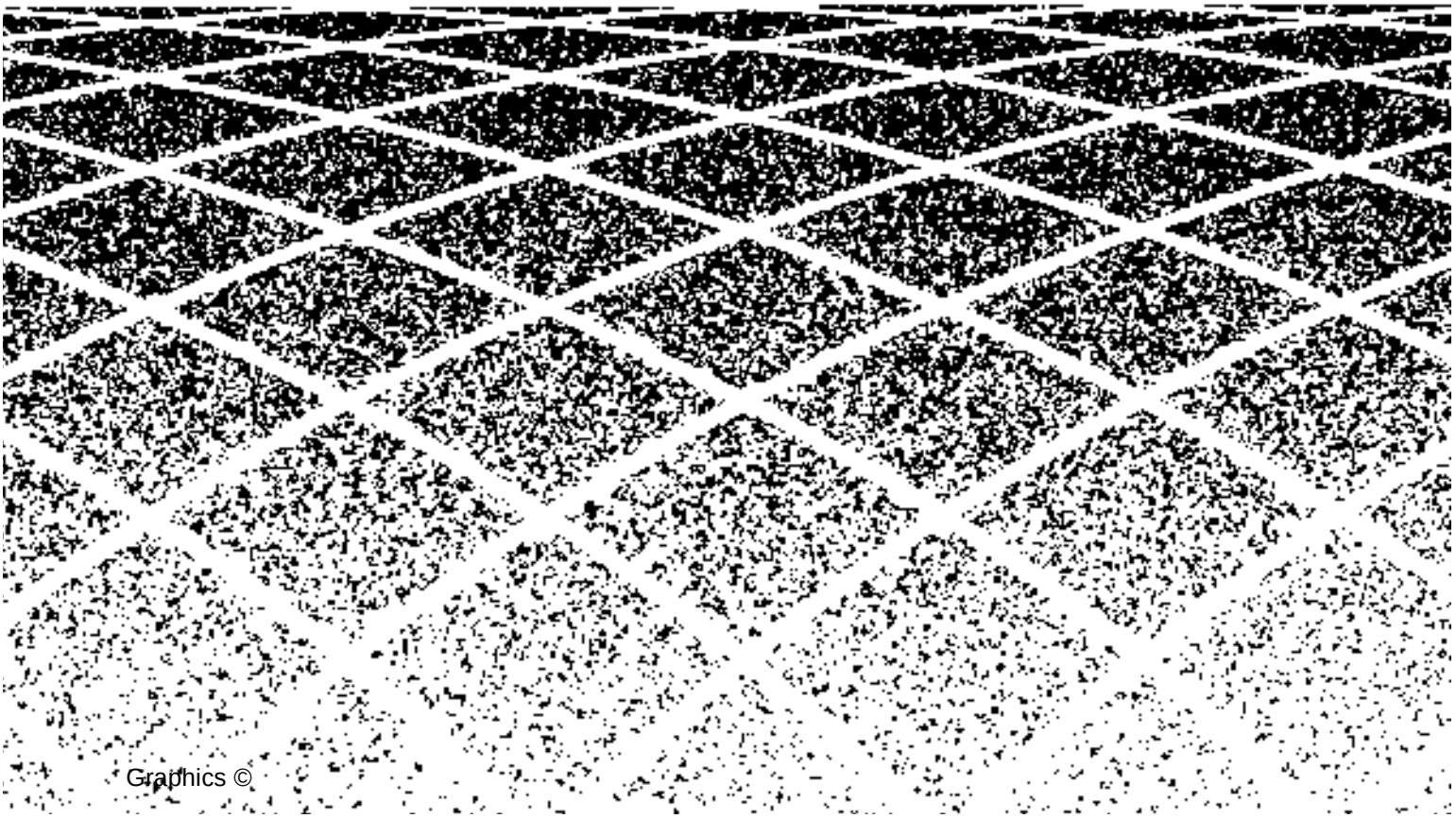
585-310-230
Issue 1
March, 1995

AT&T INTUITY™ CONVERSANT® System Command Reference



Blank Page

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

About This Book

Purpose

This book is a quick reference of all commands used with the AT&T INTUITY™ CONVERSANT® System.

How to Use This Book

This book provides an alphabetical listing of the CONVERSANT VIS commands. For each command listed, the following information is provided (where applicable):

- Name — This section provides the name and purpose of the command.
- Synopsis — This section summarizes the usage of the program being described.
- Description — This section discusses how to use the command.
- Example(s) — This section provides examples of the usage of the command, where appropriate.
- Notes — This section provides information that may be helpful under the particular circumstances described.
- Warning — This section discusses the limits or boundaries of the respective command, where appropriate.
- Caveats — This section points out possible pitfalls to consider when using the command.
- Return Values — This section describes the values returned if the command is successful or if the command failed.
- Files —This section gives the file names that are built into the program or the command.
- See Also — This section offers pointers to related information or related commands.

This book also provides a list of "Abbreviations" used in Intuity CONVERSANT VIS documentation, as well as a cross-referenced "Index" listing the principal subjects contained in this book.

Conventions Used in This Book

The following typographic conventions are used in this book:

- Terminal keys
  - — Terminal keys are shown in rounded boxes. For example, an instruction to press the enter key is shown as

    Press ENTER.

  - — Function keys (also known as *soft* keys) are shown in rounded boxes followed by the function of that key in parentheses. For example, an instruction to press function key 3 is shown as

    Press F3 (CHOICES).

  - — Two or three keys that you press at the same time (that is, you hold down the first key while pressing the second and/or third key) are shown as a series of rounded boxes. For example, an instruction to press and hold ALT while typing the letter **d** is shown as

    Press ALT D.

- User input
  - — The word *enter* means to type a value and press ENTER. For example, an instruction to type **y** and press ENTER is shown as

    Enter **y** to continue.

  - — The word *type* means to press the key or sequence of keys specified. For example, an instruction to type **y** is shown as

    Type **y** to continue.

    Do *not* press ENTER after you type the value specified.

  - — The word *select* is used to mean the following: move to the desired menu item using the arrow keys and press ENTER. For example, an instruction to select an item from a menu and press ENTER is shown as

    Select Configuration Management from the Voice System Administration menu.

  - — Information that you enter or type from your terminal keyboard is shown in **bold** type; for example

    Enter **root** at the `Console Login` prompt.

- Command and file names and their parameters are shown in **bold** type. Variable parameters are shown in ***bold italic*** type when they are part of a user input and in *regular italic* type when they are not. All are illustrated in the following example:

  Use the **print** command to print your report. The command syntax is **print *reportname***, where *reportname* is the name of the report to be printed.

- Square brackets [ ] around an argument indicate that the argument is optional.
- List of options for a single argument are divided by vertical bars (a|b|c).
- Ellipses ... are used to show that the previous argument may be repeated.

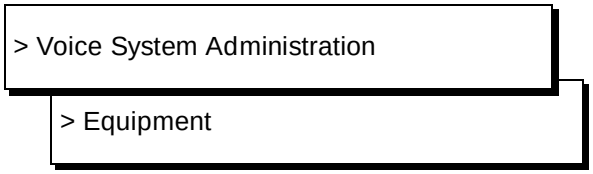
■ Screen displays

- Information that is displayed on your terminal screen — including screen displays, prompts, script code, and system messages — is shown in `typewriter-style` type; for example

  ```
  Installation is in progress -- do not remove
  the floppy disk.
  ```

- The sequence of menu options that you must select to display a specific screen is shown as follows:

  Begin at the CONVERSANT Administration menu, and select the following sequence:

  In this example, you would first access the CONVERSANT Administration menu. Then you would select the Voice System Administration option to display the Voice System Administration menu. From that menu, you would select the Voice Equipment option to display the Voice Equipment screen.

- The screens shown in the Intuity CONVERSANT library are only examples. Your screens may not appear exactly as illustrated.

Related Resources

The following books contain information related to the commands that appear in this book:

- *Intuity CONVERSANT VIS Version 5.0 Maintenance*, 585-310-153
- *Intuity CONVERSANT VIS Version 5.0 Application Development*, 585-310-227
- *Intuity CONVERSANT VIS Version 5.0 Operations*, 585-310-550

This book may be used in conjunction with all other standard and feature package in the CONVERSANT VIS library. Refer to *CONVERSANT VIS Documentation Guide*, 585-350-002, for a complete list of VIS documentation.

Trademarks and Service Marks

The following trademarked products are mentioned in this book:

- CONVERSANT is a registered trademark of AT&T.
- AUDIX and Voice Power are trademarks of AT&T.
- CLEO and DataTalker are trademarks of CLEO Communications.
- IBM is a registered trademark of International Business Machines.
- UNIX is a registered trademark of UNIX System Laboratories, Inc.
- ORACLE, SQL*Net, and SQL*Plus are registered trademarks of the Oracle Corporation.

How to Make Comments About This Book

A reader comment card is behind the title page of this book. While we have tried to make this book fit your needs, we are interested in your suggestions for improving it and urge you to complete and return a reader comment card.

If the reader comment card has been removed from this book, please send your comments to:

AT&T
Product Documentation Development
Room 22-2C11
11900 North Pecos Street
Denver, Colorado 80234

Please include the name and order number of this document.

Summary of Commands

1

What's in This Chapter

This chapter includes a summary of each command in the Intuity CONVERSANT Voice Information System (VIS) Version 5.0 documentation. Table 1-1 provides an alphabetical list and brief description of all commands in this book. You can use the table to locate the command needed to perform a particular task, and then find the annotated command page provided later in the chapter.

Table 1-1. Command Synopsis

Command	Function
"3270dip_off"	Turns off the 3270 DIP
"3270dip_on"	Turns on the 3270 DIP
"add"	Adds a phrase to a UNIX talkfile
"add_device"	Adds a new device to the **/vs/data/device_data** file
"addhdr"	Adds a voice or code header to a speech file
"addmsg"	Allows for the addition of explanation texts for error messages prior to Version 3.1
"alarm disable"	Disables the specified Alarm Contact Set
"Alarm Display"	Displays all Message IDs associated with a specified Alarm Contact Set
"Alarm Enable"	Enables the specified Alarm Contact Set for use
"Alarm Help"	Provides the user a means of assigning or removing Message IDs to each of 3 Alarm Contact Sets
"Alarm Reinit"	Causes the alarm process to reinitialize all internal data structures referring to alarms
"Alarm Retire"	Retires the specified Alarm Contact Set.25
"Alarm Status"	Displays the state and status of the specified Alarm Contact Set
"Alarm Test"	Tests the specified Alarm Contact Set for use
"annotate"	Annotates a TSM trace stream with a message
"assign card/channel"	Assigns a group number to a card or channel
"assign_permissions"	Assigns VIS security permissions to the user
"assign service/startup"	Assigns an installed service to DNIS and ANI numbers or directly to a channel
"assign_tty"	Permits the user to alter the ports assigned to be monitored by the SCCS and ARU
"attach"	Attaches a unit
"autoreboot"	Changes or displays the parameters associated with the autoreboot feature
"backup_appl"	Backs up an application if enhanced file transfer is installed
"bbs"	Reports the status of the voice system Bulletin Board
"bk_appl"l	Backs up the speech and/or transaction component of a Script Builder application

Continued on next page

"ccarpt"	Generates a call classification data summary report

Table 1-1. Command Synopsis — *Continued*

Command	Function
"cddrpt"	Generates a call data detail report
"cdsrpt"	Generates a call data summary report for a specific date
"change_device"	Changes the name of a device in the **/vs/data/device_data** file
"checktf"	Checks for the existence of talkfiles in the voice system
"chg_machname"	Changes the name of the machine the SCCS is monitoring
"codetype"	Identifies the type of coding header in a speech file
"configure"	Determines the allocation of resources for all devices to be included in the system configuration for a given hardware platform
"console_off"	Turn the flow of error messages to the console off during system operation
"console_on"	Turn the flow of error messages to the console on during system operation
"copy"	Copies a phrase from one UNIX file to another UNIX file
"cpuType"	Returns the type of CPU used in the system
"cvis_mainmenu"	Accesses the Intuity CONVERSANT administrative menu
"cvis_menu"	Accesses the Intuity CONVERSANT Voice System Administration
"dbcheck"	Checks the resources available in the database
"dbfrag"	Lists fragmentation information on the database
"dbfree"	Checks the space available in the database by partition
"dbused"	Provides database use by oracle user
"decode"	Converts adpcm16 or adpcm32 files to pcm64 files
"defservice"	Defines an IRAPI service
"delete card/channel"	Removes a card or channel from a service or an equipment group
"delete eqpgrp"	Removes an equipment group
"delete service/startup"	Removes the assignment of a service to DNIS and ANI numbers or of a service assigned directly to a channel
"detach"	Places a unit in the nonexistent state
"diagnose bus"	Tests a bus while it is in service
"diagnose card"	Tests a card while it is in service

Continued on next page

"dip_int"	Sends a DIP interrupt to a script on a channel or a range or channels

Table 1-1. Command Synopsis — *Continued*

Command	Function
"display assignments"	Displays the services assigned to channels
"display card"	Displays information about specified cards
"display channel"	Displays channel information
"display dnis"	Displays the services assigned to DNIS and ANI numbers
"display eqpgrp/group"	Displays an equipment group report
"display messages"	Displays system messages
"display_permissions"	Displays VIS security permission information for the user
"display services"	Lists all valid services to scripts
"displaypkg"	Lists the software packages installed on the VIS
"edExplain"	Edits the explanation text for one or more message tags
"encode"	Converts ADPCM16 or ADPCM32 files to PCM64 files
"erase"	Deletes a phrase from a UNIX talkfile
"etStub"	Reads the IPC message queue for error messages that use the ET process
"explain"	Displays on-line error message explanations
"findHomes"	Populates a users home directory with files saved as part of assisted upgrade
"fixLogFile"	Upgrades existing logging files
"get_config"	Retrieves the **/vs/data/conf_data** file from a floppy disk
"gse"	Invokes the Graphical Speech Editor
"gse_add"	Transfers a speech phrase from a UNIX file to the UNIX file in the Graphical Speech Editor (GSE) format
"gse_addpl"	Adds (restores) phrases to a specific speech pool from UNIX files in the GSE format
"gse_copy"	Extracts a speech phrase from the VIS speech file system to a UNIX file in the GSE format
"gse_copypl"	Copies multiple speech phrases from the speech file system in the GSE format
"hassign"	Assigns host services to host sessions
"hconfig"	Configures the host interface parameters
"hdelete"	Removes host services from host sessions
"hdiagnose"	Diagnoses the SDLC communication card

Continued on next page

"hdisplay"	Shows host applications that have been successfully verified and installed

Table 1-1. Command Synopsis — *Continued*

Command	Function
"headFIX"	Converts ET message rules header files to the logger/alerter environment
"hfree"	Releases host sessions from Script Builder host application assignments
"hlogin"	Runs the login sequence of a host script
"hlogout"	Runs the log out sequence of the host script
"hnewscript"	Installs a changed host script
"host_cfg"	Translates an ASCII configuration file into a properly formatted binary configuration file
"hsend"	Sends a file to the host via Intuity CONVERSANT file transfer
"hspy"	Displays a screen currently present on the specified host session
"hstatus"	Shows the current status of the host sessions
"iCk, iCkAdmin"	Performs various integrity checks based on the rules in a script file
"install_appl"l	Installs an application if enhanced file transfer is installed
"install_sw"	Installs a software package if enhanced file transfer is installed
"installpkg"	Installs a software package
"into_et"	Sends error messages to the error tracker (ET)
"lComp"	Combines message files to produce compressed and expanded format files
"list"	Lists the directory entries for specific phrases
"load_aru"	Downloads the parameter settings of the ARU
"load_bin"	Initializes the 3270 host card
"logCat"	Reads compressed logging files and outputs human readable messages
"logDstPri"	Creates the shared memory containing the dynamic destinations and priorities of logging messages using logMsg
"logEvent/logMsg"	Allows shell scripts to log a specific message
"logFmt"	Displays and changes the parameters used to display messages and explanation texts
"logit"	Logs arbitrary messages to the logging files

Continued on next page

"logTest"	Generates an arbitrary list of logging messages to be sent to logdaemon

Table 1-1. Command Synopsis — *Continued*

Command	Function
"mkAlerter"	Reads an alerter description and generates the code that implements the description
"mkerr"	Converts ET message rules to the logger/alerter environment
"mkETrules"	Produces a valid etStub.rules file from one or more errors files used to control the ET process
"mkheader"	Allocates user memory for script variables
"mkimage"	Performs a complete system backup
"mkMsg"	Converts ET message rules to files for use in the logger/alerter environment
"msgadm"	Facilitates the administration of system messages
"newscript"	Updates the changes to all currently assigned scripts
"reinitLog"	Used when adding custom error messages
"remove"	Places a unit in the manual-out-of-service (MANOOS) state
"remove_appl"	Removes an application if enhanced file transfer is installed
"remove_device"	Removes a device from the **/vs/data/device_data** file
"remove_sw"	Removes an installed package if enhanced file transfer is installed
"removepkg"	Removes a software package
"restore"	Restores a unit to the in-service (INSERV) state
"restore_appl"	Restores an application if enhanced file transfer is installed
"retire"	Retires critical and major alarm indications on the ARU
"rmdb"	Displays the state of the resource manager and modify debug levels
"rs_appl"	Restores the speech and/or transaction component of a Script Builder application
"save_config"	Saves the **/vs/data/conf_data** to floppy disk
"sb_backup"	Backs up a Script Builder application
"sb_restore"	Restores a Script Builder application
"sb_te"	Invokes the 3270 terminal emulator
"sb_trace"	Displays trace messages and screens being sent between Script Buillder applications and the host mainframe for the specified host channel

Continued on next page

"sccsDaemon"	Distributes messages to the SCCS or CompuLert systems and to the alarm relay unit (ARU); can be used to send commands from the user to the daemon process

Table 1-1. Command Synopsis — *Continued*

Command	Function
"show_config"	Displays and prints to a file the valid or incomplete VIS configuration
"show_devices"	Displays and prints to a file all devices and their attributes as represented in the **/vs/data/device_data**
"show_sys"	Allows you to retrieve configuration and administration information from customer sites
"soft_disc"	Sends a disconnect to a script on a channel or channels.
"soft_szr"	Starts a script on a channel
"spCtlFlags"	Sets and clears flags used to control the behavior on SP executive pack files as they run on an SP card
"spres"	Restores speech from a backup
"spsav"	Backs up speech
"spStatus"	Displays information about the pack file running on an SP card
"spVrsion"	Prints the version of the SP driver currently installed on a machine
"start_hi"	Starts the 3270 host interface software
"start_vs"	Brings the voice system up to a fully operational state
"stop_hi"	Stops the 3270 host interface software
"stop_vs"	Stops the voice system software gracefully
"sysmon"	Executes a program that monitors incoming telephone lines and the associated cards to see that they are functional
"tas"	Executes the transaction assembler program to assemble script instructions
"trace"	Outputs trace messages for the specified processes and channels
"trarpt"	Generates the call traffic report file systems
"unassign_permissions"	Removes VIS security permissions for the user
"upg"	Provides automated assistance in upgrading a CONVERSANT VIS machine to Version 5.0
"vfyLogMsg"	Verifies the information associated with a specific logging message format
"vsdisable"	Disables the automatic restarting of the voice system

Continued on next page

"vsenable"	Enables the automatic starting of the voice system at system reboot

Table 1-1. Command Synopsis — *Continued*

Command	Function
"vusage"	Displays the current load on the voice system
"wdog_off"	Disables the watchdog timer function of the ARU
"wdog_on"	Enables the watchdog timer function of the ARU
"wl_copy"	Copies FlexWord vocabularies to disk
"wl_edit"	Edits FlexWord wordlists
"wl_gen"	Creates data files for a FlexWord vocabulary
"wl_init"	Generates an initial wordlist from a set of words
"wl_install"	Reads FlexWord vocabularies from floppy disk
"xferdip_off"	Deactivates the bridging capability
"xferdip_on"	Activates the bridging capability

3270dip_off

The **3270dip_off** command turns off the 3270 data interface process (DIP).

Synopsis

3270dip_off

Description

The **3270dip_off** command deactivates the 3270 DIP the next time the voice system is started.

NOTE:
You must stop and start the voice system for this command to take effect.

Files

/vs/data/HOST3270
/etc/inittab

Example

The following example turns off the 3270 DIP:

```
3270dip_off
```

See Also

"3270dip_on"
"start_vs"
"stop_vs"

3270dip_on

The **3270dip_on** command turns on the 3270 data interface process (DIP).

Synopsis

3270dip_on

Description

The **3270dip_on** command activates the 3270 DIP the next time the voice system is started.

NOTE:
You must stop and start the voice system for this command to take effect.

Example

The following example turns on the 3270 DIP.

3270dip_on

See Also

"3270dip_off"

add

The **add** command adds a phrase to a Unix talkfile.

Synopsis

add phrase *<phrase number>* to talkfile *<talkfile number>* from *<file_name>*

Description

The **add** command adds phrases to the specified talkfile that were previously extracted from another talkfile using the **"copy"** command. The path name for the file may be the full pathname or the relative pathname. If no path is specified, the file is created in the current working directory. If you are not in the directory from which the phrase to be added is stored, give the full path name for the talkfile and the source file. If the phrases already exists, the system displays the following message:

Phrase <phrase_number> already exists in talkfile
<talk file number>
Do you want to overwrite existing phrase? (y/n)

If an error occurs, system messages are printed on the controller screen. The source file may be a full path name or a relative path name. Refer to Chapter 3, "System Message Listings," of *Intuity CONVERSANT VIS Version 5.0 Maintenance*, 585-310-153, for how to respond to a system message.

NOTE:
The add command adds a phrase to the SPEECHDIR default directory, which is **/home2/vfs/talkfiles**. In order to add a phrase, the conventional naming scheme must be followed.

Files

/speech/talk/*.pl

Example

The following example adds phrase number 275 to talkfile 25 from the phrase stored in the UNIX file phr275 in the directory **/tmp/junk**.

add phrase 275 to talkfile 25 from /tmp/junk/phr275

The following example adds phrase 104 to talkfile 18 from the phrase stored in the UNIX file phr104 in the directory **/speech/talk**.

add phrase 104 to talkfile 18 from /speech/talk/phr104

See Also

"copy"
"erase"
"list"

add_device

The **add_device** command adds a new device to the **/vs/data/device_data** file.

Synopsis

/vs/bin/util/add_device

Description

The **add_device** command is an interactive command which allows the user to add a new device to the **/vs/data/device_data** file. Once a new device is added to this file, it appears in the device menus of the **/vs/bin/util/configure** program and may be selected for possible configuration in a VIS machine.

The **add_device** program prompts for all attributes that must be specified to make a new device entry. If there are no special attributes or rules associated with the new device, executing **add_device** is all that is necessary to add complete support of the device to the **configure** program.

The **add_device** program prompts for the following attributes:

NOTE:
A "**-**" (minus) in front of the attribute's description indicates a required attribute, whereas an "*****" (asterisk) indicates an attribute to be entered only if applicable. If an attribute is not applicable, entering **q** (for quit) when the prompt for that attribute is displayed skips that attribute.

- - Device Description: An 80-character limit. This attribute should list generic name, comcodes, list numbers, part number, etc.
- - User Mnemonic Name: A 20-character limit. This attribute should be the name by which all presentations of this device appear in the **"configure"** program's menus. This name must be unique from all other user mnemonic names.
- - Program Mnemonic Name: An 8-character limit. This attribute should be the name by which the **"configure"** program will internally refer to this device. This name must be unique from all other program mnemonic names.
- - Device Type: An 8-character limit. This can usually be the same as the program mnemonic name. However, in some cases, cards with different program mnemonic names are serviced by the same software driver. In this case they may share a hardware resource such as an interrupt.

  An example is the Tip/Ring (T/R) interface cards IVP6, IVP4, and VRS6. All three can share the same interrupt and are serviced by the same driver. By specifying type as TR for all three of these cards, the "configure" program is informed of this fact.

- * Interrupts: Each interrupt that is usable by the device should be entered one at a time, as prompted. The interrupts should be entered in priority order, that is, the preferred interrupt for this device should be entered first, second choice next, etc. After all interrupts are entered, the user is prompted to indicate whether all devices of this device's type (as described above) can share the same interrupt. Response is **y** or **n**.
- * Base IO address: Each base input/output (IO) address that is usable by the device should be entered one at a time as prompted for. The addresses should be entered in priority order, that is, the preferred address for this device should be entered first, second choice next, etc. Input must be specified in hex.

  After all addresses are entered, the user is prompted to enter the number of contiguous bytes of IO space that is required from the base address for this device. The number is expected in decimal. Note that the device can have only one range of IO addresses starting at the base. If the device does not meet this criteria, the "configure" program is not able to accurately determine configurations which include this device.

  After the above information is entered, the user is prompted to indicate whether all devices of this device's type (as described above) can share the same IO address. Response is y or n.
- * Base RAM address: Each base random access memory (RAM) address that is usable by the device should be entered one at a time, as prompted. The addresses should be entered in priority order, that is, the preferred address for this device should be entered first, second choice next, etc. Input must be in hex.

  After all addresses are entered, the user is prompted to enter the number of contiguous kbytes of RAM space that is required from the base address for this device. The number is expected in decimal and is to be expressed in kbytes. Note that the device can have only one range of RAM addresses starting at the base. If the device does not meet this criteria, the "configure" program will not be able to accurately determine configurations which include this device.

  After the above information is entered, the user is prompted to indicate whether all devices of this device's type (as described above) can share the same RAM address. Response is y or n.

  The user is also prompted to enter the data transfer bit rate supported by the device. This is usually, but not necessarily (check device documentation to be sure), the same as the slot type (8 or 16 bit).

  This attribute is required because devices which can support only 8-bit data transfers may not reside in the same 128-kbyte block of RAM as devices which support 16 bit data transfers. If the new device supports transfers data transfers 8 bits at a time, enter 8. If it

supports transfers 16 bits at a time, enter 16. If the "configure" program is not to check this attribute, enter x for do not care (this should be avoided).

- * DMA Channel: Each direct memory address (DMA) channel that is usable by the device should be entered one at a time as prompted for. The DMA channels should be entered in priority order, that is, the preferred channel for this device should be entered first, second choice next, etc.
- * Slot Type: The slot type required for the new device must be entered. Supported slots are 8, 16, or 32 bit slots. Enter **q** if the device does not require a slot. Confirmation is required if **q** is entered.
- * Miscellaneous Attributes: Answer **y** or **n** to the prompts for miscellaneous attributes:
    1. Intuity CONVERSANT VIS System Board: Enter **y** if the card is a CONVERSANT network interface card or speech processing card. Enter **n** otherwise.
    2. Serial Port Required Flag: Enter **y** if the device requires a serial port. Enter **n** otherwise.
    3. Parallel Port Required Flag: Enter **y** if the device requires a parallel port. Enter **n** otherwise.

Upon entering all requested information, the user has the option of confirming it or changing some part of it. If confirmed, the information is entered into the **/vs/data/device_data** file. The new device appears in the **"configure"** program's menu the next time it is invoked.

