neither environment, merely return the plain-text string ‘‘Hello!’’ as quickly as possible.

```
$! OSU and WASD scripting face-to-face in a script that favours neither unduly
$ if f$type(WWWEXEC_RUNDOWN_STRING) .nes. ""
$ then
$    write net_link "<DNETTEXT>"
$    write net_link "200 Success"
$    write net_link "Hello!"
$    write net_link "</DNETTEXT>"
$ else
$    write sys$output "Hello!"
$ endif
```

Face-to-Face - Requests/Second

	Concurrent	CGI	CGIplus	ISAPI	DECnet-CGI	OSU-emul	OSU
‘‘Hello!’’	1	31	n/a	n/a	15	26	19
‘‘Hello!’’	10	51	n/a	n/a	23	34	20

Configuration and result files are the same as for scripting metrics above.

17.3 SSL

At this time there are no definitive measurements of SSL performance (see Chapter 13), as work on an SSL version of the WWWRKOUT utility has not yet been undertaken. One might expect that because of the CPU-intensive cryptography employed in SSL requests that performance, particularly where concurrent requests are in progress, would be significantly lower. In practice SSL seems to provide more-than-acceptable responsiveness.

17.4 Suggestions

Here are some suggestions for improving the performance of the server, listed in approximate order of significance. Note that these will have proportionally less impact on an otherwise heavily loaded system.

1. Disable host name resolution (configuration parameter [DNSLookup]). **DNS latency can slow request processing significantly!** Most log analysis tools can convert numeric addresses so DNS resolution is often an unnecessary burden.
2. Ensure served files are not VARIABLE record format (see above). Enable STREAM-LF conversion using a value such as 250 (configuration parameter [StreamLF], and SET against required paths using mapping rules).
3. Use persistant-subprocess DCL/scripting (configuration parameter [ZombieLifeTime])
4. Use CGIplus-capable or ISAPI scripts whenever possible.
5. Enable caching (configuration parameter [Cache]).
6. Disable logging (configuration parameter [Logging]).
7. Set the HTTP server process priority higher, say to 6 (use startup qualifier /PRIORITY=).
8. Reduce to as few as possible the number of mapping rules.
9. Use a pre-defined log format (e.g. ‘‘common’’, configuration parameter [LogFormat]).
10. Disable request history (configuration parameter [RequestHistory]).
11. Disable activity statistics (configuration parameter [ActivityDays]).

Chapter 18

HTTPd Web Update

The **Upd**ate facility allows Web documents and file environments to be administered from a standard browser. This capability is available to Web administrator and user alike. Availability and capability depends on the authorization environment within the server.

It **should be stressed** that this is not designed as a full hypertext administration or authoring tool, and for document preparation relies on the editing capabilities of the <TEXTAREA> widget of the user's browser. It does however, allow **ad-hoc changes** to be made to documents fairly easily, as well as allowing documents to be deleted, and directories to be created and deleted.

Consult the current **Upd**ate documentation for usage detail.

online hypertext link

Update Access Permission

If SSL is in use (see Chapter 13) then username/password privacy of the authorization environment is inherently secured via the encrypted communications. To restrict web update functionality to this secure environment add the following to the HTTPD$MAP configuration file:

```
  /upd/*  "403 Access denied."  ![sc:https]
```

Of course, the user must have write (POST/PUT) access to the document or area on the server (i.e. the *path*) and the server account have file system permission to write into the parent directory .

The server will report ‘‘Insufficient privilege or object protection violation ... /path/document’’ if it does not have file system permission to write into a directory.

Also see Section 11.10 for information on write access control for the server account.

Chapter 19

Utilities

Foreign commands for external utilities (and the HTTPD control functionality) will need to be assigned in the adminstration users' LOGIN.COM. For example:

```
$ HTTPD == "$HT_EXE:HTTPD"
$ HTTPDMON == "$HT_EXE:HTTPDMON"
$ SCRUNCH == "$HT_EXE:SCRUNCH"
$ UNSCRUNCH == "$HT_EXE:SCRUNCH/UNSCRUNCH"
$ STREAMLF == "$HT_EXE:STREAMLF"
$ WWWRKOUT == "$HT_EXE:WWWRKOUT"
```

19.1 Echo Facility

Ever had to go to extraordinary lengths to find out exactly what your browser is sending to the server? The server provides a request echo facility. This merely returns the complete request as a plain-text document. This can be used for for checking the request header lines being provided by the browser, and can be valuable in the diagnosis of POSTed forms, etc.