Files

/vs/data/device_data

See Also

"configure"
"change_device"
"remove_device"
"show_devices"
"show_config"
"save_config"
"get_config"

NOTE:
In order for the **"configure"** program to determine accurate configurations, all pertinent information should be accurately entered into the **device_data** file concerning a new device. When adding a new device, consult the Intuity CONVERSANT VIS Version 5.0 Hardware Installation documentation for your platform,

concerning the device and be certain that the information prompted for by the **add_device** program is entered completely and accurately.

addhdr

The **addhdr** command adds a voice or code header to a speech file.

Synopsis

addhdr ***[voice/pcm64/adpcm32/adpcm16/sbc24/sbc16/celp16] [tag]***

Description

The **addhdr** command is a filter that adds a header to a speech file. Two mututally exclusive types of headers are supported: voice and code. A voice header identifies a file as being editable by GSE, and includes an optional identifying tag. A code header (which can be PCM64, ADPCM32, ADPCM16, SBC24, SBC16, or CELP16) identifies the way in which the file is encoded. Code headers are required on any file that is to be played on the VIS.

Before converting between voice and code headers, you must strip off any existing headers.

NOTE:
Customers using **"gse_add"**, **"gse_addpl"**, **"gse_copy"**, and **"gse_copypl"** do not need to use this command directly.

See Also

"codetype"
"striphdr"

addmsg

The **addmsg** command is the compatibility program which allows for the addition of explanation texts for error messages prior to Version 3.1.

Synopsis

addmsg

Description

The **addmsg** command is the program that added and/or modified explanations for error numbers in VIS systems prior to Version 3.1 (V3.1). A compatibility version is provided for VIS 3.1. Use **"edExplain"** in Version 3.1.

The **addmsg** command takes one or more repetitions of the form:

{error-number}
one or more lines of explanation text

The *{error-number}* must be strictly digits. This identifies error numbers for pre-Version 3.1. The explanation follows immediately on one or more lines and is terminated by a line containing only a period.

Input is terminated by entering CTRL D when prompted for the next error number.

As each explanation is entered, it is placed in an individual file in the appropriate **/gendb/data/explain/*{dir}*** directory, where *{dir}* is the first digit of the error number.

Files

/gendb/data/explain/[0-9]/{err-num} # Explanation file
/tmp/add[mM]sg.{PID} # Two temporary files

Diagnostics

The **addmsg** command returns the number of faulty explanations entered. Any attempt to enter an explanation with no lines in it is considered to be an error. The value 0 is returned when **addmsg** is completely successful.

Examples

The following example is input for message text explanations from a tty:

```
# addmsg
What's the error code number? (Press '<Ctrl>-d' to
quit) 5620
Type in message. (End by a . on line by itself)
5620 indicates that a bad record has been received.
.
What's the error code number? (Press '<Ctrl>-d' to
quit)  <CTRL-D>
```

The following example displays input information from a file. If input comes from a file, a "here" file, or via a pipe from another process, no prompts are issued:

```
# addmsg <<!
5620
5620 indicates that a bad record has been received.
.
5621
5621 indicates that there is no more room for records.
.
!
```

See Also

"edExplain"

alarm disable

The **alarm disable** command disables the specified alarm and makes it unavailable for use.

Synopsis

alarm disable ***[all/1/2/3...]***

Description

The **alarm disable** command disables the specified Alarm Contact Set. This command does not affect the state of the contacts themselves. If an alarm occurs that is assigned to a disabled Alarm Contact Set, then the contacts will not close. Note that the **"Alarm Retire"** and test commands cause the contacts to close and open even though the set is disabled.

The numeric arguments refer to the alarm contact set.

Examples

The following is an example of the output for the **alarm disable 2** command:

Alarm Contact Set 2 now disabled

The following is an example output for the alarm disable all command:

Alarm Contact Set 1 already disabled
Alarm Contact Set 2 now disabled
Alarm Contact Set 3 now disabled

alarm display

The **alarm display** command displays all message IDs assigned to Alarm Contact Sets.

Synopsis

alarm display ***[all/1/2/3...]***

Description

The **alarm display** command displays all Message IDs associated with a specified Alarm Contact Set. The numeric arguments refer to the alarm contact set. This command is used in conjunction with the *[all]* or *[n]* options (where n is the alarm contact set number). A warning message is output if the specified Alarm Contact Sets are already disabled.

Examples

The following is a sample output for the alarm display 1 command:

No Message IDs currently assigned to Alarm Contact Set 1.

The following is a sample output for the alarm display 2 command:

Alarm Contact Set 2

```
TWIP001 TWIP002 TWIP003
```

The following is a sample output for the alarm display all command:

Alarm Contact Set 1

```
VROP001 VROP002 VROP003
```

Alarm Contact Set 2

```
TWIP001 TWIP002 TWIP003
```

Alarm Contact Set 3

```
TSM001  TSM002  TSM003
```

alarm enable

The **alarm enable** command enables the specified alarms to be available for use.

Synopsis

alarm enable ***[all|1|2|3...]***

Description

The **alarm enable** command enables the specified Alarm Contact Set for use. This command does not affect the state of the contacts themselves. If an alarm occurs that is assigned to an enabled Alarm Contact Set, then the contacts will close if they are not already closed. The numeric arguments refer to the alarm contact sets on the alarm relay card. This command is used in conjunction with the *[all]* or *[n]* options (where *n* is the alarm contact set number). A warning message is output if the specified Alarm Contact Sets are already enabled.

Examples

The following is a sample output for the **alarm enable 2** command:

Alarm Contact Set 2 now enabled

The following is a sample output for the **alarm enable all** command:

Alarm Contact Set 1 already enabled

Alarm Contact Set 2 now enabled

Alarm Contact Set 3 now enabled

alarm help

The **alarm help** command provides output information on each alarm command.

Synopsis

alarm help

Description

The purpose of External Alarm Administration is to provide the user a means of assigning or removing Message IDs to each of 3 Alarm Contact Sets. It also provides the user with the capability of enabling or disabling specific Alarm Contact Sets. The user can also test the functionality of each Alarm Contact Set without initiating a system alarm by using the test command to close a specific Alarm Contact Set. The "Alarm Retire" command will reopen the closed set.

alarm reinit

The **alarm reinit** command forces alarm processes to reinitialize internal data structures.

Synopsis

alarm reinit

Description

The **alarm reinit** command causes the alarm process to reinitialize all internal data structures referring to alarms. When alarm reinit is executed, all alarm contact sets are reset (alarm contacts are open), all alarm contact sets are enabled, the **/vs/data/alarms/alarmX files** are reread, and the **/vs/data/alarms/mask*file*** is reread. If the file **/vs/data/alarms/timer** exists, it will also be reread. In essence, execution of alarm reinit results in placing the system in a state identical to the state expected after system startup.

This command is useful for making changes take effect after the configuration file is modified. Any errors encountered in the configuration files are logged to the logger. Refer to the information on the alarm display command for additional information.

Examples

There is no sample output for the **alarm reinit** command. You may check the System Message Display screen for the results of the alarm reinit command.

alarm retire

The **alarm retire** command shuts off an alarm.

Synopsis

alarm retire *[all|1|2|3...]*

Description

The **alarm retire** command retires the specified Alarm Contact Set. The command removes external alarm by opening contacts on the specified Alarm Contact Set whether set is enabled or not. This command is used in conjunction with the *[all]* or *[n]* options (where *n* is the alarm contact set number).

Examples

The following is a sample output for the **alarm retire 2** command:

Alarm Contact Set 2 retired

The following is a sample output for the **alarm retire all** command:

Alarm Contact Set 1 already retired

Alarm Contact Set 2 retired

Alarm Contact Set 3 retired

alarm status

The **alarm status** command displays the status of Alarm Contact Sets.

Synopsis

alarm status ***[all|1|2|3...]***

Description

The **alarm status** command displays the state and status of the specified Alarm Contact Set. The numeric arguments refer to the alarm contact set. This command is used in conjunction with the *[all]* or *[n]* options (where *n* is the alarm contact set number).

Examples

The following is a sample output for the **alarm status 2** command:

Alarm Contact Set 2

```
Enabled: Yes  Status: off (open)
```

The following is a sample output for the **alarm status all** command:

Alarm Contact Set 2

```
Enabled: Yes  Status: off (open)
```

Alarm Contact Set 3

```
Enabled: No  Status: on (closed)
```

alarm test

The **alarm test** command manually initiates alarms.

Synopsis

alarm test *[all|1|2|3...]*

Description

The **alarm test** command tests the specified Alarm Contact Set for use. The command initiates external alarm by closing contacts on specified Alarm Contact Set whether set is enabled or not. This command is used in conjunction with the *[all]* or *[n]* options (where *n* is the alarm contact set number).

Examples

The following is a sample output for the **alarm test 2** command:

Alarm Contact Set 2 is now on (closed)

The following is a sample output for the **alarm test all** command:

Alarm Alarm Contact Set 1 already on (closed)
Alarm Contact Set 2 is now on (closed)
Alarm Contact Set 3 is now on (closed)

annotate

The **annotate** command annotates the transaction state machine (TSM) trace stream with a message.

Synopsis

annotate *[channel] <"message">*

Description

The **annotate** command sends a message to TSM requesting that the given message be put into TSM's trace stream. This command is useful for testing and debugging scripts.

If a channel is specified, the message is associated with the channel's trace stream. The message must be fewer than 160 characters.

The **annotate** trace message is displayed in the trace output if a trace is running when the **annotate** command is executed. If no **"trace"** command is running, the annotate trace message is discarded.

Files

/vs/bin/tools

Example

The following example sends a message to TSM to put the message "This is test 1 for channel 1" in channel one's trace stream.

annotate 1 "This is a test 1 for channel 1"

assign card/channel

The **assign card** command assigns a group number to a card.
The **assign channel** command assigns a group number to a channel.

Synopsis

assign card *<card [.port]>* to *[eqpgrp] <group number> [grpname]*

assign channel *<number>* to *[eqpgrp] <group number> [grpname]*

Description

The **assign card/channel** command is used when a system is installed, the number of channels or cards changes, scripts are added or deleted, telephone numbers change, or the user wants to reconfigure the system. The system uses the card and channel assignments to route an incoming call to the group.

The parameters that can be used with the **assign card/channel** command are:

- *number* — The channel number (a single card or channel number, a range of card or channel numbers specified m–n, or the word “all’ for all card or channel numbers)
- *eqpgrp* — The “eqpgrp” when assigning to an equipment group
- *group number* — The number of the equipment group or service group
- *grpname* — An optional character string that can be associated with “grp’“

Reference to a nonexistent channel or nonexistent group in this command causes it to fail.

Examples

The following example assigns channels 0 through 47 to equipment group 1.

assign chan 0-47 to eqpgrp 1

See Also

"assign service/startup"

"display eqpgrp/group"

"delete eqpgrp"

assign_permissions

The **assign_permissions** command assigns Voice Information System (VIS) security permissions to a user.

Synopsis

assign_permissions *<user login> <permissions level>*

Description

The **assign permissions** command assigns VIS security permissions to a user. Security permissions determine the areas of the VIS system that the user may access. For information on how user logins are created, see *Intuity CONVERSANT VIS Version 5.0 Operations*, 585-310-550.

The *<user login>* argument represents the user who is to be assigned security permissions.

The *<permissions level>* argument is the specific security class permission to be assigned. The security classes are as follows:

- Administration

  Allows the user full voice system capabilities

- Applications

  Allows the user Script Builder, configuration management, reports administration, and system monitor capabilities

- Operations

  Allows configuration management, reports administration, and system monitor capabilities

Example

The following example executes the command to assign VIS security to a user with the user login of brown.

assign_permissions brown operations

See Also

"unassign_permissions"
"display_permissions"

assign service/startup

The **assign service/startup** command assigns an installed service to DNIS and ANI numbers or directly to a channel.

Synopsis

assign service *<service_name>* [startup *<startup_name>*] to chan *<chan_list>*

assign service *<service_name>* to dnis *<phone_list>* [ani *<phone_list>*]

assign service *<service_name>* to ani *<phone_list>* [dnis *<phone_list>*]

Description

The **assign service/startup** command is used to assign services to either a set of channels or to a DNIS and ANI numbers. Services should be assigned after the service has been verified and installed, the number of channels changes or the system is reconfigured. Use the **display script** command to see a list of valid service names.

The *chan_list* variable indicates channel numbers or channel number ranges in the form *chan1-chan2*. A comma or space should be used to separate channel numbers in the list of channel numbers or ranges.

The *phone_list* variable indicates phone numbers or phone number ranges in the form *phone1:phone2*. A comma or space should be used to separate the list of phone numbers or ranges (for example, **phone1:phone2**)

The *phone number*

Examples

This example assigns service stdin (standard in as an arbitrary name for a script) to channel 0.

assign service stdin to chan 0

This example assigns service stdout (standard out as an arbitrary name for a script) to channel 1.

assign service stdout to chan 1

This example assigns service dnis to all channels.

assign service DNIS_SVC to chan all

This example assigns startup service stdout to channels 4 through 7

assign startup stdout to chan 4-7

This example assigns the service stdout and startup service stdin to channels 4 through 7.

assign startup stdin service stdout to chan 4-7

This example assigns the service stdout to DNIS 5000 through 5008 and ANI any.

assign service stdout to dnis 5000:5008

This example assigns the service stdout to DNIS 5000 through 5008 and ANI 6000.

assign service stdout to dnis 5000:5008 ani 6000

This example assigns the service stdout to DNIS any and ANI 6000 through 9000.

assign service stdout to ani 6000:9000

This example assigns the service stdout to DNIS 3000 and ANI 2000-3000.

assign service stdout to dnis 3000 ani 2000:3000

See Also

"display eqpgrp/group"

"delete eqpgrp"

assign_tty

The **assign_tty** command specifies which serial port to use for the Switching Control Center System (SCCS) monitoring and for the alarm relay unit (ARU).

Synopsis

assign_tty

Description

The **assign_tty** command permits the user to alter the ports assigned to be monitored by the SCCS and ARU. The command prompts for a device for each. Respond to the prompt with the device name; for example, tty00 or tty01.

The SCCS port and the ARU port can be the same port if the proper cabling is available. However, for greater reliability, it is recommended that you use separate ports.

NOTE:
If the port to which you assign the SCCS or ARU has a getty entry in **/etc/inittab**, it will be turned off. That is, the old getty entry is turned off. A new getty entry is needed.

Example

"assign_tty"

attach

The **attach** command attaches a unit (card).

Synopsis

attach *<unit> <number> [-i] [-n]*

Description

The **attach** command is used to attach a card that has been "detached." The unit (card) is logically attached by changing its permanent state from nonexistent (NONEX) to manual-out-of-service (MANOOS). To put the unit into service, use the **"restore"** command.

The parameters for the attach command are:

- *<unit>* — This option identifies the unit; the choices are "channel" or "card."
- *<number>* — This option specifies the channel or card number, a range of channel or card numbers in the form m–n, or the word "all" for all channel or card numbers. Card numbers are in the form *card#[.port#]* where *port#* is a port of the card *card#*. If *port#* is not given, all ports of the card specified are attached. If no card number or channel number is given, a syntax message is displayed.
- *-n* — This option disables prompting from the system whether to wait until a conflict has been resolved (see the *-i* option below) or to terminate the request to **attach**.
- *-i* — This option is used to enable secondary command registration. If T1 diagnostics are being run, this option allows the "attaching" of another card. If *-i* is used and another maintenance command is being run (**"remove"**, **"detach"**, "attach", **"restore"**, or **diagnose**), the request to **attach** is blocked and a message is printed to the screen. If *-i* is not used and any maintenance command is being run, the request to "attach" is blocked and a message is printed to the screen.

  If the command is permitted to run, it is determined if the command is in conflict with another command. A command is in conflict if the card or card associated with the command meets any of the following conditions:

  - T1 card is being diagnosed
  - Causes a change in the existing TDM bus master assignment
  - An interdependency exists with the T1 card being diagnosed (for example, PRI)

If one of the above conflicts exist and *-n* is not used, the user is asked whether to wait until the conflict is resolved or to terminate the request. If T1 diagnostics are executing on-line tests and a conflict is detected, the attach command is blocked. If T1 diagnostics are executing off-line tests and a conflict is detected, the user is asked whether to wait until the conflict is resolved or to terminate the request to attach.

To delete out of the command, press DEL. If this does not terminate the command, you may need to press CONTROL ALT DEL simultaneously. If, while running **attach**, you abort the command, a message similar to the following may appear:

At the user's request, administration of the following cmd(s) has been interrupted.

CARD NUMBERS: <card numbers>

To assure proper operation of the identified card(s), run diagnostics at the earliest opportunity.

It is recommended when **attach** is aborted, diagnostics be run on all cards being administered to ensure they are returned to a fully functional state.

Examples

The following example attaches a card to channel 2.

attach card 2

The following example attaches channels 0 through 2 and channel 5.

attach channel 0-2,5

The following example attaches a card to channel 2, port 1.

attach card 2.1

See Also

"detach"
"restore"
"remove"

autoreboot

The **autoreboot** command provides a means of changing or displaying the parameters associated with the VIS automatic reboot feature.

Synopsis

autoreboot *[enable|disable] [reboots <numbers>] [window <minutes>] [uptime <minutes>]*

autoreboot *[status|s]*

autoreboot *[help|h]*

Description

The **autoreboot** command may be used to change parameters associated with the VIS autoreboot feature and to monitor the status of these parameters. The following options are recognized:

- *enable|disable* — This option specifies whether to enable or disable the autoreboot feature. The default is enable.
- *reboots <number>* — This option specifies the number of unanticipated reboots tolerated within the time period specified by *window*. The default is 5.
- *window <minutes>* — This option specifies the time period for the *reboots* parameter. The default is 60 minutes.
- *uptime <minutes>* — This option specifies the amount of time that the system must be in service before the automatic reboot feature is activated. The default is 5 minutes.
- *status* — This option shows the current values of the automatic reboot parameters, plus the number of unanticipated reboots that occurred in the *window* minutes preceding the most recent system boot.

When the automatic reboot feature is enabled and activated, the system automatically reboots after a UNIX panic. The automatic reboot feature is activated as follows:

If there were fewer *reboots* than unanticipated reboots during the *window* minutes prior to the most recent system boot, the automatic reboot feature is activated (if enabled) *uptime* minutes after the most recent system boot.

For example, assume the automatic reboot parameters are set to their default values. A system crash occurs. The system reboots at 8:00. If there were fewer than 5 unanticipated reboots between 7:00 and 8:00, the automatic reboot feature is activated as 8:05. Otherwise, it is activated at 9:00.

An unanticipated reboot is a system boot that occurs after a system crash. A system crash can be caused (for example) by a UNIX panic, a system restart via RESET or a sudden power loss.

Example

The following example enables autoreboot feature and changes *window* to two hours:

autoreboot enable window 120

Caveat

This command must be run from ksh (KORN shell).

backup_appl

The **backup_appl** command backs up an application.

NOTE:
This command is valid only if the Enhanced File Transfer package is installed.

Synopsis

backup_appl -n <application name> ***[-d <database file>] [-t <transaction file>] [-s <speech file>] [-p <path>]***

Description

The **backup_appl** command is used to backup a Script Builder application to files on the local machine. The files in each component (database, speech, transaction) of the application are bundled into one cpio file per component. If the cpio file names and path are not specified, default names and a default path are used and all three components are backed up. The following are the default file names for each component:

databaseDbase

speechSpch

transactionTrans

Files

/tmp/sb/BkUpAppl/*<application name>*

Return Values

If the **backup_appl** command is successful, a 0 value is returned.If any value other than 0 is returned, the **backup_appl** command failed.The following are the possible reasons for failure for the **backup_appl** command:

- The hard disk is low in space.
- You are not logged in as **root** or a super user.
- The command syntax is incorrect.
- The backup tables has failed.
- The backup speech has failed.
- The backup transaction has failed.

Example

The following example backs up the “bank_balance” application using the default names for the transaction, database, and speech.

backup_appl -n bank_balance

See Also

"install_appl"
"remove_appl"
"restore_appl"

bbs

The **bbs** command reports status of the voice system Bulletin Board (BB).

Synopsis

bbs ***[-d] [-h] [-l]***

Description

The **bbs** command displays the field values of the BB slots. This information is stored in standard out (stdout). Without any options, information is extracted only from the dynamic portion of the BB and printed in short format. Otherwise the information displayed is controlled by following options:

d	This option prints information about the dynamic portion of the BB (the default).
h	This option prints information about the hardcoded portion of the BB.
l	This option generates a long listing. All fields are displayed.

The column headings and meaning of the columns in the **bbs** listing are given in Table 1-2. In the table, the letter **l** indicates the **long** option, which causes the corresponding heading to appear. The **all** option means that the heading always appears.

Table 1-2. bbs Column Headings

Column Name	Option	Description
SLT	(all)	The slot number
BBNAME	(all)	The name associated with process and slot
QKY	(all)	The message queue key
PID	(all)	The process ID
INS	(all)	The process instance
D	(all)	"YES" if process is a message-sending DIP type; otherwise "NO"
CDATE	(l)	The last process creation time
WK	(l)	The ET work state
SKEY	(l)	The semaphore key associated with process and slot
QID	(l)	The message queue ID
RE-SPA	(l)	The number of respawns from last restart of the voice system
WKCNT	(l)	The ET work count for process

Upon successful completion, **bbs** returns an exit status of zero. Otherwise, **bbs** prints an error message on stderr and returns a non-zero exit status if the voice system is not running, or if for some other reason, it can not access the BB.

Example

The following example prints a long listing, displaying all possible fields.

bbs -l

bk_appl

The **bk_appl** command backs up the speech or transaction component of a Script Builder application.

Synopsis

bk_appl *<a/s/t> <application_name> [0/1/2/f=filename]*

Variable	Definition
a	Used for backing up both the speech and transaction component of an application
s	Used for backing up the speech component only of an application
t	Used for backing up the transaction component only of an application
0	Indicates floppy drive 0 is to be used for backing up
1	Indicates floppy drive 1 is to be used for backing up
2	Indicates tape drive is to be used for backing up
f	Indicates information is to be backed up to a file; a filename must be designated

Description

The **bk_appl** command is used to back up a Script Builder application to a type of media on the local machine. This command can be used to backup either a single component (for example, speech or transaction) or both of the components (speech and transaction).

The **bk_appl** command supports backing up on floppy diskettes, magnetic tapes, and to a file. Two separate sets of floppy diskettes or a set of magnetic tapes is required in order to backup both of the components of an application. Only a single component can be backed up to a file.

NOTE:
When a file is used as a backup media, the backed up file is available either in the specified directory (file name is given with the full path name) or in the current working directory (only the filename is given) when this command is invoked.

See Also

"rs_appl"

ccarpt

The **ccarpt** command generates a call classification data summary report.

Synopsis

ccarpt *<date>*

ccarpt *<start_date> <end_date>*

Description

The **ccarpt** command generates a call classification data summary report. This report is stored in standard out (stdout).

The *<date>, <start_date>*, and *<end_date>* arguments are in the form mm/dd/yy.

Example

The first example generates the call classification data summary report for October 10, 1993. The second example generates the call classification data summary report from October 14 through October 20, 1993.

ccarpt 10/20/93
ccarpt 10/14/93 10/20/93

cddrpt

The **cddrpt** command generates a call data detail report.

Synopsis

cddrpt *<records> <service> <calldata> <date>*

Description

The **cddrpt** command generates the call data detail report. This report is stored in standard out (stdout). Before this can be done, the database system must be up and running, but the voice system does not need to be up.

The parameters for the **cddrpt** command are:

- *<records>* — This parameter represents the number of records to be reported. It can be any number, a range of numbers, or "all" indicating all records in the system.
- *<service>* — This parameter represents the script (application) name, or "all" for all applications.
- *<calldata>* — This parameter represents a flag indicating whether to include call event data or not. The valid options are either "n" for not including event data or "y" for including event data.
- *<date>* — This parameter is the date the data was collected in the system. The valid options are either a date in mm/dd/yy format or "all" indicating all records in the system.

Example

The following example generates a call data detail report for the first 100 call data collected on date October 20, 1993 for application "balance_chk." (Call event data if any is also included in the report.)

cddrpt 100 balance_chk y all 10/20/93

The following example generates a call data detail report for all call data in the system without including call event data.

cddrpt all all n all

cdsrpt

The **cdsrpt** command generates a call data summary report for a specific date.

Synopsis

cdsrpt *<hours> <service> <event data> <date>*

Description

The **cdsrpt** command generates the call data summary report for a date specified. The report is stored in standard out (stdout). Before this can be done, the database system must be up and running, but the voice system does not need to be up.

The parameters for the **cdsrpt** command are:

- *<hours>* — This parameter is the hour the call data was collected. It can be any number between 0 to 24 or “all” indicating all 24 hours.
- *<service>* — This parameter is the script (application) name, or “all” indicating all applications.
- *<event data>* — This parameter is a flag indicating whether to include call event data or not. The valid options are either “n” for not including event data or “y” to include event data.
- *<date>* — This parameter is the date the data was collected in the system (in format mm/dd/yy).

Example

The following example generates call data summary report for call data collected between 2 p.m. and 4 p.m. on date October 20, 1993 for all applications on the system. Call event data summary is included in the report.

cdsrpt 14-16 all y 10/20/93

The following example generates call data summary report for all call data collected on date October 20, 1993 for the application “balance_chk.” Call event data summary is not included in the report.

cddrpt all balance_chk n 10/20/93

change_device

The **change_device** command changes the presentation name of a device in the **/vs/data/device_data** file.

Synopsis

/vs/bin/util/change_device

Description

The **/vs/data/device_data** file, as it is initially populated, contains standard generic presentation names (user mnemonic names, see **"add_device"** in this book). The **change_device** command allows a user to change the user mnemonic name from the generic name to a possibly more specific name (for example, a comcode).

The command is interactive and menu driven; the user selects the generic name which they wish to change, then inputs the 2–20 character new name.

Upon the next invocation of the **/vs/bin/configure** program, the new names appear in the device selection menu and in all output generated by the program.

Files

/vs/data/device_data

See Also

"add_device"
"configure"
"get_config"
"remove_device"
"save_config"
"show_devices"
"show_config"

checktf

The **checktf** command checks for the existence of talkfiles in the voice system.

Synopsis

checktf ***[all | <talkfile> [<talkfile>...]]***

Description

The **checktf** command checks for specific talkfiles in the voice system. If the specified talkfile is being used by an existing application **checktf** displays the application name. If no application is associated with the talkfile, **checktf** displays a corresponding message. If the talkfile is currently not being used, checktf exits with a value of **0**. Otherwise, it exits with the number of talkfiles that were in use from the specified list. Other error exit codes are **199** (incorrect usage) and **200** (voice system is running).

Example

The following example checks for talkfile 41 in the system.

checktf 41

chg_machname

The **chg_machname** command changes the name of the machine the SCCS is monitoring.

Synopsis

chg_machname

Description

The **chg_machname** command allows you to change the name of the machine that the the SCCS is monitoring. This name is used in the header of the messages sent to SCCS. You will be prompted for the name of the machine. Use this command if an error occurred during installation in entering the name of the machine or if the current machine name is unknown.

NOTE:
You must stop and restart the VIS for the change to take effect in your system.

Example

"chg_machname"

codetype

The **codetype** command identifies the type of coding header in a speech file.

Synopsis

codetype ***file***

Description

The **codetype** command identifies the type of coding header that is present in a VIS speech file. Codetype recognizes PCM64, ADPCM32, ADPCM16, SBC24, SBC16, or CELP16 headers.

See Also

"addhdr"
"striphdr"

configure

The **configure** command determines allocation of resources for all devices to be included in a VIS system configuration for a given hardware platform.

Synopsis

/vs/bin/util/configure [new]

Description

The **configure** command is an interactive command which automatically determines allocation of the following resources within a Intuity CONVERSANT VIS system:

- Slot number
- Interrupt
- Direct memory address (DMA) channel
- Input/output (IO) address
- Random access memory (RAM) address
- Serial port number
- Parallel port number

The program is mainly for use by factory personnel for determining the configuration of a system being built for a customer. It is also useful in the field as a tool for field service personnel who are upgrading (adding new devices to) an existing configuration. All user input is in the form of user friendly prompts or menu selections.

All output from a successful configuration is written to the **/vs/data/conf_data** file. An unsuccessful or incomplete configuration is written to the **/vs/data/fail_data** file so that it may be examined to see why the configuration was not successful. Output to these files is in a compressed format. The **/vs/bin/util/show_config** command is used to view the configurations represented by these files.

The program allows the user to choose the hardware platform they are attempting to configure, followed by all devices which are to be included in the configuration. This ensures that all required devices for the platform are selected.

The program has built-in rules which prevent users from entering devices that are not supported on the hardware platform they have selected. It also checks hardware feature rules, such as how many of a particular device are supported

on the chosen platform. In all cases of device rejection by the program, the user is prompted with a message clearly explaining why the device is being rejected.

Upgrading an Existing Configuration

When **configure** is executed with no argument, it checks to see if a **/vs/data/conf_data** file currently exists. If so, the configuration represented by this file is read and used as the base configuration. This is useful in the case of a field upgrade to an existing VIS. A copy of the **conf_data** file is saved in **/vs/data/conf_MMDDYY** where MM=month, DD=day and YY=year. The existing **conf_data** file is saved because, upon determination of a valid configuration which includes the user's newly specified devices, the current **conf_data** file is replaced by a new one. It may be useful at times to see what the previous configuration looked like. An option on the **/vs/bin/util/show_config** command allows the user to view the configuration represented by the saved conf_MMDDYY file.

Once the file is read, the user is presented a menu of devices they may attempt to add to the current configuration. Once the new devices are specified, the program attempts to allocate resources to the new devices from the pool of resources not currently used by devices already in the configuration. This may be a two pass process:

- PASS 1: An attempt is made to fit the newly specified devices into the current configuration without disturbing any devices currently configured. If this can be done, Pass 2 is not executed.
- PASS 2: If Pass 1 was unsuccessful, Pass 2 attempts the equivalent of a new configuration, unassigning all currently used resources, and pooling the newly selected devices with those already in the configuration. If Pass 2 is successful, the user may be required to change the switch settings on some of the cards already in the system, move them to different slots, etc. If Pass 2 is unsuccessful, the newly specified devices will not fit into the current configuration.

Specifying a New Configuration

If **configure** is executed with no argument and a **/vs/data/conf_data** file does not exist, a menu of currently supported hardware platforms is presented, followed by the menu of devices which may be configured with that platform. The user selects the platform and devices desired, indicating when finished. The program attempts to allocate resources to each device selected. If successful, the program terminates. Otherwise, the user is asked if they wish to remove something and try again.

The **configure** command may be executed with an argument of "new." This forces a new configuration, as described above, even if a **/vs/data/conf_data** file exists.

Presetting Device Hardware Resources

You may wish to preset certain resources of a single new device being selected for a configuration. It may be desired for example to force the **configure** program to select interrupt 6 for a a particular device being specified. Whenever a single device is specified, the following prompt is shown to the user:

Do you wish to preset any hardware options of device_name? [y|(n)]

If "y" is specified, the user is allowed to preset any of the following hardware attributes of the selected device (where applicable):

Interrupt
DMA Channel
IO Address
RAM Address

When the user indicates that they are finished selecting devices, the **configure** program continues as normal. In the case of a new configuration, if a valid configuration is determined, the program terminates normally. If a valid configuration cannot be determined, a message indicating this is displayed and the program terminates. The user is not prompted to remove something and try again if any hardware option for any device was preset.

In the case of an upgrade, the program attempts Pass 1 as described above. If successful, then the program terminates normally. If Pass 1 is unsuccessful, a message indicating this is displayed. Pass 2 is not attempted if the user has preset any hardware options for any device.

Device Data

The devices presented to the user in the menus described above are kept in the **/vs/data/device_data** file. This file is in compressed format. The **/vs/bin/util/show_devices** command may be used to view its contents. All attributes of each device are stored in this file.

It may be desirable from time to time to add a new device to those which are configurable in a VIS machine by the **configure** program. The **/vs/bin/util/add_device** prompts a user for all necessary attributes required to add the new device to the **device_data** file. Once added to the file, the new device is available to the configure program.

Devices which are added to the **device_data** file may also be removed. The **/vs/bin/util/remove_device** program provides this capability.

The names by which devices are presented in the menus described above may be changed to suit a particular user's needs. The **device_data** file is shipped with default generic user names. The **/vs/bin/util/change_device** program allows a user to name devices according to user preference.

Files

/vs/data/device_data
/vs/data/conf_data
/vs/data/fail_data

NOTE:
Upon reaching the first unresolvable resource conflict, the **configure** program terminates (unless the user selects the **"remove"** option, in the case of a new configuration), and writes the incomplete configuration to **/vs/data/fail_data**. A message indicating which device and which hardware resource caused the conflict is displayed.

The format of the **/vs/data/conf_data** file and the **/vs/data/device_data** file is very specific. A user should not edit or alter these files.

The **configure** program should only be run by persons familiar with VIS system configurations and hardware platforms. Refer to Chapter 4, "Running the Configuration Program," in the hardware installation book for your platform for slot position and numbering information. Refer to the documentation on each device for the switch or jumper settings which correspond to the hardware resources determined by the **configure** program.

Serial and parallel port numbering is used for allocation purposes only. There is no correlation between the port number specified in the configuration output and actual physical ports. In configurations where there are multiple ports, any of the available ports may be used for those devices which require one.

See Also

"add_device"
"change_device"
"get_config"
"remove_device"
"save_config"
"show_config"
"show_devices"

console_off

The **console_off** command turns the flow of error messages to the console off during system operation.

Synopsis

console_off

Description

Output to the console from the CompuLert/SCCS interface package is normally disabled and handled directly by the VIS error logging system. However, if you set the CONSOLE_OUTPUT_DEFINED parameter in the default parameter file to *TRUE*, you can use the **console_off** command to turn the flow of error messages to console off during system operation.

See *Intuity CONVERSANT VIS Version 5.0 Communications Development*, 585-310-229, in Chapter 8, “Data Network Connectivity Alarms,” for instructions on changing the CONSOLE_OUTPUT_DEFINED parameter.

Examples

"console_off"

console_on

The **console_on** command turns the flow of error messages to the console on during system operation.

Synopsis

console_on

Description

Output to the console from the CompuLert/SCCS interface package is normally disabled and handled directly by the VIS error logging system. However, if you set the CONSOLE_OUTPUT_DEFINED parameter in the default parameter file to *TRUE*, you can use the **console_on** command to turn the flow of error messages to console on during system operation.

See *Intuity CONVERSANT VIS Version 5.0 Communication Development*, 585-310-229, in Chapter 8, “Data Network Connectivity Alarms,” for instructions on changing the CONSOLE_OUTPUT_DEFINED parameter.

Examples

"console_on"

copy

The **copy** command copies a phrase from a UNIX talkfile to a UNIX talkfile.

Synopsis

copy phrase *<phrase number>* from talkfile *<talkfile number>* to *<filename>*

Description

The **copy** phrase command copies a phrase from one UNIX talkfile to another UNIX talkfile. The path name for the file may be the full path name or the relative path name. If no path is specified, the file is created in the current working directory. If you are not in the directory in which the phrase to be added is stored, be sure to give the full path name for the talkfile and source file.

NOTE:
Only the login root can copy a phrase to any of the root directories. Users without root permission can copy phrases only to directories for which they have permission, usually under their login id.

NOTE:
The **copy** command copies a phrase from a UNIX talkfile within the SPEECHDIR default directory (**/home2/vfs/talkfiles**) to a UNIX file.

Example

The following example copies phrase number 2 from talkfile 1 to the file **/speech/talk/a.1**.

copy phrase 2 from talkfile 1 to /speech/talk/a.1

The following example copies phrase number 174 from talkfile 25 to the file **/speech/talk/h.4**.

copy phrase 174 from talkfile 25 to /speech/talk/h.4

See Also

"add"
"erase"
"list"

cpuType

The **cpuType** command returns the type of central processing unit (CPU) used in the system.

Synopsis

cpuType

Description

The **cpuType** command returns the type of CPU on the system, either a 386 or a 486. If the **cpuType** command returns a 3, you are using a 386. If the cpuType command returns a 4, you are using a 486. To determine the return value, examine the shell variable $?.

cvis_mainmenu

The **cvis_mainmenu** command accesses the Intuity CONVERSANT administration menus.

Synopsis

cvis_mainmenu

Description

The **cvis_mainmenu** program is a menu interface used to access the Voice System Administration menus. For more information about the menus available from Voice System Administration, see *Intuity CONVERSANT VIS Version 5.0*, 585-310-550.

See Also

"cvis_menu"

cvis_menu

The **cvis_menu** command accesses the Intuity CONVERSANT VIS Voice System Administration menus.

Synopsis

cvis_menu

Description

This is the command which provides access to the administration of the Intuity CONVERSANT Voice Information System. See *Intuity CONVERSANT VIS Version 5.0 Operations*, 585-310-550, for information about administering your VIS.

See Also

"cvis_mainmenu"

dbcheck

The **dbcheck** command checks the resources available in the database (Version 6.0 ORACLE).

Synopsis

dbcheck -i

dbcheck r

dbcheck ***[w n[,m]] [-s] [-e] [-m user[~user...]]***

Description

The **dbcheck** command checks spaces, usage, and rollback segment growth. The **dbcheck** command has three different usages.

The *-i* option installs cron entries (optional) to run **dbcheck** at regular intervals and support for logger/alerter messages. (The *-i* option only needs to run once). The cron job can be placed in either roots cron file or added to the end of the **/vs/bin/util/croncdh** job that runs once a day. The *-i* option also asks if you want new alerter messages added to the logger/alerter database along with explanations used with the explain command. This installation only needs to be run if you want the warnings to show up in the system event log or if you want to schedule automatic checking at regular intervals.

The *-r* option removes any cron entry set up by the *-i* option.

The third usage actually checks database space against a user set “water marks.” Three different things are checked:

- Free space
- Extents against the user set threshold *n* (15% default)
- Rollback segment(s) growth against the user set threshold *m* (20% default)

When executed, the **dbcheck** command generates the appropriate warnings (shown under “Diagnostics” below) if the database falls below *n* percent free or if the rollback segment grows to be more than *m* percent of the total database size.

The **dbcheck** command, by default, sends warning messages to the logger/alerter indicating a threshold has been exceeded (the *-i* option must be run first). The -e option disables the entries from going into the log file. The *-s* option prints the warning messages to standard output. The *-m user* option allows for the messages to be mailed to *user*. Multiple users can be sent the mail by separating the user names with ~. Below are sample outputs.

(Output to error log when less than 13% available space/extents or more than 23% used by rollback)

```
# dbcheck -w13,23

*  Mon Feb 15 16:35:06 1993 dbcheck logTest.c:418
DBC001    -- -- ---  Database 10 percent free, 3072 Blocks of 30720 available.
                Reason:  Low DB Space.
*   Mon Feb 15 16:35:06 1993 dbcheck logTest.c:418
DBC002     -- -- ---  Extents low, 100 used of 121, on object MY_TABLE
                Reason:  Low DB Extents
*   Mon Feb 15 16:35:06 1883 dbcheck logTest.c:418
DBC003     -- -- --- Rollback segments=7680 blocks, 25 percent of total space.
                Reason:  High Rollback Usage.
```

Files

LOGROOT=$[LOGROOT:-"/usr/spool/log"}
${LOGROOT}/head/logDBC.h
${LOGROOT}/formats/DBCmsg
${LOGROOT}/formats/formats.mk
${EXPLAINDR}//translateLst
/vs/bin/util/croncdh
/usr/spool/cron/crontabs/root
/usr/spool/cron/crontabs/root.bu

Diagnostics

The **dbcheck** command returns the following values:

0Success, no limits exceeded

1Threshold exceeded

2Processing error

3Database is not running

Caveat

Once dbcheck log messages are installed, using **dbcheck -i**, the alarm priorities, destinations, and thresholds can not be changed through the System Message Display screen as described in Chapter 3, “Configuration Management,” of *Intuity CONVERSANT VIS Version 5.0 Operations*, 585-310-550.

See Also

"dbfrag"
"dbfree"
"dbused"
"explain"
"logCat"

dbfrag

The **dbfrag** command lists fragmentation information on the database (Version 6.0 ORACLE).

Synopsis

dbfrag ***[-h -b]***

Description

The **dbfrag** command is a shell script that reports on database allocation, usage, and fragmentation. The block size reported is in ORACLE blocks (2048 bytes). You can request the information to be reported in Mbytes with the *-b* option. This tool is useful to get a quick check on database usage and provides a shell interface into some key ORACLE statistics.

This tool only reports on information in the 'SYSTEM' tablespace. With the *-h* option, the listing will be printed without a header. This option is useful if you want to parse this output to select a specific field.

The following requests fragmentation information in Mbytes (using the *-b* option).

 # dbfrag -b

SYSTEM Tablespace, Space is in Mega-Bytes

ALLOCATED	FREE	% FREE	AVG/FRAG	LARGEST	FRAGMENTS	DB_FILES	ROLLBACK
129.00	108.88	84.40	5.44	108.12	20	1	7.91

Examples

The following example gets the largest contiguous ORACLE space available.

```
dbfrag -h | awk 'length>1 {print $5}'
10240
```

Diagnostics

The program returns the following:

0Success

1Processing Error

See Also

"dbcheck"
"dbfree"
"dbused"

dbfree

The **dbfree** command checks the space available in the database by partition (Version 6.0 ORACLE).

Synopsis

dbfree ***[h]***

Description

The **dbfree** command is a shell script that lists the amount of free space in the database by free contiguous blocks. The result is a detailed listing of each free memory area followed by the sum of each partition. The free blocks are listed in 2048 bytes/block (ORACLE blocks). There is also a column that lists the same information in Mbytes. The *-h* option removes the column headers. Below is a sample output of the **dbfree** command.

Contiguous extents

TABLE SPACE NAME	FILE_ID	START_BLOCK	MBYTES FREE	ORACLE BLOCKS FREE
SYSTEM	1	5142	.02	12
SYSTEM	1	5560	.03	13
SYSTEM	1	4892	.04	18
SYSTEM	1	7892	.04	19
SYSTEM	1	4164	.05	28
: :	: :	: :	: :	: :
SYSTEM	1	5598	.73	375
SYSTEM	1	8946	4.00	2048
SYSTEM	1	12650	4.45	2277
SYSTEM	1	25179	10.00	5120
SYSTEM	1	14939	20.00	10240
sum			47.18	24070

29 rows selected.

Diagnostics

The program returns the following values:

0Success

1Processing Error

Caveats

The **dbfree** command creates a temporary table “dba_fragments” under user system that compresses the adjacent entries provided by the dictionary view “dba_free_space.”

See Also

"dbfrag"
"dbcheck"
"dbused"

dbused

The **dbused** command provides database use by oracle user (Version 6.0 ORACLE).

Synopsis

dbused *[hs] [u <uid/passwd>]*

Description

The **dbused** command is a shell script that shows the amount of space used by each object for a given user. Objects are tables, indexes, clusters, rollback, and cache. The default user is sti/sti. The *-s* option reports summary information grouped by objects. The special user "all" reports information for the entire database. The *-h* option skips the header message. This option is useful if you are parsing. The *-u <uid/passwd>* option allows the user to specify the oracle user id and password (the default is sti/sti, all for all users).

Below is an output summary for user "all."

 # dbused -su all

Space allocated to objects. Oracle blocks (2048 Bytes/Block)

TYPE	BLOCKS	MBYTES	EXTENTS	OBJECTS
CACHE	18	.04	1	1
CLUSTER	2843	5.55	41	8
INDEX	1530	2.99	200	113
ROLLBACK	4049	7.91	24	3
TABLE	1860	3.63	172	102
sum	10300	20.12	438	227

Below is output for user “sti.”

 # dbused

Space allocated to objects. Oracle blocks (2048 Bytes/Block)

NAME	TYPE	BLOCKS	MBYTES	EXTENTS	MAX_EXTENTS
C1	INDEX	5	.01	1	99
CCA	TABLE	5	.01	1	99
CCASUM	TABLE	5	.01	1	99
CDH	TABLE	5	.01	1	99
CDHSUM	TABLE	5	.01	1	99
E2	TABLE	5	.01	1	99
EVENTS	TABLE	5	.01	1	99
EVSUM	TABLE	5	.01	1	99
LDBCOLS	TABLE	5	.01	1	99

Diagnostics

The program returns the following values:

0Success

1Processing Error

See Also

"dbfrag"
"dbfree"
"dbcheck"

decode

The **decode** command converts adpcm16 or adpcm32 files to pcm64 files.

Synopsis

decode ***[adpcm32/adpcm16]***

Description

Decode is a filter that converts ADPCM16 or ADPCM32 files to PCM64 files.

Warning

Coding headers should be stripped (using the **stripdhr** command) before running **decode**.

See Also

"addhdr"
"codetype"
"encode"
"striphdr"

defservice

The **defservice** command defines an IRAPI service.

Synopsis

defservice *[-h]* [-n] [-s *<servicename>*] [-p *<process>*] [-t *P | T*] [-a *0 |1 |2 |3 |4*] [*<application>*]

Description

The **defservice** command is intended to be used by IRAPI application developers to create the registration file for an IRAPI service that is necessary for assigning/deleting service to/from a channel or DNIS and/or ANI. For TSM scripts, the output of the **"tas"** command serves as the registration file for the script.

If the **defservice** command is entered with no options, **defservice** prompts you for all of the necessary information. You will need to respond to fewer prompts if you enter the majority of the information from the command line.

The *-h* option allows you to print the usage statement and then exit.

The *-n* option uses the default values for all options not specified on the command line. However, no defaults exist for the *<process>* and *<application>* parameters.

When the application is started by the Application Dispatch (AD) process, the **IRP_SERVICE_NAME** is set to the *-s <servicename>* argument if *<servicename>* is non-*NULL*. Otherwise, **IRP_SERVICE_NAME** is set to *<application>*, where the default is *NULL*.

The *-t* option specifies whether the process that provides the IRAPI application *<application>* is a permanent (P) or transient (T) process. The default is **P** for permanent.

If the process that provides the IRAPI application *<application>* is a permanent process, then *-p <process>* must be the name the process uses as an argument to *irRegister(3irAPI)*. If the process that provides the IRAPI application *<application>* is a transient process, then *-p <process>* must be the full pathname of the process. No default exists for this option.

When the application is executed on a PRI line, the *[-a 0 |1 |2 |3 |4]* option specifies how the ANI should be supplied to the application. The valid values for this option are as follows:

- 0 (No ANI supplied) — the default
- 1 (ANI type billing number only)

- 2 (ANI type billing number preferred)
- 3 (ANI type calling party (SID) only)
- 4 (ANI type calling party (SID) preferred

The <*application*> argument specifies the IRAPI application. No default exists for this argument.

Upon successful completion, the **defservice** command creates the **/vs/trans/*<application>*.T** file.

Files

/vs/trans/*.T

See Also

assign
delete
"tas"
iRAPI -AD(4irAPI-AD)
irRegister(3irAPI)

delete card/channel

The **delete card/channel** command removes a card or channel from a service or an equipment group.

Synopsis

delete card *<card.[port]> from [eqpgrp] <group number>*

delete channel *<number> from [eqpgrp] <group number>*

Description

The **delete card/channel** command removes the specified card or channel from a service or equipment group. The parameters for the delete card/channel command are:

- *<card.[port]>* — Specifies the card/channel number (a single card/channel number from a range of 0–255, a range of card/channel numbers in the form m–n, or the word “all” for all card/channel numbers).
- *eqpgrp* — Specifies “svcgrp” when deleting from a service group or “eqpgrp” when deleting from an equipment group. If no group type is given, the “svcgrp” is assumed.
- *<group number>* — Identifies the equipment group or service group.

If you want to remove all cards or channels from a equipment group, it may be easier to delete the entire equipment group than to delete channels or cards. To delete an equipment group, use the **"delete eqpgrp"** command.

Example

The following example deletes card 4 from service group 1.

delete card 4 from svcgrp 1

The following example deletes channels 10 through 13 from equipment group 3.

delete channel 10-13 from eqpgrp 3

See Also

"delete eqpgrp"
"delete service/startup"

delete eqpgrp

The **delete eqpgrp** command removes an equipment group.

Synopsis

delete eqpgrp *<group number>*

Description

The **delete eqpgrp** removes an equipment group. The *<group number>* argument is the equipment group list. To remove all equipment groups, use the word “all” as the group number.

Example

The following example removes equipment group number 3.

delete eqpgrp 3

The following example removes all equipment groups.

delete eqpgrp all

See Also

"assign card/channel"

delete service/startup

The **delete service/startup** command unassigns the assignment of a service to DNIS and ANI numbers or of a service assigned directly to a channel.

Synopsis

delete service *<service_name>* [startup *<startup_name>*] from chan *<chan_list>*

delete startup *<startup_name>* [service *<service_name>*] from chan *<chan_list>*

delete service *<service_name>* from dnis *<phone_list>* [ani *<phone_list>*]

delete service *<service_name>* from ani *<phone_list>* [dnis *<phone_list>*]

Description

The **delete service/startup** removes the specified telephone number or channel from the group to which a script is assigned. The parameters for the **delete service/startup** command are:

- *application name* — Specifies the name of application.
- *<chan | dnis>* — Specifies the name of the service group.
- *<chan number | phone number>* — Contains a list of one or more channels or telephone numbers separated by blanks. The word “any” or “all” shows that service is removed from all calls regardless of what number was dialed.

The *chan_list* variable indicates channel numbers or channel number ranges in the form *chan1-chan2*. A comma or space should be used to separate the list of channel numbers or ranges.

The *phone_list* variable indicates phone numbers or phone number ranges in the form *phone1:phone2*. A comma or space should be used to separate the list of phone numbers or ranges.

NOTE:
Only telephone numbers that have been assigned using the **"assign service/startup"** command can be deleted.

Example

This example deletes startup service stdout from channels 4 through 7

delete startup stdout from chan 4-7

This example deletes the service stdout and startup service stdin from channels 4 through 7.

delete startup stdin service stdout from chan 4-7

This example deletes the service stdout to DNIS 5000 through 5008 and ANI any.

delete service stdout from dnis 5000:5008 and ANI any

This example deletes the service stdout from DNIS 5000 through 5008 and ANI 6000.

delete service stdout from dnis 5000:5008 ani 6000

This example deletes the service stdout from DNIS any and ANI 6000 through 9000.

delete service stdout from DNIS any and ani 6000:9000

This example deletes the service stdout from DNIS 3000 and ANI 2000-3000.

delete service stdout from dnis 3000 ani 2000:3000

See Also

"assign service/startup"
"display services"
"display dnis"

detach

The **detach** command places a unit in the nonexistent state.

Synopsis

detach *<unit> <number> [-i] [-n]*

Description

The **detach** command places a unit currently in the manual-out-of-service (MANOOS) state into the nonexistent (NONEX) state. Before this can be done, the unit must be taken from the in-service (INSERV) or broken (BROKEN) state and put in the MANOOS state using the remove command.

The parameters for the **detach** command are:

- *<unit>* — Identifies the unit. The choices are “channel” or “card.”
- *<number>* — Specifies the channel or card number, a range of channel or card numbers in the form m–n, or the word “all” for all the channel or card numbers. Card numbers are in the form card#[.port#] where *port#* is the port of the card *card#*. If *port#* is not given, all ports of the card specified are detached. If no card number or channel is given, a syntax message is displayed.
- *-n* — This optional parameter disables prompting from the system whether to wait until a conflict has been resolved (see the *-i* option below) or to terminate the request to detach.
- *-i* — This optional parameter is used to enable secondary command registration. If T1 diagnostics are being run, this option allows the “detaching” of another card. If *-i* is used and another maintenance command is being run ("remove", "detach", "attach", **"restore"**, diagnose), the request to detach is blocked and a message is printed to the screen. If *-i* is not used and any maintenance command is being run, the request to detach is blocked and a message is printed to the screen.

  If the command is permitted to run, a check is made to see if the command is in conflict with another. A command is in conflict if the card or card associated with it:

  1. Is the T1 card being diagnosed
  2. Will cause a change in the existing TDM bus master assignment
  3. Has an interdependency with the T1 card being diagnosed (for example, PRI)

  If one of the above conflicts exist and *-n* is not used, the user is asked whether to wait until the conflict is resolved or to terminate the request. If T1 diagnostics are executing on-line tests and a conflict is

detected, the detach command is blocked. If T1 diagnostics are executing off-line tests and a conflict is detected, the user is asked whether to wait until the conflict is resolved or to terminate the request to detach.

To delete out of the command, press DEL. If this does not stop the command, you may need to press CTRL and backslash simultaneously. If, while running detach, you wish to abort the command, a message similar to the following may appear:

```
At the user's request, administration of the following
cmd(s) has been interrupted.

CARD NUMBERS: <card numbers>

To assure proper operation of the identified card(s),
run diagnostics at the earliest opportunity.
```

It is recommended when detach is aborted, diagnostics be run on all cards being administered to ensure they are returned to a fully functional state.

Example

The following example detaches card 4 and places it in the nonexistent state as far as the system is concerned.

detach card 4

The following example detaches channels 1 through 3 and places them in the nonexistent state as far as the system is concerned.

detach channel 1-3

See Also

"attach"
"remove"
"restore"

diagnose bus

The **diagnose bus** command tests a bus while it is in service.

Synopsis

diagnose bus *<bus number> <immed>*

Description

The **diagnose bus** command tests the TDM bus while it is in service. The *<bus number>* option is the number of the bus you want to diagnose. The valid option for the number argument is 1. If the *immed* option is used, any calls currently being processed are dropped immediately.

This command changes the temporary state of a unit to diagnostic (DIAG). If a unit fails the diagnostics, the permanent state is changed to BROKEN; otherwise, the permanent state is unchanged.

If the bus passes diagnostics, a message such as the following is displayed:

Diag TDM n: Diagnostics completed.

You may also receive a message like the following:

Diag TDM n: Diagnostics bus n failed.

To delete out of this command, press DEL. If this does not stop the command, you may need to press CTRL and backslash simultaneously.