This facility must be enabled through a mapping rule entry.

```
script /echo/* /echo/*
```

It may then be used with any request merely by inserting ‘‘/echo’’ at the start of the path, as in the following example.

```
http://wasd.dsto.defence.gov.au/echo/ht_root/
```

19.2 Where Facility

Need to locate where VMS has the HTTPd files? This simple facility maps the supplied path then parses it to obtain a resulting VMS file specification. **This does not demonstrate whether the path actually exists!**

This facility must be enabled through a mapping rule entry.

```
script /where/* /where/*
```

It may then be used with any request merely by inserting ''/where'' at the start of the path, as in the following example.

```
http://wasd.dsto.defence.gov.au/where/ht_root/
```

19.3 Xray Facility

The Xray facility returns a request's complete response, **both header and body**, as a plain text document. Being able to *see* the internals of the response header as well as the contents of the body rendered in plain text can often be valuable when developing scripts, etc.

This facility must be enabled through a mapping rule entry.

```
script /Xray/* /Xray/*
```

It may then be used with any request merely by inserting ''/xray'' at the start of the path, as in the following example.

```
http://wasd.dsto.defence.gov.au/xray/ht_root/
```

19.4 Scrunch Utility

The ''scrunch''er is used to decrease the processing overhead of Server-Side Includes (SSI) files. See the ''Environment Overview'' for more detail on this utility.

A *scrunched* SSI file is a variable length record file, where each record either comprises a single SSI directive, with the ''<!–#'' beginning on the record boundary, or a record of other text to be output (i.e. not beginning with ''<!–#'').

Why do this? Well, if all SSI directives begin on a record boundary you only have to check the first five characters of each record to establish whether it should be interpreted or directly output! This **saves checking every character of every record** for the opening ''<'' and the following ''!–#''.

Files that have been scrunched are basically unsuitable for editing (only due to the often inappropriately sized records). Previously scrunched files may be returned to something (often exactly) resembling their original condition using the /UNSCRUNCH qualifier.

19.5 StreamLF Utility

This utility converts VARIABLE format files to STREAM_LF. The WASD HTTPd server access STREAM_LF files in block/IO-mode, far more efficiently that the record-mode required by variable-record format files. Use ''STREAMLF/HELP'' for some basic usage information.

NOTE: The server can also be configured to automatically convert any VARIABLE record format files it encounters to STREAM_LF.

19.6 HTTPd Monitor

The HTTP server may be monitored in real-time using the HTTPDMON utility. This utility continuously displays a screen of information comprising three or four of the following sections:

1. **Process Information**
   HTTPd process information includes its up-time, CPU-time consumed (excluding any subprocesses), I/O counts, and memory utilization.
2. **General Server Counters**
   The server counters keep track of the total connections received, accepted, rejected, etc., totals for each request type (file transfer, directory listing, image mapping, etc.).
3. **Proxy Serving Counters**
   The server counters keep track of proxy serving connections, network and cache traffic, cache status, etc.
4. **Latest Request**
   This section provides the response status code, and some transaction statistics, the service being accessed, originating host and HTTP request. Note that long request strings may be truncated (indicated by a bolded elipsis).

The following shows example output (after WWWRKOUT server testing):

```
Port: 8000  HTTPDMON v1.11.0 AXP (HTTPd v6.1)  Thursday, 18-NOV-1999 09:39:30

Process: HTTPd:8000  PID: 2020022F  User: HTTP$SERVER
     Up: 0 21:05:51.88  CPU: 0 07:34:19.99  Restart: 0
Pg.Flts: 3526  Pg.Used: 37%  WsSize: 6402  WsPeak: 4729
    AST:   39/200   BIO:   13/100   BYT:     576/202720  DIO:    4/100
    ENQ:    1/2000  FIL:   16/300   PRC:       5/100      TQ:   12/30

Connect: Accept:178212 Reject:0 Busy:0 Current:16 Peak:21  SSL: 0 (0%)
  Parse: 184865  Forbidden: 0  Redirect: Remote:0 Local:6654
CONNECT: 0  GET: 164717  HEAD: 20148  POST: 0  PUT: 0
  Admin: 0  Cache: Load:48 Hit:39992/0  DECnet: CGI:3327 OSU:6655
    Dir: 6670 DCL: CLI:16730 CGI:9987 CGIplus:13310/13182 Spawned:168 Current:5
   File: 3384/0  IsMap: 0  Proxy: 64911  Put: 0  SSI: 16657  Upd: 9988

    1xx: 0  2xx: 157816  3xx: 3  4xx: 18496  5xx: 1864  (0 errors)
     Rx: 19,762,345  Tx: 1,053,847,437  (bytes)

  Proxy: enabled
CONNECT: 0  GET: 58720  HEAD: 6191  POST: 0  PUT: 0
    Not: Cacheable Request:14555 Response:23786
Network: Rx:118,463,055 Tx:7,208,716 (29%)  Accessed: 38356
 Lookup: Numeric:19 DNS:21 Cache:38317 Error:1
  Cache: Rx:2,863,617 Tx:320,164,012 (71%)  Read:26554/3 Write:10
 Device: DKA0: 4110480 blocks (2007MB)  Space: available
         2171744 used (1060MB 53%), 1938736 free (946MB 47%)
  Purge: 18 00:00:54, 98 files (2254/2412), 0 deleted (0/0)

   Time: 18 15:56:11  Status: 200  Rx: 95  Tx: 34479  Duration: 0.2400
   Host: beta.wasd.dsto.defence.gov.au (131.185.250.201)
Request: GET /ht_root/exercise/64k.txt
```

The ‘‘/HELP’’ qualifier provides a brief usage summary.