Example

The following example diagnoses bus 1.

diagnose bus 1

diagnose card

The **diagnose card** command tests a card while it is in service.

Synopsis

diagnose card *<card number> [option]...*

Description

The **diagnose card** command is done at the card level for any card in the system. The *<card number>* option is the card number you want to diagnose. The word “all” can be used to specify all cards.

This command changes the temporary state of a unit to diagnostic (DIAG). If a card is stuck in the INSERV state, use the **diagnose card <number> immed** command. This temporarily removes the unit from the busy state unconditionally and places it in the manual-out-of-service (MANOOS). Note that any calls on the card when the “immed” option is used are dropped immediately.

For T1 cards the valid options are:

- *-n* — Prevents prompting from the system during diagnostic tests. The diagnostics assume the default values during the test and the user is informed when the diagnostics are completed.
- *-i* — Enables secondary command registration.See the description of *-i* for T/R and SP cards below.

For T/R and SP cards, the valid options are:

- *-n* — Disables prompting from the system whether to wait until a conflict has been resolved (see the *-i* option for T/R and SP cards below) or to terminate the request to diagnose.
- *-i* — Enables secondary command registration. If T1 diagnostics are being run, this option allows the diagnose of another card to be performed. If *-i* is used and another maintenance command is being run (**"remove"**, **"detach"**, **"attach"**, **"restore"**), the request to diagnose a non-T1 card is blocked and a message printed to the screen. If *-i* is not used and any maintenance command is being run, the request to **diagnose card** is blocked and a message printed to the screen.

  If the command is permitted to run, a check is made to see if the command is in conflict with another. A command is in conflict if the card or card associated with it:

  1. Is the T1 card being diagnosed
  2. Will cause a change in the existing TDM bus master assignment

3. Has an interdependency with the T1 card being diagnosed (for example, PRI)

If one of the above conflicts exist and *-n* is not used, the user is asked whether to wait until the conflict is resolved or to terminate the request. If T1 diagnostics are executing on-line tests and a conflict is detected, the "diagnose card" command is blocked. If T1 diagnostics are executing off-line tests and a conflict is detected, the user is asked whether to wait until the conflict is resolved or to terminate the request to diagnose.

If a unit fails the diagnostics, the permanent state is changed to BROKEN. If the unit being diagnosed previously was marked BROKEN and it passes diagnostics, it is put in the MANOOS state. Otherwise, the permanent state is unchanged.

When diagnostics are complete, T1 and SP cards are reinitialized and the appropriate software is downloaded to the cards.

For T/R cards, additional diagnostics first check the hardware status of the card specified. Then the system tests for dial tone on the card's channels not in the NONEX state. The result of the dial tone test is one of the following:

1. Nonex state — This channel is not checked for dial tone.
2. Dial Tone Found — This channel is operational.
3. NO Loop Current — The hardware cannot find telephone loop-current from the Central Office, PBX, or ACD. There probably is no phone line on this port.
4. NO Dial Tone — The hardware has found telephone loop-current but could not detect dial tone on this port.

If at least one channel detects dial tone, the entire card detects these frequencies as dial tone. If no channels detect dial tone, the card defaults to 330 and 440 Hz. The outcome of the dial tone tests do not affect the pass or fail results of the diagnostics. If no loop current is detected on a channel and the channel passed diagnostics, and the card is not MANOOS, the channel is placed in FOOS. In this case, the card does not become IDLE regardless of its previous state.

If a T/R and/or SP card passes diagnostics, a message such as the following is displayed:

Diagnose <card> n, Passed.

If a T1 card passes diagnostics, a message such as the following is displayed:

All tests passed.

You may also receive a message for a T/R and/or SP card saying:

Diagnose <card> n, failed <reason>

If a T1 card fails diagnostics, a help screen is provided giving you information to help resolve the reason for the failure. If you try to diagnose cards that are not installed in the system or if they are installed but are in the nonexistent state, an error message is displayed.

To delete out of the command, press DEL. If this does not stop the command, you may need to press CTRL and backslash simultaneously. Be aware, however, that this fixes the console, but does not terminate the diagnostic routine. If, while running diagnose, you wish to abort the command, a message similar to the following may appear:

At the user's request, administration of the following cmd(s)has been interrupted.

CARD NUMBERS: <card numbers>

To assure proper operation of the identified card(s), run diagnostics at the earliest opportunity.

It is recommended when **diagnose** is aborted, diagnostics be run again on all cards being administered to ensure they are returned to a fully functional state.

Examples

The following example runs diagnostics on card number 3.

diagnose card 3

The following example runs diagnostics on cards 4 through 7.

diagnose card 4-7

This example runs diagnostics on cards 4 through 7 immediately, dropping all calls currently in progress.

diagnose card 4-7 immed

dip_int

The **dip_int** command sends DIP interrupt to a script on a channel or a range of channels.

Synopsis

dip_int *<channel>*

dip_int *<channelStart-channelEnd>*

Description

The **dip_int** command sends a message or messages to TSM requesting that TSM send interrupt messages to the script running on *<channel>* or the range of channels *<channelStart-channelEnd>*. If no script is running on the channel or if TSM does not own the channel, no action is taken for the channel. The **dip_int** command does not wait for a response from TSM. Scripts running on the channel receive the EDIPINT event.

CAUTION:
Be careful when you use this command. It may affect other applications running on the system.

Example

The following example requests that TSM send interrupt messages to channel two.

dip_int 2

The following examples requests that TSM send interrupt messages on channels one through 32.

dip_int 1-32

Return Values

If the dip_int is successful, a 0 value is returned. If any value other than 0 is returned, the dip_int command completely or partially failed.

If dip_int returns a value of 2, then dip_int failed due to temporary condition. In this case, the user should attempt the dip_int command again.

See Also

"soft_disc"

display assignments

The **display assignments** command displays the services assigned to channels.

Synopsis

disp assignments *[<option>] [<option>]*

Description

The **display assignments** command is used to display all the services and startup services assigned to channels. The display assignments command options are as follows:

Option	Description
all (default)	Displays information on all services
<service name>	Displays channels assigned with a specific service.
startup <startup name>	Displays channels assigned with a specific startup service
channel <chan#>	Displays assignments for channel specified by *chan#.* A range of channels can be specified.

NOTE:
If more than one option is used, only channels that satisfy all the options given are displayed. If an invalid combination of options is given, an error message is displayed.

Example

The following example displays information for channel 1:

disp assignments channel 1

The following example displays information for all channels that have the service xxx assigned:

disp assignments xxx

The following example displays information for all channels that have the startup service xxx assigned:

disp assignments startup xxx

display card

The **display card** command displays information about specified cards.

Synopsis

disp[lay] card *[<option> [option]]*

Description

The **display card** command displays data about a specified card or about cards in a specified state.

The display card command options are:

- *<card#>[.port#]* — Displays information on card *<card#>* and on port *<port#>* of the specified card. All ports are shown if *<port#>* is not given. A range of cards may be specified in the form m–n without using the *<port#>* option.
- *all* — Displays information on all cards.
- *mtc* — Displays all cards being used by the maintenance process.
- *sp* — Displays all SP circuit cards
- *tr* — Displays all Tip/Ring (T/R) cards.
- *manoos* — Displays all cards in the manual out-of-service state.
- *nonex* — Displays all cards in the nonexistent state.
- *broken* — Displays all cards in the broken state.
- *ins[erv]* — Displays all cards that have at least one channel in the in-service state.
- *t1* —This options displays all T1 cards.
- *sp* — This options displays all SP cards.
- *netoos* — This options displays all cards that have at least one channel in the network out-of-service state.
- *hwoos* — This options displays all cards that have at least one channel in the hardware out-of-service state.
- *foos* — This options displays all cards that have at least one channel in the facility out-of-service state

If more than one option is used, only cards that satisfy all the options given are displayed. If an invalid combination of options is given, an error message is displayed.

Example

The following example displays card information on channel 2 port 0.

disp card 2.0

The following example displays information on all cards.

disp card all

The following example displays information on all cards in the state “Mtc.”

disp card mtc

The following example displays information on all tip/ring (T/R) cards in the state “Broken.”

disp card tr broken

display channel

The **display channel** command displays channel information.

Synopsis

disp[lay] channel *<option> [option]*

disp chan *<option> [option]*

Description

The **display channel** command is used to list information at the channel level. The display channel command options are:

- *number* — Displays information on the channel specified by *chan number.* A range of channels may be specified in the form m–n.
- *all* — Displays information on all channels.
- *mtc* — Displays all channels being used by the maintenance process.
- *telephone <tel number>* — Displays channels with telephone numbers assigned.
- *tr* — Displays all Tip/Ring (T/R) channels.
- *manoos* — Displays all channels in the manual out-of-service state.
- *nonex* — Displays all channels in the nonexistent state.
- *broken* — Displays all channels in the broken state.
- *t1* —This options displays all channels assigned on T1 cards.
- *sp* — This options displays all channels assigned to SP service.
- *netoos* — This options displays all channels assigned to network service.
- *hwoos* — This options displays all channels assigned to hardware service.
- *foos* — This options displays all channels assigned to facility service.

If more than one option is used, only channels that satisfy all the options given are displayed. If an invalid combination of options is given, an error message is displayed.

Example

The following example displays information for channel 1.

disp channel 1

The following example displays information all channels being used by the TSM process.

disp channel tsm

The following example display information on all channels.

disp channel all

display dnis

The **display dnis** command displays the services assigned to DNIS and ANI numbers.

Synopsis

disp dnis

Description

The **display dnis** command is used to display all the services assigned to DNIS and ANI numbers.

Example

The following example displays information for all the services assigned to DNIS and ANI numbers:

disp dnis

display eqpgrp/group

The **display eqpgrp/group** command displays an equipment group report.

Synopsis

disp eqpgrp ***<group number>***

disp group ***<group number>***

Description

The **display eqpgrp** command is used to list all the equipment assigned to the specified equipment group. The *<group number>* is the number of the equipment group. If the group number is missing, a syntax message is displayed. If you specify "all," every equipment group is displayed.

Example

The following example lists all the equipment assigned to equipment group 1.

disp eqpgrp 1

The following example lists all the equipment assigned to equipment groups 2 through 20.

disp group 2-20

The following example lists all equipment assigned to all equipment groups.

disp eqpgrp all

See Also

"assign card/channel"

"delete eqpgrp"

display messages

The **display messages** command displays system (error) messages.

Synopsis

display messages
[priority <alarms, critical, '*C', major, '',minor, '*',events, all>] [-c]**

[start <mm/dd HH:MM:SS>]
[stop <mm/dd HH:MM:SS>]
[card <range,T1,TR,SP,...,all>]
[channel <range,T1,TR,SP,...,all>]
[ID <message ID1,message ID2,all>]
[source <TSM,VROP,SPIP,TRIP,...,all>]
[pattern <regular expression search pattern>]
[number,all]

Description

The **display messages** command displays error and status messages that have been logged by the voice system. Various options are provided so that the display can be limited to specific types of messages. If no arguments are supplied to **display messages**, information is displayed on how to read the messages (the message format) as well as command usage. The messages are written to standard output.

If more messages exist than can be displayed on the screen, you will be prompted with "Press the ENTER key to see more, or enter 'q' to quit." If you do not wish to be prompted to press ENTER (that is, display all of the messages at once), you may use the **-c** option.

The *priority* argument should be used to display messages with specific types of urgencies. Two groups of priorities exist: alarms and events. Alarms are messages that have been reported as *C (critical), ** (major), or * (minor) priorities. Events are all the remaining messages that have no priority (for example, status messages). For example, to display the last 100 alarms, type the following:

display messages priority alarms 100

You can also display specific priorities using the priority option. You can specify either the name of the priority or its symbol (for example, critical or *C) To display all of the critical messages, type the following:

display messages priority critical all

NOTE:
You should use the **priority alarm** argument when alarms are needed, otherwise use the **priority events** argument. The priority argument must be used with this command.

Combinations of priorities can also be displayed by listing each priority separated with a comma. For example, to display the last 100 alarms messages, type the following:

display messages priority '*C','','*' all**

where *C, **, and * must be enclosed in quotes.

Display Message Options

If you wish to display only specific types of messages, you may precede the number of messages to be displayed with one or more of the following options:

- *start*
- *stop*
- *card*
- *channel*
- *id*
- *source*
- *pattern*
- *number*

If more than one of the options is specified, only messages that meet all of the specifications are displayed.

start

The *start* option allows you to specify a starting time for display of messages. Only messages that were logged on or after the time you specify will be displayed. The time can be specified by date and/or a time. The word “today” is equivalent to specifying the current date. Examples of specifying the date are:

- “May 1, 1992”
- “05/01/93”
- “05-01-93”

Examples of specifying the time are:

- hh:mm:ss
- hour=hh
- min=mm
- sec=ss

  where hh is 0 to 23, and mm and ss can be 0 to 59.

DO NOT mix the hh:mm:ss format with the item==xx format. If portions of the time are not specified, the time default is 0 hours, 0 minutes, and 0 seconds.

Also, giving only the time of day indicates the current date. For example, if today is January 15, 1993, the command **display messages start “12/31 09:00”** displays all of the messages that were logged starting at 9 am on December 31, 1993. In order to display messages from a previous year, you *must* specify the year. The entire start date and time must be enclosed in quotes (for example, **display messages start “April 21, 1993 13:00:00”**).

If only the date is specified, the time defaults to the beginning of the day. For example, **display messages start today** displays all of the messages that were logged today (the day in which the command is executed).

stop

The *stop* option allows you to display messages logged up to a specific time. The date and time syntax is the same as that for the *start* option. Therefore, **display messages stop today** displays all messages that were logged before today.

The *start* and *stop* options can be used together to display messages that were logged over a specific period of time. For example, **display messages start "May 1" stop "May 2"** displays all messages logged on May 1 of this year.

If you want the start and stop options to be the same day (for example, May 1), you must specify the hours and minutes for which you want to display messages. Otherwise, the time defaults to 00:00 for both the start and stop options and no messages are displayed.

card

The *card* option allows you to specify messages logged about a specific card or cards. For example, **display messages card 2** displays all messages logged that are associated with card 2. You can display combinations of cards. For example, **display messages card 2,3** displays messages for cards 2 and 3 and **display messages card 0-2** displays messages for cards 0, 1, and 2.

You can also use the *card* option to display messages logged about a specific type of card. For example, **display messages card t1** displays all messages logged about T1 cards.

channel

The *channel* option works like the *card* option. For example, **display messages channel tr** displays all messages logged about Tip/Ring (TR) channels, whereas **display messages channel 5** displays all messages logged about channel 5.

> **NOTE:**
> The *channel* option requires an argument. Typing **display messages channel 100** attempts to display all messages pertaining to channel 100. If you want to display the last 100 messages pertaining to any channel, type **display messages channel all 100**.)

Note that specifying both the *card* option and the *channel* option displays all of the specified card-related messages but, of the channels that are specified, only those that reside on the specified cards are displayed. For example, **display messages card t1 channel all 100** displays the last 100 messages logged for T1 cards and T1 channels, whereas **display messages card t1 channel tr**

never displays no messages because it is impossible for a TR channel to reside on a T1 card.

id

The *id* option allows you to display specific message ids that have been logged. For example, **display messages id TWIP004** displays all occurrences of that message. For example, **display messages id TWIP004,TWIP009** displays all occurrences of both messages.

source

The *source* option allows you to display messages logged by a particular system process. For example, some of the standard system processes are listed in Table 1-3.

Table 1-3. Standard System Processes

Process Name	Function	Types of Messages Reported
ASAI	Adjunct/Switch Application Process	ASAI Problems/Status
MTC	System Maintenance Process	Card/Channel status, Diagnostic Results
SPIP	SP Card Interface Process	Speech,TTS,PRI,SR Problems/Status
TSM	Script interpreter/processor	Script Problems
TRIP	Tip/Ring (Analog) Interface Process	TR Problems/Status
TWIP	T1 Interface Process	T1 Problems/Status
VROP	Speech Database Process	Playback/Coding Database Problems

For example, **display messages source TRIP** displays all messages logged regarding T/R cards and channels.

pattern

The *pattern* option allows you to specify a regular expression as accepted by logCat that may appear in any part of a message. (Refer to the "logCat" command later in this book for additional information.) The *pattern* must enclosed in quotes and surrounded by slashes (/). For example, **display messages pattern '/XYZ/'** provides all messages that use the pattern XYZ anywhere in the message.

NOTE:
The *pattern* option is case-sensitive.

number

The *number* option specifies the number of messages you want to display, or you can use the *all* value to display all messages. The command accepts a three-digit number so you can display up to 999 messages.

NOTE:
Although the *number* option only allows up to 3 digits, you may have more than 999 messages logged. Therefore, you can only view up to 999 messages in the message log report with the display messages command. The **"logCat"** command with the -t option can be used to display all logged messages. Refer to the **"logCat"** command later in this book for more information.

Display Format

All messages are displayed with two or three lines of information. Messages are separated by a blank line to ease viewing. Figure lists the system message formats along with definitions and examples. Each message displayed conforms to the format shown as follows:

PR DAY MON DD HH:MM:SS ZZZ YYYY SOURCE
TTTTTTTT YY UU NUM TEXT...
TEXT (Continuation if necessary.)
blank line

Field	Definition	Examples
PR	Priority	*C (Critical), ** (Major), * (Minor), " "(Event)
DAY	Day	Sun - Sat
MON DD	Date	Jan 1 - Dec 31
HH:MM:SS	Time	00:00:00 - 11:59:59
ZZZ	Time Zone	EST, EDT, CST...
YYYY	Year	1992,...
SOURCE	Source	TSM, TWIP, VROP,...
TTTTTTTT	8 char Msg ID (Tag)	TWIP2104,...
YY	FRU Type	TR, T1, SP, or HO or -- if N/A
UU	Unit Type	CA (Card) or CH (Channel) or -- if N/A
NUM	Unit Number	000 to 999 or --- if N/A
TEXT	Message Text	Varies with message (See example below); can be more than one line long.

Example

The following example is representative of the output from typing **display messages**:

```
                MESSAGE LOG REPORT

Pr  Time            Source
--  ----            ------

** Wed Dec 30 15:55:16 1992TWIP
   TWIP017  T1 CA   0   Facility out of service.
            Reason: Blue alarm

*  Wed Jan  6 13:38:21 1993TRIP
   TRIP002  TR CA   1   Corrupted data detected on TDM bus.
            Timeslot 254. Reason: TDM Parity Error

*  Wed Jan  6 13:41:52 1993TRIP
   TRIP005  TR CH  24   No loop current.
```

display_permissions

The **display_permissions** command displays the current VIS security permissions for a particular user.

Synopsis

display_permissions *<user login>*

Description

The **display_permissions** command displays the current VIS security permissions for a particular user if any has been assigned.

The *<user login>* argument represents the user for which permissions are to be displayed.

Example

The following example executes the command to display VIS security permissions for a specific user.

display_permissions brown

See Also

"unassign_permissions"
"assign_permissions"

display services

The **display services** command lists all valid services or scripts.

Synopsis

display services

disp services

Description

The **display services** command lists all valid services, or scripts, on a system.

Example

The following examples lists all valid services or scripts currently on the system.

disp services

displaypkg

The **displaypkg** command lists the software packages installed on the VIS.

Synopsis

displaypkg

Description

The **displaypkg** command lists the software packages currently installed on the VIS. It is helpful to use this command to see what software is on the system before using either the "installpkg" or "removepkg" commands.

Example

The following example lists all the software packages currently installed on the VIS.

displaypkg

See Also

"installpkg"
"removepkg"

edExplain

The **edExplain** command edits the explanation text for one or more message tags.

Synopsis

edExplain ***{msgID} [...]***

Description

The **edExplain** command edits the explanation text for one or more message tags.

The following are environment variables for the **edExplain** command:

EDITOR	The program used to "edit" the explanation text. Default: vi
EXPLAINDIR	The root directory of the explanation texts. Default: **/gendb/data/explain**
VERBOSITY	If set to anything, **edExplain** will run verbosely.

An explanation file is basically a clear text file. Its contents are displayed "as is" to the user when this explanation is requested. If it is a primary explanation procedure (an explanation that the end user will want to reference by name), it should begin with a line of the form:

<< {tag} [{tag}...] >>

This identifies the explanation or procedure and all its alternate names as defined in the translation file, **$EXPLAINDIR/translateLst**.

The **translateLst** file should be updated to include the msgID, msg string, and file name, in which the explain text can be found (usually just the msgID name). When exiting the **translateLst** file, enter **:w!** followed by **q**.

Two exceptions exist to the rule that the file contains clear text that will be displayed to the user:

1. Any line beginning with a "#" character is considered to be an internal comment and is not displayed
2. Lines beginning with ".explain" are special directives to include at this point another explanation text in place of this line.

Example

In the following example, the first line is the SCCS identification line and is not displayed to the end user. The second line identifies the explanation. Then the text describing the problem follows.

```
#       %W% %T% %H%
<< TWIP007 TWIP_BDERR >>
.... text of explanation describing what a T1 card error means...
```

encode

The **encode** command converts ADPCM16 or ADPCM32 files to PCM64 files.

Synopsis

encode *[adpcm32/adpcm16]*

Description

Encode is a filter that converts PCM64 files to ADPCM16 or ADPCM32 files.

NOTE:
ADPCM16 is easy to code and saves space, but does not provide good quality sound.

Warning

The voice header used by GSE should be stripped (using **stripdhr**) before running encode.

Appropriate code headers must be added (using **"addhdr"**) before the converted file can be played on the VIS.

See Also

"addhdr"
"codetype"
"decode"
"striphdr"

erase

The **erase** command deletes a phrase from a UNIX talkfile.

Synopsis

erase phrase *<phrase number>* from talkfile *<talkfile number>*

Description

The **erase** command deletes the phrases identified by the phrase ID from the UNIX file. The phrase number many be any of the following:

- A single phrase (for example, 1)
- A set of phrases (for example, 1, 2, 5)
- A range of phrases (for example, 1–5)
- All phrases (for example, all)

After you enter the **erase** command, the system displays the following message, asking you to confirm the command before each phrase is erased:

Do you want to erase phrase <phrase#>? (y/n)

If the "all" option is used for phrases, the system prompts you only *once* to confirm the command:

Are you sure you want to erase ALL phrases from talkfile <talkfile#>? (y/n)

If the specified phases does not exist, the system displays:

Phrase <phrase#> does not exist in talkfile <talkfile#>
No action taken.

When the system has deleted the phrase or phrases, the system prompt is displayed.

NOTE:
The **erase** command removes a phrase from the SPEECHDIR default directory, which is **/home2/vfs/talkfiles**.

Example

The following example erases phrase 174 from talkfile 23.

erase phrase 174 from talkfile 23

The following example erases phrases 218 through 222 and phrase 225 from talkfile 26.

erase phrase 218-222, 225 from talkfile 26

The following example erases all phrases from talkfile 29.

erase phrase all from talkfile 29

See Also

"add"
"copy"
"list"

etStub

Th **etStub** process provides the compatibility daemon to log old **et_send**() messages into the new error logging system.

Synopsis

etStub ***[-c] [-r {rule-file}]***

etStub ***-s ~{cmd}~***

Description

The **etStub** process is a daemon process which reads the IPC message queue for error messages that used to be serviced by the Error Tracker (ET) process in Intuity CONVERSANT VIS generics prior to Version 3.1. It takes each error message received and transforms it based on an ASCII rules file into a logging message for the new logging/alerting system. It is provided as a compatibility feature so that executables that worked in previous generics can be supported until the executable can be upgraded to the new logging methods.

The **etStub** process is normally started from **/etc/inittab** with an entry of the following form:

CVet:respawn:4:/vs/bin/vrs/etStub </dev/null >/dev/null 2>&1

Used in this form, **etStub** continuously reads the ET message queue and logs the messages that arrive.

The rules file that specifies the priority, destination, format, and whether the message should be flagged as obsolete for each message is normally **/vs/data/etStub.rules**. An alternate file can be specified with the *-r* flag. This would be most commonly used in conjunction with the *-c* flag, which causes **etStub** to just read the rules files, check it for syntactic correctness and immediately exit. It is recommended that any time the rules file is altered, that **etStub -c -r *{file}*** be used to insure that the file is correct before installing it.

The **etStub** process can be requested to reread its rules file by using the *-s* option. By executing **etStub** as a command with the *-s* option and *{cmd}* set to *rules*, a message is sent to the **etStub** causing it to discard its current rules and reread the rules file. This process can also be used to request that an alternate rules file be used or that etStub exit. Refer to the following examples:

```
etStub -s rules          # reread current rules file
etStub -s "rules {file}" # read new rules from {file}. Quotes required.
etStub -s exit           # request the etStub daemon to exit & respawn
```

NOTE:
Notice that the quotes are required in the second example since there is a space between the command word rules and the file name.

Rules File

The format of the rules file is as follows:

```
#     Comments begin with the '#' character

{msg-#} {flag} {priority} {dst} ".....format of message....."
...
```

The following provides an explanation of the fields in the rules files:

{msg-#}	The number that identifies this message. It is defined by a header file found in **/att/msgipc/etmsgs**.
{flag}	This may be either 0, meaning the message is to be considered OBSOLETE and noted as such when it is logged, or -, meaning that the message will not be flagged obsolete when received and logged.
{priority}	This takes one of the following values: -, *, **, or ***C**. These correspond to NONE, MINOR, MAJOR, and CRITICAL.
{dst}	This is the destination for the message as used to be specified in the errors files. Legal values are **LOG** and **PRT**. Messages with a destination of **LOG** are sent to the MASTER_LOG files and to either the EVENT_LOG files or the ALARM_LOG files based upon the priority of the message. Those with priority NONE go to the EVENT_LOG destination in addition to the MASTER_LOG destination. All non-zero priority messages go to the ALARM_LOG destination in addition to the MASTER_LOG destination. Messages with destination of **PRT** go to the same destinations as those marked **LOG**. They also go to the **stdout** of the **"etStub"** process itself. If this is directed to **/dev/null** as shown in the **/etc/inittab** example, then it has no visible effect. If the **stdout** is directed to a device, such a **/dev/console**, then these messages show up at this device as well.
format	The format of the message is the same as the one that appeared in the original **/vs/data/errors** or **/gendb/data/errors** file.

Files

/vs/data/etStub.rules

Exit Values

The **"etStub"** process consists of a whole series of well defined exit values that can help in determining why it has exited. The exit values come in two forms, those used when it is running as a daemon process and those used when it is running as a command that sends messages to the daemon process. The exit values when it is running as a daemon process appear in the DEAD_PROCESS entries listed by **init** in the **/etc/utmp** and **/etc/wtmp** files. If the **etStub** daemon process is not staying up, examining the output of **who -du /etc/wtmp |grep Cvet** can provide useful information about what the problem is in addition to what is being logged to the error logging system, assuming that the **logdaemon** process is up and properly running.

Table 1-4. Exit Values of the etStub Daemon Process

1 = X_MSGRCV_FAILED	An attempt to receive a message failed. Has the message queue been removed or does it have unexpected modes?
2 = X_NO_MSGQ	The process was unable to attach a message queue. Is there more than one **"etStub"** daemon process?
3 = X_QKEY_BAD	Bulletin board routines are reporting that the message queue key is bad.
4 = X_MSG_TOOBIG	The message received was too big and had to be truncated. This is not an exit condition at this time.
5 = X_NO_DEVTBL	The device table shared memory does not exits. Was an attempt made to run **"etStub"** at run level when the VIS system was not active? Or has the shared memory been removed?
6 = X_NO_BB	The bulletin board shared memory does not exist. Was an attempt made to run **"etStub"** at a run level when the voice system was not active? Or has the shared memory been removed?
7 = X_LOGINIT_FAILED	An attempt to contact the **logdaemon** process failed. Is **"etStub"** running a level when the logdaemon is not running?

The following exit values apply when the *-s* option is being used to send commands to the **"etStub"** daemon process.

Table 1-5. Exit Values of the etStub Command Process

8 = X_NO_CMD	No command was provided.
9 = X_UNRECOGNIZED_CMD	The command was not recognized.
9 = X_AMBIGUOUS_CMD	The command was ambiguous.
10 = X_WRONG_ARGS	The number of arguments provided with the command are wrong.
11 = X_TOO_LONG	The command string was too long for the message buffer and could not be transmitted to the **"etStub"** daemon process.
12 = X_MSGSND_FAILED	The **msgsnd** () system call failed.

See Also

"mkETrules"

explain

The **explain** command displays on-line error message explanations.

Synopsis

explain *{msgID} [...]*

explain *-l {pattern} [...]*

explain *-d {msgID} [...]*

Description

The **explain** command displays on-line error message explanations. The *{msgID}* is one of the two forms of identification that comes with each message. The primary form is ***{CLASS}nnn***, where *{CLASS}* is the class of messages, such as CGEN, TSM, etc., and *nnn* is the index of the message within the class of messages. The second form, available with most messages is the mnemonic form (for example, CGEN_NOMSGQ or CGEN_MSGRCV).

If the explanation of the message fits in 24 lines and only a single explanation has been requested, it is printed without interruption. If the explanation is longer than 24 lines or more than one explanation is requested, the output is *paged* via the use of a paging program. Use the *-d* option to disable paging. The default paging program is **/bin/pg**.

If the *-l* option is used, **explain** looks up all messages whose *{msgID}* matches the pattern. For example, **explain -l A V** lists the names of explanations available that begin with either “A” or “V,” while **explain -l VROP** lists all explanation names available that begin with **VROP**. In other words, the *{pattern}* is anchored at the beginning of the *{msgID}* and assumes a match of anything after the pattern selected.

Variables for Advanced Users

The **explain** command is also affected by certain environment variables. These environment variables are intended for advanced users only.

PAGER

The pager program used if the explanation is longer than 24 lines or more than one explanation is requested. The default is **pg**. If you do not want paging even for long explanations, using *-d* or setting **PAGER=cat** will disable paging. A one line form would be:

PAGER=cat explain {msgID} or **explain -d {msgID}**

EXPLAINDIR

The directory in which the explanation directories are found. The default is **${PRODUCTROOT}/gendb/data/explain**.

PRODUCTROOT

This is the installation directory and defaults to / (root).

VERBOSITY

This is a debugging aid. Setting it to anything causes debugging output to be generated while **explain** performs its job.

The **"edExplain"** command allows you to add or change explanations. An explanation comes in two parts, a file containing the explanation itself, and a set of synonyms or translations that allow the **explain** command to find the file under more than one tag. To create a new explanation, you must provide both. When modifying an existing explanation, all you need to do is edit the file containing the explanation.

The explanation file itself is almost a clear text file of what you want the user to see when they ask for the explanation. There are two features of the file that are not plain clear text. All lines beginning with the "#" character are treated as internal comments and are not output. Also lines of the form **.explain {msgID}** have special meaning. They cause the inclusion of the explanation text specified by the *{msgID}*. This allows you to have common explanations and reference from more than one explanation.

The recommended format for an explanation procedure is:

 # Comment and SCCS keywords

<< {msgID} [{msgID}...] >>
{text of message}
...

When creating a new explanation procedure, you will be asked to edit the synonyms list and be placed in the appropriate **translateLst** file. There are instructions at the top of the file. Each non-comment line is a list of synonyms, with the right most word on the line being the name of the file in which the text is located. For example:

ADM001ADM_SYSERR
ALERT003AL_INVALID_THRESHOLDAL_INVALID_T

The description for ADM001 and ADM_SYSERR are found in a file named ADM_SYSERR. The description for ALERT003 and AL_INVALID_THRESHOLD are found in a file named **AL_INVALID_T**. The second example has a truncated file name, because file names are limited to 14 characters in most UNIX systems and if you want to use source code control, then the file name must not be longer than 12 characters. The recommended way to store an explanation is under a file name related to the mnemonic *{msgID}* rather than the {*CLASS}nnn* name, since the later is meaningless. A file name of the form *{CLASS}nnn* does not provide a sophisticated user with much information about the contents of the file, while the mnemonic form does. If the mnemonic is longer than 12 characters, then you should create a shorter name related to the mnemonic that is unique within 12 characters.

There are some environment variables that affect the behavior of **"edExplain"**:

EDITOR

This is the name of your preferred text editor. The default is vi.

EXPLAINDIR

This is the directory in which the explanation directories are found. The default is **${PRODUCTROOT}/gendb/data/explain**.

PRODUCTROOT

This is the installation directory and defaults to /.

VERBOSITY

This is a debugging aid. Setting it to anything cause debugging output to be generated while **"edExplain"** performs its job.

Files

/gendb/data/explain # directory in which explanation directories are located.

/gendb/data/explain/translateLst # file containing the synonym list of {msgID}s."

See Also

"edExplain"

findHomes

The **findHomes** command populates your home directory with user files saved as part of assisted upgrade.

Synopsis

findHomes ***[-?] [-v] [-D <dir>]***

Description

The **findHomes** command is part of the Intuity CONVERSANT VIS Version 5.0 Upgrade Assistance Package. It provides a convenient way to restore your files from the location where they are saved by the Upgrade Assistance Package to the home directory for each user defined in the **/etc/passwd** directory, if the user has the same login ID as they had on the pre-upgrade system.

For each user, the entire directory structure (including all files) preserved from the user home directory on the pre-upgrade machine is moved to the user home directory on the upgraded system. If a saved file has the same name as a file which already exists in a user's home directory on the upgraded system, the saved version is moved to **o.<filename>** in that directory.

Files for any users whose login ID changes from the pre-upgrade system to the upgraded system must be manually moved from their saved location to their new home directory. This manual intervention will also be required for users who did not use their login ID as the name of their home directory on the pre-upgrade system.

The **findHomes** command should be run after the assisted software upgrade has completed and logins for all users expected to move from the pre-upgrade system to the upgraded system have been administered on the upgraded system. See *Intuity CONVERSANT VIS Version 5.0 Operations*, 585-310-550, for more information about administration of user logins.

The *[?]* argument displays a help message.

The *[-v]* argument causes a list of saved files to be printed as they are moved.

The *-D <dir>* argument specifies an additional directory to be searched for saved user files.

The Upgrade Assistance Package saves user files in **/home/o.<homedir>**, where *<homedir>* is the last directory in the full-path-name home directory specified for each user in the **/etc/passwd** file on the pre-upgraded system. Often, *<homedir>* is the user's login ID. If the file restoration to this directory fails, then the files are restored in a directory with the full pathname of the user's home directory on the pre-upgraded system.

NOTE:
A message is printed indicating any directory for which all the files are not successfully relocated.

Example

The following example causes all files found in **/home/o.*<homedir>*** and **/usr/*<homedir>*** to be moved to the home directory specified for each non-system user specified in the **/etc/passwd** file on the upgraded system:

findHomes -v -D/usr

fixLogFile

The **fixLogFile** command upgrades existing logging files after **"lComp"** is run so that data continues to be readable by **"logCat"**.

Synopsis

fixLogFile ***[-d] [-s {save-file}] [-r] [-a] [-S] [-o {spec}] [-n {spec}] file1 [file2...]***

Description

When classes of logging messages are expanded, contracted, inserted, or removed, **fixLogFile** can change the index assignments of messages. When this happens, messages whose indexes changed and were logged under the previous environment become unexpandable by **"logCat"**. The **fixLogFile** command, given information about the previous assignments and the new assignments, upgrades logged data so that it remains expandable by **"logCat"**.

Each message is examined. If the class of messages appears in the new environment and still covers the index assigned to the message, a new index is assigned based on where it appears in the new environment. If the class of messages is no longer part of the message logging environment or if a class is reduced in size so that it no longer covers the index of a message, then it is necessary to do one of three things:

- *-d* — Deletes the message entirely from the logging file.
- *-r* — Demaps the message. This entails expanding the message in the old environment and then creating a new logging message using the LOG_REMAP_DISCARD format so that the data is still readable in the log files, but is marked as being part of a discarded message environment. This is the default behavior.
- *-s {save_file}* — Removes the message from the original logging file and saves it in the specified file, thus preserving the unique data for possible later retrieval.

Normally, **fixLogFile** generates a short message about each file that it converts. The *-S* flag suppresses this output.

The **fixLogFile** command requires access to the **old o.systemLog.h** and **o.textLogFmt** files and the new **systemLog.h** file to perform its job. It expects to find these files in **$LOGROOT/formats**. If alternate sources of these files are to be used, the *-o* and *-n* flags are used. Each of these flags takes a *{spec}* argument, which has the following form:

{dir}[,{systemLog.h}}[,{textLogFmt}]]

The default values for these two specifications is:

-o ${LOGROOT}/formats,o.systemLog.h,o.textLogFmt
-n ${LOGROOT}/formats,systemLog.h,textLogFmt

The *{dir}* portion specifies an alternate directory in which the ***[o.]*systemLog.h** and ***[o.]*textLogFmt** files are to appear. If the remainder of the *{spec}* is missing, the default file names apply. If specified, the ***{systemLog.h}*** and ***{textLogFmt}*** portions specify the names of these two files as they appear in the specified *{dir}*. Any section of the specification that is skipped retains its previous or default value.

A list of one or more logging files may be specified. If they are listed, each one is assumed to be a compressed logging file and is converted. The *-a* option automatically converts all of the compressed logging files found in ***${LOGROOT}*/data**. No file names can be provided if the *-a* option is specified. When the *-a* option is used, each regular file found in ***${LOGROOT}*/data** is examined to see if it is a compressed logging file. If it is not, it is ignored. If it is, it is converted.

After the files are converted, the time stamps are reapplied so they have the same date after conversion as they did before the conversion.

Caveats

The **fixLogFile** command only takes care of changes in classes of logging message. For example, if the class PERM was added, removed, or moved, **fixLogFile** could correctly deal with the changes to the logging files. The **fixLogFile** command does not deal with reorganizations or changes of messages within a class. Do *not* change the order of appearance messages or the arguments to a logging message if you expect to be able to expand the data in the future or save the previous **textLogFmt** file for the expansions.

If the conversion takes place while the **logdaemon** process is running, be sure to either stop and restart **logdaemon** or reinit it using the **"reinitLog"** command.

See Also

"logCat"
logdaemon

get_config

The **get_config** command gets the **/vs/data/conf_data** file from floppy disk.

Synopsis

/vs/bin/util/get_config

Description

The **get_config** command is used to retrieve from the CONFIGURATION DATA floppy for a particular machine the **/vs/data/conf_data** file. That file represents the configuration of the VIS machine as it was shipped from the factory or as determined by the **/vs/bin/util/configure** program after the last upgrade was performed on the machine.

The **get_config** command should be used as the first step in upgrading an existing VIS machine in the field.

Files

/vs/data/conf_data

See Also

"add_device"
"change_device"
"configure"
"remove_device"
"save_config"
"show_config"
"show_devices"

gse

The **gse** command invokes the Graphical Speech Editor (GSE) used for speech files.

Synopsis

gse ***[-l<chan#>] [-p<playchan#>] [-r<recchan#>]***

Description

The **gse** command invokes the GSE for speech files. Speech files are opened using a standard Motif interface, and various editing tasks such as cutting, pasting, and changing speech volume can be performed. Recording and playback is provided through the tipring card as well.

NOTE:
The Voice Information System (VIS) must be stopped while using the GSE. Entering the **gse** command results in a system prompt asking you if it is okay to stop the VIS.

The [-l chan#] argument is used to specify a single channel to use for speech playback and recording. If this argument is not specified, and if the -p and -r arguments are not specified, then the channel number defaults to 0.

The [-p playchan#] argument is used to specify which channel to use for speech playback.

The [-r recchan#] argument is used to specify which channel to use for speech recording.

The [-D directory] argument is used to specify that the program will start in the given directory.

The [-O] option is for debugging purposes: it causes certain tipring events to be displayed in an output window.

The [-I] option inhibits automatic resetting of the tipring circuit cards.

The designated channels must be configured for both input and output; this will facilitate recording and playing back speech phrases. The *-l* channel is used for telephone lines and incorporates both audio input and output. Alternatively, the *-p* and *-r* channels must be used together to establish the audio input/output, where *-r* (input) is designated for the microphone, and *-p* (output) for the speaker.

WARNING:
Channel numbers for the gse are hardware channel numbers, not VIS channel numbers. The method for computing hardware channel numbers is given in Appendix C, "Computing Channel Numbers," of Intuity CONVERSANT VIS Version 5.0 Command Reference.

See Also

"gse_add"
"gse_addpl"
"gse_copy"
"gse_copypl"

gse_add

The **"gse_add"** command transfers a speech phrase from a UNIX file to a UNIX file in the Graphical Speech Editor (GSE) format.

Synopsis

gse_add *<talkfile number> <phrase number> <codestyle> <input file>*

Description

The **gse_add** command transfers a speech phrase from a UNIX file to a talkfile in a GSE format. Use this command when a phrase that does not belong to a speech pool needs to be added to a talkfile.

The *<talkfile number>* and *<phrase number>* parameters refer to the talkfile and phrase identifiers in the speech file system. The talkfile and phrase numbers of the phrases to be added must be known. Use the list command when this information is not known. The *<codestyle>* parameter can be pcm64, adpcm32, or adpcm16. The *<input file>* parameter is the file from which the phrase is to be taken.

Example

The following example adds the phrase 100 to talkfile 103 using ADPCM32 format from the file **/usr/speech/103/1000**.

gse_add 103 100 adpcm32 /usr/speech/103/1000

See Also

"gse_addpl"
"gse_copy"
"gse_copypl"

gse_addpl

The **"gse_addpl"** command adds (restores) phrases to a specific speech pool from UNIX files in the Graphical Speech Editor (GSE) format.

Synopsis

gse_addpl *<speech pool> <input directory> <codestyle> [<file1>...<fileN>]*

Description

The **gse_addpl** command reads the phrase list file in the speech pool (/speech/talk/*<speech pool>*.pl) to determine the talkfile, phrase numbers, and file names of the phrases to be added. Use the Shared Speech Pools screen for the Script Builder application to determine which speech pools are being used by the application.

The *<speech pool>* parameter is the name of the speech pool to which the speech is to be added. The *<input directory>* parameter is the name of the directory where the GSE edited files are located. The *<codestyle>* is either pcm64, adpcm16, or adpcm32. The *<file1>* through *<fileN>* parameters are the optional file names identifying the phrase names to be added. If no file names are specified on the command line, all phrases in the speech pool for which file are found in *<input directory>* are added. If file names are given, only the phrases with the particular file names that are specified in the phrase list file are added.

An unrecorded phrase is marked with a negative number in the phrase list file. If gse_addpl is used to add a previously unrecorded phrase, the phrase number is changed to a positive value to indicate the phrase exists. The applications using that phrase list must then be verified and installed with the specific speech pool.

Example

The following example adds phrases 1000, 1001, and 1002 to talkfile 103 using ADPCM32 format from files **f1000**, **f1001**, and **f1002** in the directory **/speech/talk/talk3.files**

gse_addpl talk3 /speech/talk/talk3.files adpcm32

See Also

"gse"
"gse_add"
"gse_copy"
"gse_copypl"

gse_copy

The **gse_copy** command extracts a speech phrase from the VIS speech file system to a UNIX file in the Graphical Speech Editor (GSE) format.

Synopsis

gse_copy *<talkfile number> <phrase number> <output file> [“<tag>”]*

Description

The **gse_copy** command extracts a speech phrase from the VIS speech file system to a UNIX file. This command may be used when a phrase that does not belong to a speech pool needs to be edited.

The *<talkfile number>* and *<phrase number>* parameters refer to the talkfile and phrase identifiers in the speech file system. The talkfile and phrase numbers of the phrases to be added must be known. Use the list command when this information is not known. The *<output file>* parameter is the file where the speech phrase is to be placed. The “*<tag>*” parameter is an optional 50 character string that is placed into the GSE voice header of the output file. The GSE displays the tag value when the file is being edited.

NOTE:
You must keep record of which extracted files are associated with what talkfile and phrase in order to return the speech to its proper place after editing. We recommend that the *<output file>* be the same as the *<phrase number>* and the directory containing the *<output file>* be the same as the *<talkfile number>*. See the example below.

Example

The following example extracts phrase 1000 from talkfile 103 and places it in the file **/usr/speech/103/1000** for editing by the GSE.

gse_copy 103 1000 /usr/speech/103/1000

See Also

"gse"
"gse_add"
"gse_addpl"
"gse_copypl"

gse_copypl

The **gse_copypl** command allows multiple speech phrases to be copied from the speech file system in the Graphical Speech Editor (GSE) format.

Synopsis

gse_copypl *<speech pool> <output directory> [<file1>...<fileN>]*

Description

The **gse_copypl** command allows you to copy multiple speech phrases from the speech file system. It reads the phrase list file belonging to the speech pool (**/speech/talk/*<speech pool>*.pl**) to determine the talkfile, phrase numbers, and output file names for the phrases to be extracted. Use the Shared Speech Pools screen for the Script Builder application to determine which speech pools are being used by the application.

The *<speech pool>* parameter is the name of the speech pool from which the speech is to be retrieved. The *<output directory>* parameter is the name of the directory where the output files are to be placed. The *<file1>* through *<fileN>* parameters are the optional file names identifying the particular phrase names to be extracted. If no output file names are specified on the command line, all phrases in the speech pool are extracted. If file names are given, only phrases with those file names specified in the phrase list file are extracted.

Example

The following example extracts phrases 1000, 1001, and 1002 from talkfile 103 and places them in files f1000, f1001, and f1002 (respectively) in the directory **/speech/talk/talk3.files**. These files are then ready to be editing using the GSE.

gse_copypl talk3 /speech/talk/talk3.files

See Also

"gse_add"
"gse_addpl"
"gse_copy"

hassign

The **hassign** command assigns host service to host sessions.

Synopsis

hassign *[hostsvc] <host_application> to <session #> [FTSCRT]*

Description

The **hassign** command assigns applications to session numbers. It is necessary to use this command to associate an application with host interactions to a given logical unit (LU). One host session corresponds to one LU.

The *FTSCRT* argument is required to assign that session for file transfer. If you are using file transfer, valid session numbers are 0–31 (that is, only sessions on the first host communication card can be used for file transfer).

The **hassign** command automatically executes the host maintenance "login" sequence on the specified session.

If the application "appl" is currently assigned to a particular session, and a new application "new_app" is assigned to that same session, the old application "appl" is logged out and "new_app" is logged in. In other words, the application currently assigned to the session is replaced by the new application you wish to assign.

If the application "my_appl" is assigned to session 0 and you wish to assign the application "your_appl" to session 0, you must:

1. **"hlogout"** the application "my_appl"
2. **"hfree"** the application "my_appl"
3. **"hlogin"** the application "your_appl"

If the application "my_appl" is already assigned to session 0 but the state is logged out and **hassign** is run on "my_appl" again, nothing happens. You need to run **"hlogin"** to log the application "my_appl" back in.

Example

In the following examples, the user is assigning the host application “my_appl” to session 0, to session 0–19, and to all sessions (up to 64 total with 2 host communications cards installed), respectively.

hassign my_appl to 0
hassign my_appl to 0-19
hassign my_appl to all

In the following example, the user is assigned in the host application “file_trans” to session 0 for file transfer.

hassign file_trans to 0 FTSCRT

See Also

"hdelete"

hconfig

The **hconfig** command configures host interface parameters.

NOTE:
To configure host interface parameters, it is recommended that you perform the procedure discussed in Chapter 3, "Configuration Management " of *Intuity CONVERSANT VIS Version 5.0 Operations*, 585-310-550.

Synopsis

hconfig ***[-t {SDLC | TR}] [connection name]***

hconfig ***-c [-t {SDLC | TR}]***

hconfig ***-l [-t {SDLC | TR}]***

hconfig ***-w <field name_1>=<value_1>...<fieldname_n>=<value_n><connection name***

hconfig ***-d <connection name>***

hconfig ***-r <old name> <new name>***

hconfig ***-n***

hconfig ***-s***

hconfig ***-b***

hconfig ***-a <connection name>***

hconfig ***-m***

Description

The **hconfig** command is the command interface used to read and write LINKix configuration files.

When used with no options, the **hconfig** command will output the current values for the connection parameters for the specified connection, or all connections, if no connection is specified. If the -t option is specified, only connections of the specified type, SDLC or Token Ring, are displayed. If a connection is an SDLC connection, the following fields are displayed: connection name, card number, line type, node id to send, encoding, constant carrier, poll address, and LU. If a connection is a Token Ring connection, the following fields are displayed:

connection name, adapter, local sap, remote sap, node id to send, remote address, and LU.

Variable	Definition
-c	Used to generate a list of defined connection names. If the -t option is specified, only the names of connections of the specified type, SDLC or Token Ring, are displayed.
-l	Used to generate a list of defined LU's and their associated connection names. If the -t option is specified, only the LUs of the specified type, SDLC or Token Ring, are displayed.
-w	Writes the value <*value*> to the fieldname <*fieldname*> parameter of the connection <*connection name*>. The following fields are supported for SDLC connections: name, card_number, line_type, node_send, encoding, const_carrier, poll_address, LU.
-d	Used to delete a connection.
-r	Used to rename a connection from <*old name*> to <*new name*>
-n	Used to renumber host sessions. The voice system must be stopped before renumbering is done.
-s	Used to synchronize the /vs/data/hostsvc file with the current configuration. The voice system must be stopped before resynchronization can be done.
-b	Used to generate a list of the SDLC cards in the system. The output shows the card number, whether a connection has been configured for that card, the card type (PC/XL or FIFO/SIB), the IRQ, the I/O address, and the RAM address (for PC/XL cards only).
-a	Used to specify the connection over which NetView Alerts are sent.
-m	Used to determine the Token Ring MAC Address of the voice system.

NOTE:
The voice system must be restarted before any configuration changes take effect.

Example

In the following examples, the user is unassigning the host application "my_appl" from session 0, from session 0–19, and from all sessions (up to 64 total with 2 host communications cards installed), respectively.

hdelete my_appl from 0
hdelete my_appl from 0-19
hdelete my_appl from all

See Also

"hassign"
"hfree"

hdelete

The **hdelete** command removes host service from host sessions.

Synopsis

hdelete ***[hostsvc] <host_application> from <session number or range or all>***

Description

The **hdelete** command executes the logout sequence that is defined in the application's host session maintenance section and automatically removes the host application association from the session number. One host session corresponds to one logical unit (LU).

NOTE:
The **"hfree"** command works similarly to the **hdelete** command except that does not execute the logout sequence and should be used only when you need to release the session immediately.

Example

In the following examples, the user is unassigning the host application "my_appl" from session 0, from session 0–19, and from all sessions (up to 64 total with 2 host communications cards installed), respectively.

hdelete my_appl from 0
hdelete my_appl from 0-19
hdelete my_appl from all

See Also

"hassign"
"hfree"

hdiagnose

The **hdiagnose** command diagnoses the SDLC communication card.

Synopsis

hdiagnose conn *<connection name>*

hdiagnose info *<connection name>*

Description

The **hdiagnose** command runs card level diagnostics on the SDLC card associated with the connection <*connection name*>.

The **hdiagnose info** command attempts to determine actual values for host protocol parameters by watching link traffic.

The <*connection name*> variable should be a valid name of a host connection or you can specify the connection name as **all**.

NOTE:
The voice system must be down for hdiagnose to run. All host connections, including connections not being diagnosed, will be disrupted during command execution.

Example

The following example displays the host application currently verified and installed on the system as well as the current session assignments.

"hdisplay"

hdisplay

The **hdisplay** command shows host applications that have been successfully verified and installed, as well as the application assignments on the host sessions.

Synopsis

hdisplay ***[hostsvc]***

Description

The **hdisplay** command displays the host applications that have been verified and installed on the system. The **hdisplay** command also displays the current session assignments for each host application.

Example

The following example displays the host application currently verified and installed on the system, as well as the current session assignments:

hdisplay

headFIX

The **headFIX** command converts user application Error Tracker (ET) message rules header files used in generics prior to VIS Version 3.1 (V3.1) to the header files required for the VIS V3.1 logger/alerter environment.

Synopsis

headFIX ***[-y {year}] [-c{company}] [{class1}_et.h] [{class2}_et.h...]***

Description

"Full" conversion from the ET environment associated with VIS generics prior to V3.1 to that used by V3.1 requires that the application source be modified or converted and then recompiled. Additionally, several files used in conjunction with the ET environment must be converted to those used by the V3.1 logger/alerter.

The **headFIX** converts the ET error rule header files to a set of header files used by the V3.1 alerter/logger. The ET rules header file reserved for applications (generics prior to V3.1) is **/att/msgipc/etmsgs/appl_et.h**; however, any properly formatted rules header file may be converted. The **headFIX** command converts the ET header file name *{class}_et.h* into a corresponding file ***{CLASS}*.h** which is used by the V3.1 alerter/logger. File names input into the converter must adhere to this naming convention. Following conversion, the **log*{CLASS}*.h** header file resides in your working directory. This file must be moved to the directory **/usr/spool/log/head** where i5 is used by the processes which use the alerter/logger.

If no ET header files are supplied as command arguments, a help message is displayed explaining the usage of the command and the target locations of the output. If the file argument is not found, is not named properly, or is not formatted in accordance with ET rules files, the appropriate error and help messages are displayed.

Each **log*{CLASS}*.h** file created by **headFIX** has a copyright header. By default it is for the current year and is assigned to AT&T. A alternate year can be specified using the *-y* flag and the company name can be altered using the *-c* flag.

Example

The following example converts ET headers {class1}_et.h {class2}_et.h etc.

headFIX class1_et.h class2_et.h...

See Also

"mkMsg"

hfree

The **hfree** command unconditionally releases host sessions from Script Builder host application assignments.

Synopsis

hfree *<host_application> or <session number or range or all>*

Description

The **hfree** command releases sessions from their Script Builder application assignments. One host session corresponds to one logical unit (LU). It frees the assignment and leaves the host session on the screen it was currently. This command can be helpful in resolving a problem with a particular screen. Normally, the **"hdelete"** command should be used to make a session available. The **hfree** command is used commonly when problems occur on sessions and troubleshooting is needed.

NOTE:
The **hfree** command does not automatically log out the specified session. Use the **"hdelete"** command to log out a session and make the specified session available.