This information is, in part, obtained from the following logical names:

- **HTTPD*port*$COUNT1**
- **HTTPD*port*$COUNT2**
- **HTTPD*port*$PID**
- **HTTPD*port*$REQUEST**

The server counter values are carried over when a server (re)starts (provided the system has stayed up). To reset the counters use the on-line server administration menu (see Chapter 14).

If [DNSlookup] is disabled for the HTTP server the HTTPDMON utility attempts to resolve the numeric address into a host name. This may be disabled using the /NORESOLVE qualifier.

19.7 Server Workout *(stress-test)*

The WWWRKOUT (‘‘World Wide Web Workout’’) utility exercises an HTTP server, both in the number of concurrent connections maintained and in the number of back-to-back sequential connection requests and data transfers.

This utility can be used to *stress-test* the WASD VMS HTTP server (or any other), or to make comparisons between it and other servers. When stress-testing a server, evaluating performance or just using it to try and break a test-bed server, it is best used from multiple, autonomous systems concurrent.

It sets up and maintains a specified number of concurrent connections to a server. It reads a buffer of data from each connection in turn, where data is waiting (does not block), until the document transfer is complete and the connection closed by the server. It then closes the local end and immediately reuses the now-free socket to initiate another sequence. If enabled (it is by default), the utility attempts to reflect the real-world in varying the data transfer rate for each connection, by setting the number of bytes read during each read loop differently for each connection. All transfered data is discarded.

The data transfer rate for each connection is displayed at connection close. It is by default it is the effective transfer rate, that is the rate from opening the connection to closing it, and so includes request processing time, etc. If the ''/NOEFFECTIVE'' qualifier is employed it measures the document data transfer rate only.

Although a single document path may be specified on the command line it is preferable to supply a range of document paths, one per line in a plain text file. Each document path and/or type specified should be different to the others, to exercise the server and file system cache. Any number of paths may be specified in the file. If the file is exhausted before the specified number of connections have been established the file contents are recycled from the first path. If a path or a file of paths is not specified the utility just continually requests the welcome document.

To assess a server's total throughput choose paths that lead to large documents (> 50K), where the overhead of connection setup, rule processing and transfer initiation are less significant than the data transfer itself. The buffer size variation functionality should be disabled using the ''/NOVARY'' qualifier when assessing data transfer rates. Responsiveness is better assessed using small documents (< 2K), where the overhead of the non-data-transfer activities is more significant.

WWWRKOUT [server_host_name[:port]] [path] [qualifiers]

- **/[NO]BREAK** will randomly break a small number of connections during the data transfer (tests server recovery under those circumstances)
- **/BUFFER=** number of bytes to be read from server each time (default is 1024, will be modified by the default ''/VARY'' qualifier)
- **/COUNT=** total number of connections and requests to be done (default is 100)
- **/[NO]EFFECTIVE** measures data transfer rate from request to close (if ''/NOEFFECTIVE'' is applied the rate is measured during data transfer only
- **/FILEOF=** file name containing paths to documents
- **/HELP** display brief usage information
- **/OUTPUT=** file name for output
- **/PATH=** single path to document to be retrieved
- **/PORT=** IP port number of HTTP server (default is 80)

- **/PROXY=** host name and port of proxy server
- **/SERVER=** HTTP server host name
- **/SIMULTANEOUS=** number of simultaneous connections to be set up at any one time (default is 10)
- **/[NO]VARY** varies the size of the read buffer for each connection (default is vary)

Examples:

```
$ WWWRKOUT
$ WWWRKOUT www.server.host "/web/home.html"
$ WWWRKOUT www.server.host:8080 /FILEOF=PATHS.TXT
$ WWWRKOUT /SERVER=www.server.host /PORT=8080 /NOBREAK /NOVARY /FILEOF=PATHS.TXT
$ WWWRKOUT www.server.host:8080 /FILEOF=PATHS.TXT /NOEFFECTIVE /NOVARY
$ WWWRKOUT www.server.host /FILEOF=PATHS.TXT /COUNT=500 /SIMUL=20 /BUFFER=512
```

The ‘‘/HELP’’ qualifier provides a brief usage summary.