See Also

"hdelete"

hlogin

The **hlogin** command runs the login sequence of a host script.

Synopsis

hlogin ***<host_application> or <session number or range or all>***

Description

The **hlogin** command invokes the login procedure that is defined in the application's host session maintenance section. This command is often used in the system's cron table to automatically log in the host as soon as it is available each day.

NOTE:
The session must be in the logged out state before you may execute this command.

Example

The following example invokes the login procedures for the host application "my_appl."

hlogin my_appl

The following example invokes the login procedures for the host applications assigned to sessions 0 through 9.

hlogin 0-9

See Also

"hassign"
"hlogout"

hlogout

The **hlogout** command runs the log out sequence of the host script.

Synopsis

hlogout *<host_application> or <session number or range or all>*

Description

The **hlogout** command invokes the logout procedure that is defined in the application's host session maintenance section. This command is often used in the system's cron table in order to log off the host automatically before it goes down each night. It is a clean, convenient way to log out of the host application.

NOTE:
The session must be in the logged in state before you may execute this command.

This command should be performed once an application has been developed and hassigned to a session to test the logout procedure.

Example

The following example invokes the log out procedure for the host application "my_appl."

hlogout my_appl

The following example invokes the log out procedure for all session numbers.

hlogout all

See Also

"hassign"
"hlogin"

hnewscript

The **hnewscript** command installs a changed host script.

Synopsis

hnewscript *<host application>*

Description

The **hnewscript** command updates the system memory with the newest copy of the specified host application. This can cause the host application to be logged out and then logged back in using the newly defined host maintenance. This is required to put the updated version into effect, and Script Builder automatically prompts you when the verify and install have been composed and the host maintenance has changed.

Example

The following example updates the system memory with the most current copy of the host application “my_appl.”

hnewscript my_appl

WARNING:
*The **hnewscript** command may temporarily prevent access to any host sessions which have been modified while they are in the process of logging out and logging back in.*

host_cfg

The **host_cfg** command translates an ASCII configuration file into a properly formatted binary configuration file.

Synopsis

host_cfg *<ASCII_config_file> <binary_config_file>*

Description

The **host_cfg** command translates the data from the specified ASCII configuration file into the format needed by the 3270 host card. This command then writes the data to the specified binary configuration file.

The *<ASCII_config_file>* argument is the name of the ASCII configuration file. This file is usually in **/usr/lib/3270/host.cfg0** or **/usr/lib/3270/host.cfg1** for card 0 and 1, respectively. These ASCII files contain configuration data for 3270 host cards 0 and 1.

The *<binary_config_file>* argument is the file name where the compiled ASCII configuration data is stored. Once the binary configuration file is created, it can be used via the **"load_bin"** command to initialize the card.

A series of keyword name = keyword value(s) entries appear in the ASCII configuration file with only one “name=value(s)” entry per line. The keyword name must start at column one on each line. An equal sign (=) should separate the keyword name from the keyword value(s). All letters in all keyword names should be upper case. A list of keyword values for a given keyword name must be separated by commas. All white spaces (spaces and tabs) are ignored. Refer to *Intuity CONVERSANT VIS Version 5.0 Communications Development*, 585-310-229, for additional information about configuration parameters.

Files

/usr/lib/3270/host.cfg0 **/usr/lib/3270/host.cfg1**	The ASCII configuration files
/usr/lib/3270/host.bin0 **/usr/lib/3270/host.bin1**	The binary configuration files

Example

The following example shows the contents of an ASCII configuration file for a card name “host3270a” using SNA protocol with 6 logical units (terminals), half duplex operation, cluster controller address of 193, with an XID of 0176c8c4 hex, and NRZ encoding.

```
SYNCPORT=host3270a
EXEC_TYPE=SNA
CRT24_80=2, 32, 64, 108, 187, 254
SDLC_ADDR=193
XID=0176c8c4
LINE_MODE=HALF
NRZ_CODE=NRZ
```

See Also

"load_bin"

hsend

The **hsend** command sends a file to the host via CVIS_FTS.

Synopsis

hsend file=*<cs_file> [dest=file_destination] [opt=option_list|n]*

Description

The **hsend** command is used to send a file to the host via cvis_fts. The arguments for the **hsend** command are:

- The *file* parameter is mandatory argument for the **hsend** command. The *<cs_file>* parameter is the full path name of the UNIX system file to be sent to the host. Refer to *Intuity CONVERSANT VIS Version 5.0 Communications Development*, 585-310-229, for file name guidelines for file transfer.
- The *dest* parameter is an optional argument, where *<file_destination>* is the final destination of the file at the host. If this parameter is not specified, the DESTINATION parameter value in the file **/vs/data/fts_config** is used.
- The *opt* parameter is an optional argument, where *option_list|n* is the list of option parameters or the letter *n* (for no options). Options must be separated by a space. Refer to *Intuity CONVERSANT VIS Version 5.0 Communications Development*, 585-310-229, for a detailed options list. If *opt=n*, the PARAM1, PARAM2, and PARAM3 values in the **/vs/data/fts_config** file are not used. If this argument is missing (the default), the PARAM1, PARAM2, and PARAM3 values in the **/vs/data/fts_config** file are used.

Return Values

Refer to *Intuity CONVERSANT VIS Version 5.0 Communications Development*, 585-310-229, for information on CLEO file transfer return codes.

hspy

The **hspy** command displays a screen currently present on a specified host session.

Synopsis

hspy ***<session number or range or all>***

Description

The **hspy** command shows what screen currently is being presented on that specified session, a range, or all. One host session corresponds to one logical unit (LU). This information helps to isolate what screens might be involved in a problem, should one occur. This tool can help to resolve problems, but should not be the only source of problem isolation.

Example

The following example displays the screen currently be presented on session 0.

hspy 0

Output

A screen of data representing what is currently present on a host session.

WARNING:
This screen presents what the process that communicates with the host believes is present, but it may not be the actual screen present on that host session.

hstatus

The **hstatus** command shows the current status of the host sessions.

Synopsis

hstatus ***<host application> or <session_number or range or all>***

Description

The **hstatus** command reports the current status of the host application assigned to the associated host sessions. One host session corresponds to one logical unit (LU). The possible status states are as follows:

- Logging in — This is a temporary state indicating the session is in the process of logging in immediately after a manual **"hassign"** or **"hlogin"**.
- Logged in — This state occurs after a successful login. The session is ready to accept a transaction (the transaction base screen is reached).
- Logging out — This is a temporary state indicating the session is in the process of logging out immediately after a manual **"hlogout"**.
- Logged out — This state indicates that service is still assigned, but the session has been logged out.
- Recovering — This state occurs if the login procedure fails, the transaction ends somewhere other than the transaction base screen, or the recovery procedure ends somewhere other than the transaction base screen.
- Unassigned — This state indicates that service was never assigned to the session or service was assigned and later manually deleted.
- Not available — This state indicates the session is not available for use.
- Free — This state indicates the session was manually freed.
- Transaction — This state indicates the session is currently involved with a transaction.

This command is helpful in debugging problems with host applications and to check on the number of sessions actively involved on a call. Refer to Chapter 6, “System Monitor,” of *Intuity CONVERSANT VIS Version 5.0 Operations*, 585-310-550, for information on the Host Monitor screens.

Example

The following example displays the current status of the host applications assigned to sessions 0 through 9.

hstatus 0-9

The following example displays the current status of all session numbers.

hstatus all

iCk, iCkAdmin

The **iCk** process is the daemon process which performs various integrity checks on the Intuity CONVERSANT system based on rules in a script file.

The **iCkAdmin** command is a related administration command to **iCk**.

Synopsis

iCk ***[-v NNN] [{envName}={value}] [{rule-file}]***

iCk ***-c [-i | -f {file} | cmd...]***

iCkCmd ***[-i | -f {file} | cmd...]***

iCkAdmin ***[-c] [-a {on|off}] [s {entryType [:{ID}]}]***
[-e {entryType [:{ID}]} [iCk.rules-file]

Description

The **iCk** process performs various jobs that fall into the category of “integrity” checks. It is driven by an ASCII file containing rules describing the checks desired to be performed. Its primary job is to run as a daemon process, started by **init**, and to perform the specified jobs at the intervals specified by the rules. **iCk** ‘s secondary job is to serve as a command interface to a human user and convey commands to the **iCk** process which is running as a daemon process.

As a daemon process, **iCk** accepts one flag, the *-v* flag, which initializes the internal verbosity flags according to the value NNN provided. This value can be in decimal, hexidecimal, or octal. None of the symbolic flag names apply in this mode. The bit meanings are as follows:

Table 1-6. Verbosity Flag Values

Bit	Name	Description
0x0001	V_RESCANBB	Log messages whenever the bulletin board is rescanned
0x0002	V_TIMINGMSG	Log messages when timing messages are sent
0x0004	V_HUNGPROCESS	Log messages when hung process checking is performed
0x0008	V_AUTOREBOOT	Log messages when "autoreboot" processing is performed
0x0010	V_FILEMAX	Log messages when maximum file checks are performed
0x0020	V_FILECHECK	Log messages when file ownership/modes are checked
0x0040	V_PIPECMDS	Log messages when pipe commands are received
0x0080	V_TRACE	Log messages about all major routines in **iCk**
0x0100	V_SERVICE	Log messages whenever a service is queued or performed

Environment Variables

The **iCk** command also accepts environment variables from the command line of the form:

{variable-name}={value}

These can be used to set the following environment variables that also affect **iCk**'s behavior:

VERBOSITY This is an alternative way to set the internal verbosity flags. The meanings of the bits are the same as for the value supplied to the *-v* flag.

SHELL This specifies the name of the shell to be used when executing commands. The default is **/bin/sh**.

UTMP This specifies where the "utmp" file associated with the system is located. Currently, this value is not used except for debugging purposes.

PATH — This indicates where **iCk** finds executable programs. The default is **/bin:/etc/:/usr/bin:/vs/bin:/vs/bin/util:/vs/bin/tools**.

When running as a daemon process, **iCk** accepts a file name, which is the name of the rules file from which it is supposed to operate. If not specified, the default rules file is **/vs/etc/iCk.rules**.

When **iCk** is executed with the *-c* flag or by the alternate name **iCkCmd**, is run as the command interface to the **iCk** daemon process.

-i — This option specifies that **iCk** to run in interactive mode. This causes it to generate prompts as it requests information from its standard input. Without the *-i* flag, **iCk** silently accepts input from its standard input. This might be useful if used in a shell script.

-f {file} — This value causes **iCk** to read a series of commands from the specified file or device instead of from its standard input.

{cmd}... — This field causes **iCk** to use the remaining arguments on the command line as the commands to be sent to the **iCk** daemon process.

See the “Commands” section for details about commands to which **iCk** will respond.

Administers Rules File

The **iCkAdmin** command administers the **iCk** rules file. It has no direct communication with the **iCk** daemon process. Changes it might make to the rules file do not take effect until the **iCk** daemon process is requested to read the modified rules file.

- *-c*

  This option causes **iCkAdmin** to verbosely check out the rules file and report complaints.

- *a {on|off}*

  This option causes the rules file to be read, the **"autoreboot"** entry set the specified state, and written back out again.

- *-s {entryType[:{ID}]}*

  This option causes the rules for the specified entries to be shown.

- *-e {entryType[:{ID}]}*

  This option allows interactive editing of the specified entries. *This feature is not yet complete*.

For both the *-s* and *-e* options, the **entryType** is the name of a type of entry minus the “$” character, that is, **rescanBB**, **timingMsg**, etc. The optional *{ID}* field

means the name of the process for **timingMsg** and **hungProcess** entries and the name of the file for **fileMax** and **fileCheck** entries.

Rules File

Comments begin with the "#" character and continue to the end of the line. All blank lines are ignored. Activity requests are indicated by keywords, all of which begin with the "$" character.

In the descriptions of the activities, the following definitions apply:

- *{process}* — This is the ASCII name of a process appearing in the bulletin board, that is, TSM or MTC.
- *{runlevels}* — This is specification of which run levels at which to perform the activity.The syntax is the same as used by init, that is, 4 = run level 4, 234 = run levels 2, 3, or 4.
- *{checkPeriod/Time}* — This indicates the activities performed repetitively will have a specification of either how often to perform the activity or at what times of the day or week to perform the activity. One of three forms is used:

  — -

  Perform the activity once when the rules are first read and then do not perform it again.

  — checkPeriod

  A period of time is specified as the sum of a number of different time elements: [NNd] [NNh] [NNm] [NNs]. For example, 5m means "every 5 minutes," and 5h 30m means "every five and a half hours." Each element is a number followed by the type of specifier, d, days, h, hours, m, minutes, s, seconds. The order is irrelevant. 5h 30m is the same as 30m 5h.

  — Time

  If it is more important that an activity be performed at a specific time of day or week, then the "time" format should be used. It has the following form: **X {monthday} {weekday} {hour} {min} {s**ec}

  All five elements are required for the specification to be accepted. Each element can be:

  - *

    All items in class (days of the month, hours in the day, etc.)

  - N

    The specific item.

  - N-M

The items between N and M inclusive.

- N,M

  The individual items N and M in the class.

The items within each class are:

- *{monthday}* — 1-31
- *{weekday}* — ASCII day of the week (sun, mon,...)
- *{hour}* — 0-23
- *{min}* — 0-59
- *{sec}* — 0-59

For example: "* * * **0 0**" means perform each hour on the hour. "**13 fri 12 0 0**" means perform the activity at noon on any Friday the 13th.

- *{cmd}* — This specific command is executed if the activity so dictates. Within the command itself, there are four meta-words that can be used to generate flexible commands. Not all four meta-words have meaning in all cases.

  ***%f*The full file name**

  ***%d*The directory portion of the file name**

  ***%b*The base name of the file name**

  ***%p*The process identification (PID) of the process**

Activities

- **$timingMsg *{process} {runlevels} {checkPeriod/Time}***

  This activity causes a timing message to be sent to a specified process at regular intervals whenever the system is at one of a specified run levels. Currently, the TSM and the VROP processes expect to receive timing messages, once every 2 seconds.

- **$hungProcess *{process} {runlevels} {checkdPeriod/Time} {timeout{fill|report|exec cmd}}***

  This activity causes a specific process, whose name appears in the bulletin board, to be evaluated to see if it is hung in regard to reading its messages. Processing only takes place when the system is at one of the specified run levels. *{timeout}* is the length of time the process can stay in the "working" state before being declared hung. Once a process is determined to be hung, one of three responses are possible:

  — kill

  The process is killed by sending it first a **SIGUSR1** signal, followed by a **SIGKILL** signal if it does not voluntarily exit.

— report

A message is logged to the effect that the process is hung. No other action is taken.

— exec

The specified command is executed. The %p meta-word has the value of the PID of the process associated with the rule.

- **$autoReboot *{off/on} {u-reboots} {ubPeriod} {runlevels} {setPeriod}***

This activity controls the feature which automatically sets the UNIX kernel auto-reboot flag. If the entry is marked “off,” then the auto-reboot flag is not automatically turned on. It can still be manually set with an **iCk** command. If the entry is marked “on,” then the automatic setting is enabled. The remaining parameters control when the flag is set. The algorithm that controls the setting of the flag is as follows:

1. The number of unanticipated reboots of the kernel is determined by examining the **/etc/wtmp** file (the history file of **init** actions) for “change of run level” entries and “boot time” entries. Any entry falling within the {*ubPeriod}* of time prior to the most recent system boot time are considered. If a “boot time” entry is preceded by a “change of run level” to levels 0, 5, or 6, the boot is considered anticipated, since someone deliberately entered the command to reboot the system. If a “boot time” entry is NOT preceded by such a “change of run level” entry, then the reboot is considered unanticipated. This includes power failures, reset button pushes, and panics of the UNIX kernel.
2. If the number of unanticipated reboots is LESS than *{u-reboots}*, the auto-reboot flag is set *{setPeriod}* amount of time AFTER the system comes up to one of the run levels specified by *{runlevels}*.
3. If the number of unanticipated reboots is GREATER THAN OR EQUAL to *{u-reboots}*, setting of the auto-reboot flag is inhibited and is not set until the system has been up at one of the run levels specified by *{runlevels}* for a *{ubPeriod}* of time.

For example: typing **$autoReboot on 5 60m 4 5m**, which is the standard default setting specifies that if LESS THAN 5 unanticipated reboots have occurred in the past 60 minutes, the auto-reboot flag is set in the UNIX kernel 5 minutes after reaching run level 4. If 5 or more unanticipated reboots have occurred in the past 60 minutes, then the auto-reboot flag is not set until 60 minutes after reaching run level 4.

- **$fileMax *{file} {maxSize} {checkPeriod/Time} reduce {minSize}***
  $fileMax *{file} {maxSize} {checkPeriod/Time} remove*
  $fileMax *{file} {maxSize} {checkPeriod/Time} exec {cmd}*

  This activity checks one or more files to insure that they have not grown too large. *{file}* is the name of a file or a pattern specifying a set of files. *{maxSize}* is the maximum size in bytes that a file to grow to before it triggers a response from **iCk**. A check on the size of the file or files is made as specified by *{checkPeriod/Time}*. One of three responses to a file becoming too large can occur:

  — reduce

  The offending file is reduced in size by saving the last *{minSize}* bytes of the file and discarding the rest.

  — remove

  The offending file is removed entirely.

  — exec

  The command specified is executed. In this case the meta-words **%f**, **%d**, and **%b** are defined as the various parts of the file name and can be used in the command.

- **$fileCheck *{file} {runlevels} {checkPeriod/Time} {type} {owner} {groups {modemask} {modes} [cmd]***

  This activity can be used to insure that a specific file or files exist and have the proper ownership and modes. *{file}* specifies the file or a pattern which selects a set of patterns. *{runlevels}* specify at which run levels the checks are made. *{checkPeriod/Time}* specifies the frequency of checks. *{type}* specifies the type of file. It can take one of seven values:

  — - The type does not matter.

  — f The file is a “regular” file.

  — d The file is a directory.

  — p The file is a named pipe.

  — c The file is a character special device.

  — b The file is a block special device.

  — l On SVR5.4 systems, the file is a symbolic link.

  The *{owner}* variable specifies who owns the file. If this value is - then who owns the files is not of interest. *{group}* specifies which group owns the file. If this value is - then which group owns the files is not of interest. *{modeMask}* specifies which bits of the mode are of interest while *{modes}* is the state of the bits desired. For example, if both *{modeMask}* and *{modes}* were 0444, then the check would be to insure that the file was readable by anyone, but whether it was writable or executable is not of interest. If on the other hand *{modeMask}* was 0777, while *{modes}* was 0444, then the check would be to insure that the file was only readable

and must not be writable or executable by anyone. If a file fails to pass a $fileCheck test, it is always reported. If the optional *[cmd]* is specified, then this command is executed. The meta-words %f, %d, and %b are set to the various parts of the file name for use in the command.

- **$EOF**

  This special mark indicates the end of the rules. Anything beyond this mark in the rules file is ignored.

Example Rules

$fileMax /etc/wtmp 360000 ~* * * 0 0~ reduce 36000
If the file **/etc/wtmp** exceeds 360,000 bytes, reduce it to 36,000 bytes. Check the size of the file on the hour. (The structures in this file are 36 bytes in length and it must be an integral number of structures, hence the chosen sizes.)

$fileCheck /etc/passwd - - f root - 0777 0444
Check only once. The **/etc/passwd** file should be owned by root and be read-only to everyone.

$fileCheck /etc/shadow - - f root - 0777 0400
Check only once. The **/etc/shadow** file should be owned by root and be read-only only to root.

$fileMax /tmp/*.lst 10000 - remove
Remove all the files in **/tmp** ending with an extension of *.lst* if they are bigger than 10,000 bytes. Do this only once.

$fileMax /tmp/*.hist 0 - exex ~/bin/mv %f %d/o.%b~
For any non-zero length files in **/tmp** with an extension of *.hist*, save them as **/tmp/o.*.hist**

Commands

In command mode **iCk** responds to the following commands. Each command sends a message to the **iCk** daemon process except for the first command. All commands can be abbreviated to the shortest unique string, hence **au** is sufficient to identify the "autoreboot" function and **ac** the **activate** function. For most commands one letter is sufficient.

- **x “ | “ exit “ | “ ^D**

  This command exits from the interactive command mode. This does not affect the **iCk** daemon process.

- **bootCnts [period]**

  This command computes the UNIX reboot information from the **/etc/wtmp** file. If *period* is supplied, this length of time is used. If it is not supplied, then the window period of time for the **$autoReboot** rule is used. This command generates three numbers, the total number of reboots in the specified period of time *prior* and including the current boot of the system, the number that were anticipated (or deliberate) and the number of unanticipated reboots. This request does not communicate with the **iCk** daemon process.

- **autoReboot {set|clear}**

  This command forces the kernel auto-reboot flag into the specified state.

- **readRules [rule-file]**

  This command rereads the rules file. If a new file name is provided, then it is read instead of the previous file. Before using this command, the new rules should be checked with the **iCkAdmin** command to insure syntactic correctness.

- **wakeup**

  This command makes the **iCk** daemon wakeup immediately and check its state.

- **rescanBB**

  This command makes the **iCk** daemon wakeup and reexamine the bulletin board for new instances of known process types.

- **quit**

  This command causes the **iCk** daemon to exit gracefully. (Since **iCk** is normally run from the **/etc/inittab** file, **init** immediately respawns the daemon.) In interactive mode, the command requires confirmation.

- **verbosity *{value}***

  This command sets the **iCk** daemon’s verbosity flags to the specified values. In this case the symbolic names are accepted as well as octal, decimal, or hexidecimal values. Combined values can be produced by separating values with the ‘|‘ character.

- **activate *{spec}***

  This object, in conjunction with the V_TRACE flag, causes the activities specified by *{spec}* to be logged whenever they execute.

- **inhibit *{spec}***

  This object, in conjunction with the V_TRACE flag, causes the activities specified by *{spec}* to not be logged whenever they execute.

- **print *{spec}***

  This object logs the status of the activities specified by *{spec}*. The status information logged as a result of the **print** command varies based on the activity. The common information printed is the activity index, which may be used in future *{spec}* 's, the rule index, which should correspond to the position of the rule in the rules file, and the type of the activity. In addition, there is the *a_clockID*, which is non-zero if an alarm is running for the current activity and the **a_nextAlarm**, which indicates at what time the next alarm is set to expire. At the end of the entry is the **a_flags**, 0, meaning no flags are set, AF_SUPPRESS_TIMING, meaning that timing is deliberately suppressed for the time being, AF_CHECK_NEW_RUNLEVEL, meaning that when the run levels change, this activity is checked to see if it should reactivate, and AF_DEBUG_OFF, which is set for any activity that has been inhibited by the **inhibit** command. There is also the **a_state**, which indicates the current state of the activity. Its values are:

  - AS_INACTIVE — This value is currently not being processed.
  - AS_TIMER_RUNNING — There is currently an alarm outstanding for this activity.
  - AS_SERVICE_QUEUED — An alarm has expired for this activity, but has not yet been processed.
  - AS_IN_PROGRESS — An activity is currently being processed.

The above-mentioned commands, activate, inhibit, and print, require an activities specification. Such a specification is defined from the following list of objects. More than one object can be combined with the '|' character:

- **rescanBB**

  This object is the **$rescanBB** activity.

- **timingMsg**

  This object is all the **$timingMsg** activities.

- **hungProcess**

  This object is all the **$hungProcess** activities.

- **autoReboot**

  This object is the **$autoReboot** activity.

- **fileMax**

  This object is all the **$fileMax** activities.

- **fileCheck**

  This object is all the **$fileCheck** activities.

- **miscellaneous**

  This object applies to the **print** command only. It causes a report of whether the autoreboot flag has been automatically set or not, the state of the UNIX kernel autoreboot flag, the current run level, the number of rules read, and the number activities currently in force to be logged.

- **all/ALL**

  This object specifies all activities.

- **NNN**

  This object, where NNN are digits, specifies an explicit activity by its index in the array of all activities.

All remaining information is activity specific. By activity the information logged is:

- **$timingMsg**

  The name of the process, the bulletin board slot, and instance.

- **$hungProcess**

  The name of the process, the PID, the bulletin board state, work count, time, flag, slot, and instance. The flag can have values of HP_STUCK, meaning that it does not seem to be reading its message queue, HP_SIGUSR1, meaning it has been sent a SIGUSR1 signal to request it to die, and HP_SIGKILL, meaning that it has been killed with the uncatchable SIGKILL signal.

- **$autoReboot**

  The computed unanticipated reboot count at the time the system was last rebooted plus the length of the period over which the computation is made.

- **$fileMax**

  The name of the file.

- **$fileCheck**

  The name of the file.

- **core**

  This command is available for debugging purposes. It causes **iCk** to produce a core file in **/tmp/iCk.core** via a core dump operation is a spawned child process. In other words, **iCk** itself does not stop, but you do get a reliable core of **iCk** for debugging evaluation.

Default File

The **iCk** process responds to default parameters placed in **/vs/etc/default/iCk**. Initially there are two values, which set specific internal parameters:

- RUNLEVELTIMEOUT

  This parameter specifies how long to wait after changing run levels before accepting the value from **/etc/utmp** without confirmation from **iCkCmd**. The default is 3 minutes.

- RECHECKTIMEOUT

  This parameter specifies how long to wait after changing run levels before rechecking for new processes in the bulletin board. The default is 30 seconds.

  Also any environment variables desired can be set in the default file.

Files

/vs/etc/iCk.rules	# the default rules file
/tmp/iCkPipe	# the named pipe used to speak to iCk
/vs/etc/default/iCk	# default parameters

Caveats

The **iCk** process is a daemon process running as “root.” Since the rules support the concept of executing an arbitrary command, the **/vs** and the **/vs/etc** directories need to be protected against tampering and the **iCk.rules** file should only be writable by authorized users.

See Also

"logCat"

install_appl

The **install_appl** command installs an application.

NOTE:
This command is valid only if the Enhanced File Transfer package is installed.

Synopsis

install_appl -n *[o] <application name>*

Description

The **install_appl** command is used to install a verified application received from the host, bundled using the **"backup_appl"** command. It requires the name of the application. The *[o]* option overwrites the existing application.

NOTE:
You must use the "restore_appl" command before using the **install_appl** command.

Return Values

If the **install_appl** command is successful, a 0 value is returned. If any value other than 0 is returned, the **install_appl** command failed. The following are the possible reasons for failure for the **install_appl** command:

- The hard disk is low in space.
- The command syntax is incorrect.
- The application already exists.
- The application has not been verified.
- Unrecorded phrases exist.
- An inconsistency is found in the transaction.

Example

The following example installs the application “bank_balance” received from the host.

install_appl -n bank_balance

See Also

"backup_appl"
"remove_appl"
"restore_appl"

install_sw

The **install_sw** command installs a software package.

NOTE:
This command is valid only if the Enhanced File Transfer package is installed.

Synopsis

install_sw ***[-p <path>] [-n <cpio file name>]***

Description

The **install_sw** command is used to install a software package received from the host. The package is a file in cpio format. If the path and the cpio file name are not specified, the default path (**/tmp/stag**) and the default file name (**h_install**) is used.

Use the following command to view the contents of a floppy to determine if it is an installable package and that is does not use full path names, like **/etc/profile**:

cpio -iBcvtI /dev/rdsk/f0

The following files are needed to perform the UNIX "installpkg" command: **Size**, **Name**, **Files**, **Install**, **Remove**, and ***<package name>***.

The following steps creates a sample “bundled” cpio file called fts from an installable UNIX package:

1. Insert the first floppy of the package to be bundled in the floppy disk drive.
2. Enter **mkdir mydir**
3. Enter **cd mydir**
4. Enter **cpio -iBacvdI /dev/rdsk/f0**
5. Enter **find . -print | cpio -ocd > fts**
6. Transfer this binary file to the target system using file transfer or enhanced file transfer.
7. Enter **install_sw** on the target system.

Return Values

If the **install_sw** command is successful, a 0 value is returned. If any value other than 0 is returned, the **install_sw** command failed. The following are the possible reasons for failure for the **install_sw** command:

- The hard disk is low in space.
- The user is not root or a super user.
- The command cannot read “path>/<name>.”
- The package already exists.
- The command syntax is incorrect.
- The package is missing the necessary installation programs.

Example

The following example installs the “sbpkg” cpio file which contains the Intuity CONVERSANT Script Builder Version 5.0 package.

```
install_sw -n sbpkg
```

See Also

"remove_sw"

installpkg

The **installpkg** command installs a software package.

Synopsis

installpkg

Description

The **installpkg** command loads a software package from floppy disk to the hard disk. You must be logged in as **root** to execute the **installpkg** command.

The **installpkg** command performs the same functions as installing software through the System Administration menu. It is recommended that **installpkg** be used when loading software into a new system. Once a system is running, use the System Administration menu to load any additional optional software packages. For more information about installing a package through the System Administration menu, see *Intuity CONVERSANT VIS Version 5.0 Operations*, 585-310-550.

Example

The following example invokes the program to load a software package from floppy disk to the hard disk.

installpkg

into_et

The **into_et** command sends error messages to the error tracker (ET).

NOTE:
This command can only be used in the logger/alerter environment (prior to Version 3.1). Refer to the **"logit"** command for additional information on sending error messages (post Version 3.1).

Synopsis

into_et ***[-fhlpstv] [-n no_messages] [-r error no_times]***

Description

The **into_et** command sends the error messages to ET and is used primarily to verify that the specified errors are being recognized by ET. Once errors are sent, display the Event Log Report in the **"cvis_menu"** system to see if ET logged them appropriately.

The error messages can be specified as the numeric message id or as the mnemonic (the default). After processing its options (except for the -r option), into_et goes into interactive mode where error messages can be entered one per line using the following format:

mnemonic channel card e_arg[0] e_arg[1] e_arg[2] e_arg[3] e_strarg

or

message id

The input fields above are separated by tabs and/or spaces and are optional except for the mnemonic or message id. The mnemonic should be exactly as defined in the **/att/msgipc/etmsgs** directory's files or **into_et** rejects it. ET applies the e_arg and e_strarg fields to the rule associated with the error to form the text description; they are the same arguments that are passed to **et_send**.

To quit **into_et**, perform the following procedure:

1. Enter a lower case letter for the mnemonic.
2. Enter a number less than 100 for the message id.
3. Press DELETE.

The into_et command recognizes the following options:

- *-a* — Sends all message ids known to **into_et**. **Into_et** sends 20 at a time, then sleeps for 5 seconds before continuing with the next 20.
- *-f* — Provides a fuller prompt message instead of the default terse message while in interactive mode.
- *-h* — Displays a help message.
- *-l* — Accepts message ids as input instead of mnemonics (the default).
- *-n no_messages* — Sends at most the specified maximum number of messages. **Into_et** exits after the number of messages sent equals no_messages.
- *-p* — Displays all known messages ids or mnemonics. It sends all message ids known to **into_et**. **Into_et** sends 20 at a time, then sleeps for 5 seconds before continuing with the next 20.
- *-r* — Sends the specified error message the specified number of times. The error argument is either a mnemonic or message id depending on whether option -1 is specified. **Into_et** exits after the last message is sent and thus it skips interactive mode.
- *-s* — Accepts an unknown mnemonic and send message id 17 that is unknown currently to ET.
- *-t* — Sends a message in file into.t in the current directory. Each line in into.t specifies an error message to be sent, having the same fields as when specifying error messages interactively (see above). However, note that **into_et** interprets the first field as a mnemonic only; the first field should be a message id.
- *-v* — Turns on verbose mode. All arguments sent to ET are displayed.

ET expects every error message received to have a source of the error. Error messages sent **into_et** have ET as the source of the error. In effect, **into_et** pretends to be ET.

Upon successful completion, **into_et** displays the total number of error messages sent and an exit status of zero is returned. Otherwise, **into_et** prints an error message on stdout and returns a non-zero exit status.

NOTE:
After the initial prompt, **into_et** no longer prompts for a error message which makes it difficult to tell if the message has been sent and whether to proceed to enter the next error message. If the cursor rests on the beginning of the next line, you can enter the next error message.

NOTE:
The **into_et** messages are subject to flood control, meaning that multiple messages of the same type are not shown in the error log unless flood control is turned off. See the discussion of **etset** for more information.

Examples

In the following examples "<cr>" stands for carriage return. Assume that mnemonics (message id) DIP_READ_ERROR (1440) and DIP_HOST_ACCESS (1450) are known while BAD_ARGS and BAD_NAME are unknown to **into_et**.

Example 1: The following example sends DIP_READ_ERROR to ET twice.

```
into_et -f <cr>
DIP_READ_ERROR <cr>
DIP_READ_ERROR <cr>
q <cr>
```

Example 2: The following example is an alternate way to send the DIP_READ_ERROR to ET twice with verbose mode on.

```
into_et -v -r DIP_READ_ERROR 2 <cr>
```

Example 3: The following example is a third way to send DIP_READ_ERROR to ET twice>

```
into_et -l <cr>
1440 <cr>
1440 <cr>
1 <cr>
```

Example 4: The following example sends DIP_HOST_ACCESS, DIP_READ_ERROR, and BAD_ARGS to ET.

```
into_et -s <cr>
DIP_HOST_ACCESS <cr>
DIP_READ_ERROR <cr>
BAD_ARGS <cr>
q <cr>
```

See Also

et_send
"logit"

lComp

This command combines a series of message files and produces a file of compressed format files and an expansion format file.

Synopsis

Comp ***[-s name] [-c name] [-t name] [-d name] [-m name] <file1> [file2...]***

Description

lComp compiles logging format files. The input files are in the form:

XXX...NNN... message.....%fff[<<SQL spec>>]....
%fff[<<SQL spec>>]....%fff[<<SQL spec>>]...

In other words, the input files contain standard C format statements, with optional SQL field definitions included. Long lines may be broken up with backslash, newline sequences. Such lines are concatenated, discarding the backslash and newline characters, by **lComp** and treated as one long line during compilation.

lComp produces five files, a header file, a compressed format file, an expansion format file, a data dictionary file, and a data dictionary mapping file. The default names are: **systemLog.h**, **cmpLogFmt**, **textLogFmt**, **dataDictLog**, and **ddMapLog**.

-s name	Changes the **systemLog.h** file to ***name***
-c name	Changes the **cmpLogFmt** file to ***name***
-t name	Changes the **textLogFmt** file to ***name***
-d name	Changes the **dataDictLog** file to ***name***
-m name	Changes the **ddMapLog** file to ***name***

The **systemLog.h** file contains a series of defines of the form:

#define _*{FILE}*_START NN

where *{FILE}* is the all uppercase form of the input file name. This header file allows applications to refer to errors of a specific class relative to the beginning of the class of errors and so avoid having to edit code as the various classes of errors codes grow or shrink.

The **cmpLogFmt** file contains the compressed formats, which the log subroutine uses to produce compressed logging messages.

The **textLogFmt** file contains two sections. The first section is a series of offsets to each expansion format and its length. The second section contains the expansion formats, which **expandLog** uses to convert a compressed logging file into a human readable statement.

The **dataDictLog** file contains SQL names for the variable fields in each message. They are of the form:

abs_index <FS>fld-name,type[,length[,precision]]<FS>...<GS>

The *abs_index* is the index number of the message within the universe of all messages compiled by **lComp**. If the optional SQL specification does not appear after the format, **lComp** generates one of the form:

CLASSNNN_M,type[,len[,precision]]

based on the format. *CLASS* is the uppercase name of the file the message came from, *NNN* is the index of the message within the file, and *M* is the field within the message, starting at 1.

The **ddMapLog** file contains structures describing where to find each data dictionary entry for each message. It also contains an array with the class names.

See Also

"logCat"

list

The **list** command lists the directory entries for specific phrases in the UNIX file.

Synopsis

list -l *[phrase <phrase list>] [in] [talkfile <talkfile list>]*

Description

The **list** command displays the phrases stored in the specified talkfile. The valid arguments for the **list** command are:

- *<phrase list>* — Specifies the number (or range) of phrase(s) to be listed. If you want to list all phrases in a particular talkfile, enter **all** for *<phrase number>*.

  The following example displays all phrases in talkfile 104:

  list phrase all in talkfile 104

- *<talkfile list>* — Specifies the number (or range) of talkfile(s) containing phrase(s) to be listed. If you want to list a particular phrase number in all talkfiles, enter **all** for *<talkfile number>*.

  The following example displays phrase 1010 in talkfile all:

  list phrase 1010 in talkfile all

The listed entries are sorted by talkfile number and phrase. The information printed for each phrase consists of talkfile number, phrase number, phrase size in bytes, phrase size in blocks, the phrase length in seconds, and the speech coding type.

NOTE:
The **list** command lists the directory entries for specific phrases in the SPEECHDIR default directory, which is /home2/vfs/talkfiles.

Examples

The following example displays phrase 174 as stored in talkfile 25.

list phrase 174 in talkfile 25

The following example displays phrase 12 as stored in talkfile 1.

list phrase 12 in talkfile 1

The following example displays all phrases stored in all talkfiles:

list phrase all in talkfile all

See Also

"add"
"copy"
"erase"

load_aru

The **load_aru** commands are used to download the parameter settings of the ARU.

Synopsis

load_aru_b

load_aru_c

Description

The **load_aru** commands are used to load the permanent parameter settings into the ARU associated with this system. This command should only have to be issued when the ARU is initially installed or if the settings have become corrupted. The ARU retains its settings even when powered down.

Two models of the ARU are available. These models require different initialization settings and self-retire alarms at different rates of speed. The form of the command you use depends on the model. Check the 8-digit model identification number in the upper left-hand corner of the back of the ARU. If it includes the letter “B,” use the **load_aru_b** form of the command. If it includes the letter “C,” use the **load_aru_c** form. The C model also includes a scan points connector on the right of the back panel that is not found on the B model.

Two settings files can be downloaded. The default set makes minor alarms self-retiring, but not critical and major alarms. The retire set makes all three alarm types self-retiring. When you are prompted for the settings file option, enter the file name exactly as presented on the screen; for example, **aru_dflt_b** or **aru_retire_b** for the B model and **aru_dflt_c** or **aru_retire_c** for the C model.

To retire an alarm manually, use the **"retire"** command or push the release (RLS) button on the front panel of the ARU.

Examples

load_aru_b

load_aru_c

See Also

"retire"

load_bin

The **load_bin** command initializes the 3270 host card.

Synopsis

load_bin *<binary_config_file> <card_number>*

Description

The **load_bin** command loads the executable program and binary configuration files into the specified 3270 host card for the VIS. The **load_bin** command reads in the specified binary configuration file to obtain the configuration data and executable program indication. It also resets the appropriate 3270 host card, downloads the appropriate executable program to the card, downloads the binary configuration data to the card, sends the device data to the device driver, and starts the card.

The *<binary_config_file>* argument is the filename of the binary configuration file. The binary configuration file is usually stored in **/usr/lib/3270/host.bin0** or **/usr/lib/3270/host.bin1** for card 0 and 1, respectively. Binary configuration files are produces using the **"host_cfg"** command.

The **load_bin** command should be used only when the voice system is not running and not taking calls as this command resets the sessions on the card. For dialup links, the link is dropped after executing **load_bin**.

The *<card_number>* argument is a single digit value. Values are 0 or 1. A 0 represents 3270 host card number 0 and 1 represents 3270 host card number 1.

Files

/usr/lib/3270/host.bin0	The binary configuration file for card 0
/usr/lib/3270/host.bin1	The binary configuration file for card 1

Example

The following example requests card 0 be loaded using the specified configuration file:

load_bin /usr/lib/3270/host.bin0 0

See Also

"host_cfg"

logCat

The **logCat** command reads the compressed logging files and outputs human readable messages.

Synopsis

logCat ***[-{t|b} lines] [-a locant] [-z locant] [-v] [-c] [-m] [-r root] [-s locant] [-q locant] [-w width] [-p continuation-prefix] [-d data -l log-prefix | file] [-f format] [-V]***

Description

The **logCat** command reads in a file of compressed logging messages generated by *log* and expands them to a readable format.

The default action, with no arguments, is to list all log files of the type specified first in the Config file. For example, **logCat -d${LOGROOT}/data -l{primary-log-prefix}**. The options are as follows:

- *-t lines* — Tails the last “lines” of file.
- *-b lines* — Shows beginning “lines” of file.
- *-v* — Specifies the verbose mode (that is, report the file names of the files examined).
- *-c* — Continuously displays the last lines of file. If the **logdaemon** switches to a new file, follow it.
- *-m* — This option is the meticulous time check. Normally, the log file name and the creation date are used to determine the date of the file. If the creation dates have been messed up, the *-m* flag causes the time stamp of the first message in each log file to be used instead of the name and modification date. This is slower but more reliable.
- *-r root* — Specifies an alternate root directory for **textLogFmt** file. The default is **/usr/spool/log**. Also, the **data** directory containing the compressed logging data files is expected to be in the root directory if not overridden by the *-d* flag or the LOGDATA environment variable.
- *-a locant* — Specifies the place to start printing.
- *-z locant* — Specifies the place to stop printing.
- *-s locant* — Searches for specific patterns or times.
- *-q locant* — Searches for specific patterns or times. This is the same as *-s* if the locant is a time locant. If the locant is a search pattern, the search is applied to the raw compressed log data instead of the expanded log data. This means that the pattern can only include variable portions of the logged messages. It is much faster than the *-s* option when properly applied.

A locant is one of two things, either a date/time stamp or a search pattern.

Dates can be any of the standard readable formats: mmm dd, yyyy, mm/dd/yy, mm-dd-yy, etc. The time is hh:mm:ss. It is also possible to specify the separate elements as: sec=nn min=nn hour=nn mday=mm mon=nn or mon=mmm year=nn[nn] wday=n or wday=ddd yday=nnn. Portions left out default to this date, 0 hours, 0 minutes, and 0 seconds, that is, giving only the time of day indicates today's date. If the form "item=xxx" form is used, all elements not specified default to '*', hence "wday=Sun" means all messages on any Sunday. Do not mix standard format with the "item=xxx" format. The results are not predictable.

Spaces should be enclosed in quotes, for example, **-a"7/14/87 05:08:30"**. Search patterns are enclosed in '/' characters, with an optional repetition count following, for example, -z/GEN006/2 means the second message containing GEN006. The repetition count has no meaning with the *-s* or *-q* locants, but does for the *-a* and *-z* locants.

The search capability supports the following meta-search constructs:

^Beginning of message

$End of message

***Any number of unspecified characters**

?A single unspecified character

[xxx]Any character in the list "xxx"

[!xxx]Any character not in the list "xxx"

\{chr}Normal C backslash conventions, \n \t \b \f \r \NNN \\ \/ \[

- *-w width* — If lines are to be wrapped, this is the width at which the wrapping should take place. 0 means no wrapping and is the default. The width can also be supplied via the environment variable LOGCOLUMN
- *-p continuation-prefix* — This is the string to be appended to each continuation line. The default is no continuation prefix. The continuation prefix can also be provided via the environment variable LOGCONTPREFIX data
- *-d data* — This option is the name of the directory to find the log files in. The data directory can be provided in the environment variable: LOGDATA. The default is ***${LOGROOT}/*data**. The *-d* argument takes precedence over the environment variable.
- *-l log-prefix* — Prefix of the log files to examine. The default is the first log file in the **Config** file. The log-prefix can also be provided via the environment variable LOGFILEPREFIX file
- *file* — Explicit file to be displayed. If "-", use standard input. The use of a file name overrides the *-d* and *-l* options.

- *-f format* — Format specification for printing messages. The default is **%P %T %N %S:%L\n%M**

  The format specifier uses the following notations:

  %P(..)Priority level format: %d or %s

  %T(..)Time level format: all options supported by "date" command

  %NName of process specified by the loginit call of the process

  %SSource file name

  %LLine number

  %MMessage text

  %%The % character

  \{chr}Standard C backslash conventions

  ...All other characters are printed as is. The format can be provided via the environment variable LOGFORMAT

- -V — This option makes the control characters visible. They are printed as \X if they have a special C notation, otherwise as <NNN>, where NNN is the octal value.

Environment Variables

Environment variables are checked whenever the related command argument is missing from the command line. If both the command argument and the environment variable are missing, the specified default is used.

LOGROOT
: This variable is the directory in which the **textLogFmt** is found, containing the expansion formats. Also, the directory in which the **data** directory is found if LOGDATA is not specified.

LOGDATA
: This variable is the directory in which the log data files are to be found. The default is ***${LOGROOT}*** **/data**.

LOGFORMAT
: This variable is the format in which to print the log messages. The default is **%P %T %N %S:%L n%M**

LOGCOLUMN
: This variable is the column at which to wrap long expansions. The default is 0, meaning do not wrap long messages.

LOGCONTPREFIX
: This variable is the string to be prepended to continuation lines when long lines are being wrapped. The default is no prefix.

LOGFILEPREFIX

This variable is the logfile prefix to be examined in no -/ argument is specified. If neither a -/ argument is specified nor LOGFILEPREFIX set, then the first log destination in the Config file of the type 'L' is used.

See Also

"lComp"

logDstPri

The **logDstPri** command creates the shared memory containing the dynamic destinations and priorities of logging messages using the **logMsg()** interface.

Synopsis

logDstPri ***[-H {dir}] [-c] [-v] [-d] [-x {cnt}] [rules]***

Description

The **logDstPri** command reads an ASCII rules file, described in **msgDst**, and then sets up a shared memory segment using the information in the rules file so that any process in the system using the **logMsg ()**, **vlogMsg ()**, **or logSysError ()** library calls can determine the appropriate priority and logging destinations for each message they send.

By default, the rules files are expected to appear in ***${LOGROOT}*/msgDst.rules**, where *${LOGROOT}* is **/usr/spool/log**. By default, the header files used to translate ASCII names of message indices into numbers are expected to appear in the directory ***${LOGROOT}*/head**. An alternate directory for the header files can be specified via the *-H* option on the command line. An alternate rules file can be specified as a file name on the command line.

After changing the rules file, it is recommended that the rules be checked before they are put into service. The *-c* flag causes **logDstPri** to read the rules file and report any rules that are misformatted or not understood. The return value from **logDstPri** is the number of errors detected.

To see the error complaints and install the rules all at once, specify the *-v* flag. This causes the verbose complaints to be generated. The *-c* flag implies the *-v* flag.

When **logDstPri** is resetting the values in shared memory, as opposed to creating the shared memory for the first time, it can be requested to delete the old shared memory and create a new segment by specifying the *-d* flag. Do not use the *-d* flag on a running system because any process that is already using the old shared memory continues to use it even after it is “deleted.” This means that two different rules files might be in force at the same time. It may be necessary to specify the *-d* flag if a large number of new messages have been added to the rules file. Currently, **logDstPri** creates the shared memory 200 entries larger than the highest logging message index found it its rule file. This means that as long and the new rules file does not go beyond 200 entries higher than the current highest entry, everything is okay. The number of extra entries can be altered by specifying the *-x* option.

Files

${LOGROOT}	Default is **/usr/spool/log**
${LOGROOT}/msgDst.rules	The message priority and destination file
${LOGROOT}/head/*.h	Header files used by the logging system

Shared Memory Segment

The shared memory segment is keyed off the inode of the rules file and the define symbol *LDP_KEY*, defined in **log/head/logDstPri.h**. The library routine **ftok({file},LDP_KEY)** is used to generate the shared memory key.

See Also

"logCat"
"logEvent/logMsg"
"logDstPri"
msgDst

logEvent/logMsg

The **logEvent/logMsg** command allows shell scripts to log a specific message.

Synopsis

logEvent *[script] [msg] [dst] [pri] [srcFile] [srcLine] <arg1> ...*
logMsg *[script] [msg] [srcFile] [srcLine] <arg1> ...*

Description

The **logEvent/logMsg** command allows shell procedures to log messages using specific messages. This is as opposed to the **"logit"** command, which generates messages within the logging system, but which always uses SYSMSG as the message format for the messages it generates. The **logEvent** command emulates the *logEvent( )* library routine, while the **logMsg** command emulates the *logMsg( )* library routine.

The **logEvent** command requires a destination and a priority when it is called, and messages logged via this interface are explicitly logged to the specified destinations and at the specific priority.

The logMsg command does not take a destination mask or a priority. It gets these pieces of information from the logging destination and a priority shared memory maintained by the **"logDstPri"** command via the **/usr/spool/log/msgDst.rules** file.

Both **logEvent** and **logMsg** require that the proper number of arguments be supplied for the specified message and that numeric arguments in the message format match pure numbers from the argument list. For example:

GEN012 OUT_OF_RANGE %D<<value,D>> is out of range \ for %s<<arg,S>> in %s<<routine,S>>.

This format requires that the first argument be a number, therefore,

logMsg XXX LG_OUT_OF_RANGE -- yes var compute

would fail because “yes” is not a number, while

logMsg XXX LG_OUT_OF RANGE -- 10 var compute

would work.

The following are the arguments for use with these commands:

script	Name of the shell script for which the message is being logged. Normally, this would either be **basename$0** or in ksh **${##*/}**
msg	The symbolic name of the message, for example, **LG_OUT_OF_RANGE**
dst	This is only used with the **logEvent** command. It is the bit mask specification of where the message will be sent. It may be a number or symbolic destinations, as specified in *msgDst.rules*. If more than one symbolic destination is specified, they should be concatenated with '**+**,' for example, **stderr+log**
pri	The priority of the message. This is only used with the **logEvent** command. It may be any of the following: **0**, **-** or **NONE**, **1**, **M** or **MANUAL, 2, *** or **MINOR, 3, **** or **MAJOR, 4, *C** or **CRITICAL**
srcFile	The name of the file from which the **logEvent** or **logMsg** command is being issued. If you do not care, you may use "**-**." Supplying the correct value is of value for debugging purposes, particularly if a script might generate at the same message from more than one place. If there are many individual functions within your script, you might find it advantageous to use the name of the function instead of the file.
srcLine	The line within the file from which the **logEvent** or **logMsg** command is being issued. If you do not care, you may use "**-**." You might use **$LINENO** from the ksh environment, which is the line with the script or within a function.
arg1	For each argument required by a specific message format, one argument is required. Neither too many or not enough is acceptable. Also, the size and type of the argument must be appropriate: **%s** (takes any kind of argument) **%d %u %o %x %X** (argument must be a pure integer type number, for example, -**10**, **5**, **0177**, **0x8e**) **%f %e %g %E %G** (argument will be interpreted as a pure floating point number, for example, **15**, **15.3**, **1.56E3**) **%c** (argument must be a single character, for example, **x**, **5**, **%**)

See Also

"logCat"
"logit"
"logDstPri"
log

logFmt

The **logFmt** command displays and changes the parameters used to display messages and explanation texts, specifically the messages mnemonics and screen width.

Synopsis

logFmt ***[global] {display|interactive|{opt}={value}}***

Description

Each logging message has a class name and a mnemonic name associated with it. A class name, for example ICK001, is the combination of the name of the class, for example, ICK, and the index of the message within the class, for example, 001. The mnemonic name is a short composite string of characters which identifies the type of logging message. The mnemonic name for ICK001 is ICK_BAD_CMD. By default the mnemonic names of messages are not displayed when **display messages** is used to examine the logging files. If you want the mnemonic message names to appear, then **logFmt** allows you to alter the system so that they either appear for everyone by default or appear for you specifically.

You can also adjust the width of the screen display. By default the screen width is set to 75 characters. If you have a wider screen, you may wish to specify that more of the screen be used to display messages.

global	This modifier causes the action specified to operate on the "global" (system wide) parameters that control the behavior of **display message**. You must be **root** if you want to change the global parameters. You can examine the global parameters without being **root**.
display	This verb causes **"logFmt"** to display the current parameters. If *global* is specified, then the system-wide parameters are displayed, otherwise your personal parameters are displayed.
defaults	Specifying *defaults* without the *global* option causes your personal preferences about mnemonics and screen width to be removed. You then get the system-wide settings. Specifying *defaults* with the *global* option causes the system-wide settings to be reset so that mnemonics are off and the default screen width is 75 characters.
interactive	This option interactively prompts for the parameters controlled by **"logFmt"**. Pressing ENTER in response to any query causes the current value to be retained. The current value appears within **[ ]** s.
mnenonics=enable	This option causes mnemonics to be displayed when logging messages are examined with **"display messages"**.
mnemonics=disable	This option causes mnemonics to not be displayed when logging messages are examined with **"display messages"**.
width=NN	This option causes the screen width to be set to NN, where NN is between 40 and 199 columns. The default setting is 75. Do not attempt to set the screen width to a value wider than your screen can actually handle or the display will be unpleasant when using **"display messages"**.

When mnemonics are enabled, they also show up when **"explain"** is used to examine the description of a message. Whether mnemonics are enabled or not, the mnemonic name can always be used to select an explanation using **"explain"**.

Files

/vs/data/logFmtParms	# Global parameters file
${HOME}/.logFmtParms	# User's parameter file
/usr/spool/log/textLogFmt	# Current default expansion format file
/usr/spool/log/textLogFmt.Mne	# Expansion file with mnemonics
/usr/spool/log/textLogFmt.NoM	# Expansion file without mnemonics

Examples

The following example enables the mnemonics. This affects only you and overrides the system-wide setting.

logFmt mnemonics=enable

The following example sets the system wide default so that mnemonics are not displayed. Any user wishing to see mnemonics has to personally enable mnemonics. You need to be root to execute this command.

logFmt global mnemonics=disable

The following example displays the system wide settings for mnemonics and screen width.

logFmt global display

The following example sets your personal screen width to 130 characters when displaying messages using display messages.

logFmt width=130

logit

The **logit** command represents a program used to log arbitrary messages to the logging files.

Synopsis

logit ***[-p {priority}] [-d {destination}] [message...]***

Description

The **logit** command sends an arbitrary message to the logging files and logs it under GEN002. By default, the priority is E_NONE, which is 0. Alternate priorities for the logged message can be selected with the *-p* option. Legal values for *{priority}* are 0–4 or none, manual, minor, major, or critical. The message is identified as the current user.

By default, messages generated by **logit** are sent to the MASTER_LOG which is associated with the first destination defined in the ***$LOGROOT*/Config** file (normally, the flat log files with the prefix log). Messages are also generated in the expanded form on SYSDBG so that you can see what has been logged. Alternate destinations may be specified with the *-d* flag. The destination specification may be numeric (for example, 0x11) or as a series of symbolic names taken from the list MASTER_LOG, SYSDBG, SCREEN, SYSCONS, RFSTATS, and ALERTERMSG separated by "+." The destination can also be a mix of both styles. Regardless of the specification of the destination, MASTER_LOG is always added to the list.

Example

Enter the following at the system prompt:

logit -p3 -d SYSDBG+0x20 Sample major alarm

System response:

```
** Tue Oct 22 20:43:09 1992 LOGIT logit.c:136
GEN002 abc: Sample major alarm
```

The output appears in the MASTER_LOG by default and sent to the alerter pipe, as 0x20 is equivalent to destination ALERTERMSGS.

See Also

"logTest"
"logEvent/logMsg"

logTest

The **logTest** command is a program which can generate an arbitrary list of logging messages to be sent to the logdaemon.

Synopsis

logTest ***["-s dir"] [-v] [-x] ["file1 file2..."]***

Description

The **logTest** command reads a script of logging messages to be sent to the **logdaemon** process. The format of the script is:

{interval} {dst} {priority} {process} {index} [arg arg...]

Following is a description of each of the values used in the **logTest** script.

- *{interval}* — This value is the number of seconds to wait before sending this message. If this is 0, then the message is sent immediately.
- *{dst}* — This value is the destination mask. Each destination is specified in the Config file for the logging system. Normally, this value will be 0x01, meaning logging the message is the primary log files only. The standard VIS Config file looks like:

```
#     @(#)Config      4.1.1.1 20:48:10 12/19/93

L C 100000 log 10                 # dst bit 0x0001
D E 0 stderr                      # dst bit 0x0002
S E 0 SCREEN                      # dst bit 0x0004
T E 0 /dev/console                # dst bit 0x0008
N C 0 $LOGROOT/alertPipe          # dst bit 0x0010
L C 100000 alarm 10               # dst bit 0x0020
L C 100000 event 10               # dst bit 0x0040
N C 0 $LOGROOT/sccsCtlPipe        # dst bit 0x0080
N C 0 $LOGROOT/mxmtrPipe          # dst bit 0x0100

$EOF
```

Therefore, to send something to the "alarm" log and the "sccsCtlPipe" destinations, *dst* should be set to 0xa0, the combination of 0x20 and 0x80. 0x01, the master log, is added automatically to all destinations.

NOTE:
Realize that messages sent with **logTest** are independent of the logging priority and destination mechanism that makes use of **/usr/spool/log/msgDst.rules**. With **logTest** you can send a message to any destination with any priority as anybody. It is truly a test tool.

- *{priority}* — This is a value from 0 to 4, specifying the priority of the message. 0 is E_NONE, 1 is E_MANUAL, 2 is E_MINOR, 3 is E_MAJOR, and 4 is E_CRITICAL. See the **log.h** file for more details.
- *{process}* — This value is the name of the process to be forged into the logging message. Normally, each process attaches an identifying string to each message it logs. *{process}* allows **logTest** to act like any process.
- *{index}* — This value specifies the log message index associated with the message of interest. It can be either the absolute index of the message, which might be obtained by examining a compressed log file with a text editor (it is the first field of each message), or the relative name of the message index in the form **log*TYPE(index)*** where *TYPE* is the module associated with the message, (for example, GEN, LOG, SYS, PERM, etc), and *index* is the relative location of the message within the module. **logGEN(20)** means the twentieth message in the GEN category, or it can be the symbolic name of the message, (for example, **LG_NUMARGS**).
- *[arg]* — This value is the argument required by the message format. The number of arg elements depends upon the specific logging message chosen. There should be one argument for each "%" argument specifier in the **log/formats/{TYPE}msg** file that describes the logging message. If the argument include spaces, it should enclosed in double quotes. Given the following formats:

SYS001 EPERM	**File protection (modify) error %s**
SYS002 ENOENT	**No such file or directory. %s**
SYS003 ESRCH	**No such process for kill or ptrace. %s**
PERM12 PERM_SHADOW	**%s %s %s by %s**
PERM13 PERM_PROFILE	**%s's .profile %s by %s**

the following lines would be appropriate as input to **logTest**:

```
 3   0x01  2  doghouse  logSYS(2) fidget.c
10   0x01  0  editUser logPERM(12)  linda "added to" /etc/shadow root
 0   0x01  0  editUser logPERM(13)  linda  modified  root
```

The first line is logged in 3 seconds, from the process "doghouse," saying that it could not open or create the file **fidget**. The second line comes 10 seconds later, indicating that "editUser" added "linda" to the shadow password file and that **root** made the change. The third line comes immediately and says that linda's .profile was modified by **root**.

NOTE:
No checking is done on the number of arguments required. It is the responsibility of the user to supply the proper number. If the wrong number of arguments is provided, **"logCat"** complains when trying to explain the message, saying that there is an expansion failure.

The *-v* flag causes **logTest** to send copies of the messages it is going to log to the standard error output. The *-x* option prevents **logTest** from sending the messages to **logdaemon**. The *-x* automatically implies and enables *-v*, since if it does not send the message to **logdaemon**, the only other thing it can do is list it. *-x* is used to test run a script without actually logging all the messages.

To be able to use the symbolic names for the message indicies, it is necessary that **logTest** be able to read a copy of the appropriate **systemLog.h** file created by **"lComp"**. If **systemLog.h** does not exist is the current directory, the *-s* switch specifies the directory in which **logTest** can find **systemLog.h**. If **logTest** cannot find **systemLog.h**, only absolute log indicies are accepted in the input.

See Also

"logit"

mkAlerter

The **mkAlerter** command reads an alerter description and generates C or C++ code which implements the description.

Synopsis

mkAlerter ***[-M] [-o {executable}] [-p {templ-path}] [-t [-f]] [-q] [-v] [-l] [X=Y...] [{alerterfile}.A...]***

Description

The **mkAlerter** command is a program that reads an alerter description and translates it, with the help of code template files, into compilable C or C++ code. It also produces a make file for compiling the code. Alerter description files always have a ".A" extension. By default **mkAlerter** produces a single source file, with an extension of ".c". It also produces a header file (extension ".h") and a make file (extension ".mk"). If the make file already exists, **mkAlerter** does not overwrite the existing file. This allows you to modify the make file as desired without fear of it being destroyed the next time mkAlerter is used, but does take advantage of the knowledge contained in the make file template used by **mkAlerter** when it does create a make file. The source file and the header are *always* overwritten each time **mkAlerter** is run. No modifications should ever be made to these intermediate source files, since the changes are lost the next time **mkAlerter** is run. If the *-M* flag is specified at execution time, **mkAlerter** splits the source file produced into two pieces, one containing *main ()* and the other containing everything else. The source file containing *main ()* ends in "*Main.c*" with truncation as is necessary. Once produced, this file, like the make file, is be overwritten. If you wish to produce your own initialization, you can use the *-M* option and then make your changes to the "**Main.c*" file.

Normally, the make file specifies that the executable to be produced by this alerter description is the same as the name of the alerter description minus the "*.A"* extension. The *-o* option allows you to specify an alternate executable name. This is used when the make file is generated.

The code template files are normally expected to exist either in the current directory or in **/usr/lib/alerter**. If the templates are not found in either of these places, **mkAlerter** uses its own internal copies, but also reports the fact. If the templates exist elsewhere, an alternate path can be specified with the *-p* option. Each directory which should be searched is separated by ':' characters, the same as a normal UNIX PATH description.

To get the initial template files, the user can specify the *-t* option. This causes **mkAlerter** to create each of the required template files using its internal copies. At this point each site may, if desired, alter these templates to produce alerter code appropriate for its needs. By itself the *-t* flag does overwrite existing template files. The *-f* flag causes the new templates to overwrite existing ones.

The current list of template files and their contents follows:

AlertInc.t	Description of include files.
AlertCopyR.t	Copyright notice.
AlertHead.t	Template of the header file.
AlertMain.t	Description of main () function.
Alerter.t	Primary template describing the alerter program.
AlertTest.t	Description of the code to respond to timeouts for alerting.
AlertMsg.t	Template describing a subroutine to process messages for a particular logging destination.
AlertDir.t	Template describing the subroutine to handle logging messages sent directly to the alerter process.
AlertMk.t	Template for the makefile.
AlertObj.t	Template for each *.o file in the makefile.

The *-q* option is not currently implemented. It is meant to check the templates for completeness. The *-v* flag increases the verbosity of **mkAlerter** while it performs some of its activities.

Normally **mkAlerter** produces *#line* directives, which are used by the C compiler to report where errors are detected during compilation. While these are good during the compiling phase, they mislead most debuggers and make debugging difficult. The *-l* option suppresses the *#line* directives and is recommended when the debugging phase includes the use of a process debugger, such as **sdb** or **pi**.

It is also possible to specify variable assignments that appears in the make file via the X=Y syntax. Of particular interest is CC=CC, which also causes **mkAlerter** to generate C++ code rather than C code.

See Also

readAlerterDesc

mkerr

The **mkerr** command converts user application error tracker (ET) message rules used in generics prior to VIS Version 3.1 (V3.1) to a file suitable for use with the VIS V3.1 “transparent” logger/alerter environment.

NOTE:
This command is obsolete for the Version 3.1 logger/alerter environment.

Synopsis

mkerr *[-o] [-n] [error_file]*
mkerr *[var]*
mkerr *[-h]*

Description

The “transparent” ET environment allows application code compiled for VIS generics prior to V3.1 to run without modifications, conversion, or recompilation. It does suffer from the fact that old style messages do not appear in the same format as the new messages in terms of numbering or layout.

The transparent ET environment requires that a rules file be built from the original ET errors files used to drive the earlier ET supported systems. The process **"etStub"** uses the file **/vs/data/etStub.rules** for this transparent conversion. mkerr can be used to create the file **/vs/data/etStub.rules** from the user application ET rules file **gendb/data/errors** found in previous generics.

The **mkerr** command creates etStub.rules from the *{error_file}* provided as an argument. The command output file is automatically installed in the directory **/vs/data**. If no *{error_file}* is supplied, a help message is displayed explaining the usage of the command and the target of the output. If the *{error_file}* file does not exist or is not formatted in accordances with the expected ET rules file, appropriate error and help messages are displayed.

The *-o* option flags each occurrence of a message as “obsolete.” The *-n* option causes the rules to be generated without the “obsolete” flag set; hence the messages generated by these rules are not flagged as OBSOLETE. The default setting is *-o*.

The *-var* option automatically converts the **/gendb/data/errors** file and installs the result in the directory **/vs/data**. This option insures compatibility with the intent of this same command name and option in previous VIS generics.

Example

The following example converts the ET rule file errors to the Version 3.1 logging environment.

```
mkerr errors
```

mkETrules

The **mkETrules** command provides the compatibility daemon to log old et_send() messages into the new error logging system.

Synopsis

mkETrules ***[-o|-n] [{ET-errors-file...}]***

Description

The **mkETrules** command is a shell problem which produces a valid **etStub.rules** file from one or more errors files of the type that used to control the ET process. It takes as input one or more of the old style **errors** files and generates a file called **etStub.rules** in the current directory. By default all entries are created with the *O* (obsolete) flag set. The *-o* flag specifies that the obsolete flag be set explicitly. The *-n* flag specifies that the rules created not be marked with the “obsolete” flag.

See Also

"etStub"

mkheader

The **mkheader** command allocates user memory for script variables.

Synopsis

mkheader *<application name>*

Description

The **mkheader** program creates an address in user memory for each script variable. This information is stored in an ***application-name* def.h** header file and is used in naming both the output file and the allocation program. The joint usage of the same header file enables the script to interact with the transaction state machine (TSM). The *-e* option specifies exact string matches.

The **mkheader** program prompts an operator to enter three types of information at the system console. The information may be entered interactively or batched together in a single file. Interactive entries are ended by entering CTRL D. The system prompts for:

- Variable names
- Header file names in order of dependency
- Structure names with header file locations

When **mkheader** is entered with an argument (limited to 7 characters) for *application-name*, an ***application-name*def.h** header file is created for the output information. The **mkheader** program then prompts for three types of information which it uses in producing the output file.

1. It prompts the operator for the name of one of the variables - char, int, or short. Char is the only variable which requires a length (default = 1).

   It then allocates space for the variables at the beginning of the allowable user memory and places this information in the newly created header file.

2. Mkheader prompts the operator to enter header files which are needed in order to make the files covered in the third section compile. They should be named in the order of dependence. For example, if information in the header file **b.h** is needed by the header file **a.h**, header file **b.h** must be entered first and then header file **a.h**.

   Full pathnames must be given. The file **mesg.h** and the structure mbhdr are common to all scripts and are entered automatically.

   The header files can be stored in a batch file. The batch file could contain the following header files.

   #include "/att/msgipc/dbcom.h"
   #include "/att/include/shmemtab.h"
   #include "/att/msgipc/tsm stop.h"
   #include "/att/msgipc/cdata.h"

3. The last prompt is used for allocating the space for each structure.The operator is prompted to enter each header file name and its structure names. For each header file, the operator enters the word all (if all structures are needed) or specific structure names.

   Mkheader recursively allocates memory and produces ***application-name*def.h** defines for structure members which are themselves structures (except for struct mbhdr).

As a shortcut, the input for the three prompts may be stored in another file (data file) and read in each time. For example:

mkheader *application_name* < data file

Once the header files have been entered, mkheader writes a program called ***application_name*_aloc.c** to allocate the rest of user memory. The resulting source code is automatically compiled, using **mkheader.a** library functions, and then executed. This adds the remaining structure definitions to the ***application_name*def.h** header file. TSM does not allow a script to use more than 50,000 bytes of user memory. Scripts that exceed this limit are not run when data beyond the limit are accessed.

Files

/vs/bin/vs/mkheader
/vs/bin/vrs/mkheader.a

Examples

The following are examples of the prompts and the output for the mkheader program. This example shows a user who needs some space for 20 characters, 2 integers, and a short variable. The user also needs to have space declared for a structure called dowj, which is used by the script. The header file is found in **/att/msgipc/tsmdipappl.h**.

In the example, the structure size of SZDOWJ is 16, which is automatically supplied by mkheader.

console input: mkheader <application_name>

FIRST PROMPT: Type in the variables you need space for according to the following format:

type name [length]

Example 1: int yn
Example 2: char dg 20

(End input with CTRL-D)

Variable?: char dg 20
Variable?: int yn
Variable?: short cid
Variable?: int iom

Variable?: (CTRL-D)

SECOND PROMPT: Please enter any dependency files that the header files in the next section will need in order to compile. Use full path names. (End input with CTRL-D)
File name? /u/factory/file.h
File name? (CTRL-D)

THIRD PROMPT: Enter the header file name and structure names needed
to create the def.h file. Use full path names. (End input
with CTRL-D)
Header file?: /att/msgipc/tsmdipappl.h
Structures or all?: dowj
Header file?: (CTRL-D)

Compiling: application-name aloc.c
Running: application-name aloc
Output is called: application-name def.h

This is the final *application_name*def.h file produced by this example.

```
/*****PRE-ALLOCATION OF USER SPACE *****/

#define DG:0
#define YN:20
#define CID:24
#define IOM:26

/***** DOWJ STRUCTURE *****/

#define DOWJ:30
#define RCODE:30
#define TIMEDATE:31
#define CATNUM:42
#define MKTSTAT:43
#define DOWHOUR:44
#define SZDOWJ:16
```

In this second example, the command line includes a data file from which the system gets the information usually entered by the users in response to system prompts.

The data file, called “data” in this example, contains the following information:

```
char name 20
int answer
short reply
^D
/att/include/shmemtab.h
^D
/att/msgipc/cdata.h
Day_pntr cdata
^D
```

The following appears on the screen:

```
Conversant% mkheader test6 < data
Type in the variables you need space for according to the following
format:
type name [length]

Example 1: int yn
Example 2: char dg 20

(End input with CTRL-D)

Variable?:
Variable?:
Variable?:
Variable?:

Please enter any dependency files that the header files in the next
section will need in order to compile.
Use full path names.
(End input with CTRL-D)

File name?: File name?:

Enter the header file names and structure names needed to create the
def.h file. Use full path names.
(End input with CTRL-D)
Header file?: List of structures or all?:Header file?:

Compiling /usr/has/another/test6_aloc.c

Running /usr/has/another/test6_aloc

Output is called /usr/has/another/test6def.h

I am now checking for any duplicate defines that will cause problems

The following is the contents of the test6def.h file:

/******* PRE-ALLOCATION OF USER SPACE *******/

#define NAME:0
#define ANSWER:20
#define REPLY:24

/******* DAY_PNTR STRUCTURE *******/

#define DAY_PNTR 26
#define FILE_FIRST 26
#define REC_FIRST 28
#define FILE_LAST 30
#define REC_LAST 32
#define SZDAY_PNTR  8

/******* CDATA STRUCTURE *******/
```

```
#define CDATA 34
#define SCRIPT 34
#define CHAN 50
#define EQUIP 52
#define STARTTIME 54
#define STOPTIME 58
#define EV0 62
#define EV1 66
#define EV2 70
#define EV3 74
-:
-:
-:
#define EV96446
#define EV97450
#define EV98454
#define EV99458
#define SZCDATA428
```

NOTE:
Make sure that all variable names are unique without respect to case as lower case letters are changed to upper case for the final output.

mkimage

The **mkimage** command performs a complete system backup of all the contents of the root disk file system.

Synopsis

mkimage

Description

The **mkimage** command performs a complete system backup by copying the UNIX files in the **root** and **usr** file systems to cartridge tape.

NOTE:
This command can only be run from the **root** directory.

When specifying the **mkimage** command, the VIS requests to place the system into single-user mode. The **mkimage** command aborts if you do not give the system permission. Once in single-user mode, you must relogin and re-execute the **mkimage** command to continue the **mkimage** process. The **mkimage** unmounts all mountable file systems and then mounts **/usr**, **/var**, **/home**, and **/home2** file systems, which are the only file systems beside the root file system and **/stand** that appear on the root disk in a standard VIS. The system then creates a list of files to archive to tape and prompts you for the insertion of a tape.

CAUTION:
Do not rename the file systems mentioned above as the newly named file system would not be included in the image tape.

Once the image creation has finished, the tape is verified by reading the table of contents from the tape and comparing it with the original list of files used to create the tape. If any errors are found, you see the following message that directs you to check for specific files for further information about the failure:

```
ERROR:Verification failed. Wait for the light on the tape
unit to go off before removing the tape.

Three files have been written to the /tmp directory which
show the results of the backup and verification. $DISK_FILES
contains a list of all the files which were to be backed up.
$TAPE_TOC contains a list of all files which were actually
written to the tape. $DIFFOUT contains the difference
between these two files.
Analysis of these files may help in understanding the nature
of the failure.
```

Also, be sure you are using the supported cartridge type and that your tape drive is being cleaned regularly. Execute the -init 6- command to return to multi-user mode.

The **mkimage** command then returns the VIS to multi-user mode by rebooting. If no errors are found, you are prompted to make a note of the file system partition sizes after the VIS returns to multi-user mode.

NOTE:
You do not get a warning from the VIS before it reboots to return to multi-user mode.

NOTE:
The **mkimage** command can run anywhere from 45 minutes to a couple of hours creating the image tape. Several tapes could be required depending on the amount of space used in the root disk file systems.

NOTE:
The complete system image tape should only be used to restore a system root disk that has been severely damaged and needing file-system reconstruction at the lowest level. Use the **backup** and **"restore"** commands to recover from minor file damage or corruption.

Example

The following example backs up the **root** and **usr** file system to cartridge tape:

mkimage

mkMsg

The **mkMsg** command converts user application error tracker (ET) message rules used in generics prior to VIS Version 3.1 (V3.1) to the files required for the VIS V3.1 full logger/alerter environment.

Synopsis

mkMsg ***[-y {year}] [-c {company}] {error_file1} [{error_file2}...]***
mkMsg -h

Description

Full conversion from the ET environment associated with VIS generics prior to Version 3.1 (V3.1) requires that the application source be modified or converted and then recompiled. Additionally, several files used in conjunction with the ET environment must be converted to those used by the logger/alerter.

The **mkMsg** command converts the ET error rules files to a set of files used by the V3.1 alerter/logger. The ET rules file reserved for applications (generics prior to V3.1) is **/gendb/data/errors**; however, any properly formatted rules file may be converted. Prior to reformatting via **mkMsg**, you must separate your ET rules files that correspond to message classes, each class being defined by an ET message header file. The naming convention for this header file was ***{class}*_et.h**. The header file corresponding to **/gendb/data/errors** was **/att/msgipc/etmsgs/appl_et.h**. The **mkMsg** command converts the ET file named *{CLASS}* into a corresponding file ***{CLASS}*msg** which is used by the V3.1 alerter/logger. To maintain a consistent naming convention, we recommended that the /gendb/data/errors file be copied to appl prior to conversion. Following conversion, the ***{CLASS}*msg** should be copied into **/usr/spool/log/formats** where it can be used by the alerter/logger. The **/usr/spool/log/formats/formats.mk** should also be modified to include the new ***{CLASS}*msg** file.

If no ET message files are supplied as command arguments, a help message is displayed explaining the usage of the command and the target locations of the output. If the file argument is not found or is not formatted in accordance with ET rules files, appropriate error and help messages are displayed.

Each ***{CLASS}*msg** file created by **mkMsg** has a copyright header. By default it is for the current year and is assigned to "AT&T." An alternate year can be specified using the *-y* flag and the company name can be altered by using the *-c* flag.

msgadm

This program facilitates the administration of system messages in the VIS application software.

Synopsis

msgadm ***[-e] <[-f <[<input_file>|-]>|<command>]>***

Description

The **msgadm** provides an interface to the Intuity CONVERSANT VIS logger/alerter administrative files. Commands to **msgadm** may be specified individually on the command line using **msgadm** ***<command>*** or may be specified as input from a file or standard input using the *-f* flag and a file name argument as in **msgadm -f** ***<filename>*** or **msgadm -f -** for file input and standard input respectively. The *-e* flag forces **msgadm** to write $EOT after completing each operation resulting in command output to standard out.

Each command may require one or more of the following variable arguments:

- *<message_ID>* — Specifies a member of the set of system message IDs which include all those whose message class is indexed through the **systemLog.h** file and whose mnemonics appear in a configured **log*XXX*.h** file.
- *<priority>* — Specifies a priority tag as defined with the $priority operator in the **/vs/data/msgDst.rules** file. Use **msgadm** priorities to see a list of priority tags configured with the system.
- *<time>* — Specifies a non-zero positive integer indicating time in seconds if suffixed by “s,” minutes if suffixed by “m” or hours if suffixed by “h.”
- *<dst>* — Specifies the set of destination tags defined in the **/usr/spool/log/msgDst.rules** file through the $destination operator. To see the list, execute **msgadm** destinations. Note that only the latest destination specified in **/usr/spool/log/msgDst.rules** is used.
- *<threshold>* — Specifies a non-zero positive integer indicating a threshold value.

The following are examples of how the msgadm command may be used:

- **msgadm set <*message_ID*> priority <*priority*>** — Sets the priority of <*message_ID*> to <*priority*> if <*message_ID*> is already in the **msgDst.rules** file. If <*message_ID*> does not exist, an entry is created with the indicated priority and the default destination(s).
- **msgadm <[add/delete]> <*message_ID*> destination <*dst*>** — If added and specified, and <*message_ID*> exists in the **msgDst.rules** file, adds a new destination entry to the file. If the entry does not exists a new entry is created with the default destination plus the specified destination. The priority is set to the default priority. If delete is specified, <dst> is removed from the destination set for <*message_ID*>. The log or MASTER_LOG destination cannot be removed from a message.
- **msgadm set <*message_ID*> window <*time*>** — Sets the threshold window time of threshold <*message_ID*> to <*time*>. If no threshold structure has yet been created for <*message_ID*>, one is created with a threshold of 100 and threshold message set to THR001.
- **msgadm <[add/delete]> <*message_ID*> threshold <*threshold*> message <*thresh_message_ID*>** — Adds or deletes a <*threshold*>/<*thres_message_ID*> pair from the **thresh.rules** file. If add is specified, and no entry for <*message_ID*> exists, an entry is created with a threshold window of 1 hour. If delete is specified and the <*threshold*>/<*thres_message_ID*> pair is the last pair specified, the entire threshold structure is removed.
- **msgadm display <[<*message_ID*>/all]>** — Lists all administrative parameters associated with <*message_ID*> or all system messages if *all* is specified.
- **msgadm priorities** — Outputs the default list of message priorities. **msgadm destinations** outputs the default list of message destinations.
- **msgadm thresholds** — Outputs the set of thresholding messages.
- **msgadm sync** — Makes all changes made through previous calls to **msgadm** take affect in the live logger/alerter system. If sync is not used, a system reboot is required to make changes take affect.
- **msgadm -f <[<*input_file*>/-]>** — Forces **msgadm** to read from <*input_file*> or standard in if "-" is specified. The expected input is **msgadm** command line arguments as defined above, one complete set of command line arguments is expected per line. Errors in the input result in no changes to the logger/alerter configuration files regardless of where the error occurred in the input.

Examples

The following example sets the priority of system message VROP003 to critical. Note that *C has been quoted to protect it from the shell.

msgadm set VROP003 priority '*C'

The following example adds a threshold of 10 with a threshold message of THR001 to the thresholding structure for the VROP003 message. It is assumed that THR001 is a valid message Id.

msgadm add VROP003 threshold 10 message THR001

The following example displays the message administration parameters associated with message VROP003.

msgadm display VROP003

System response:

```
        Message Id:             VROP003 (VROP_NOSPBUF)
        Message Priority:       *
        Message Destinations:   log|alarm

        Threshold Period:       1m
        Message Thresholds:
        Threshold               Threshold Message Id
        ----------             --------------------
                20              THR003 (THRESH_MAJOR)

        Message Text:
VROP003  -- -- --- (VROP_NOSPBUF) No resources available on
    SP for speech %s.  Reason:  No SP window buffers.
```

The following example shows the use of the file input mechanism. This example sets message VROP003 to priority "-" (none), changes its destination from the alarm to the event and removes its thresholding structures (if any exist). It then sets TSM002's priority to "*C" (critical), assuming "*C" is defined in the **msgDst.rules** file, and makes the changes take effect in the current environment through the sync directive.

```
message -f - <<!
set VROP003 priority -
delete VROP003 destination alarm
add VROP003 destination event
delete VROP003 threshold 10 message THR003
delete VROP003 threshold 100 message THR004
set TSM002 priority '*C'
sync
!
```

See Also

"display messages"
"explain"

newscript

The **newscript** command updates the changes to all currently assigned scripts.

Synopsis

newscript

Description

The **newscript** command notifies the TSM and CDH processes that an existing script in the **/vs/trans** directory has been changed. After **newscript** is run, TSM reloads all scripts from disk when they are run next instead of using any copies it has in memory.

Files

/vs/bin/util/newscript

Example

The following example notifies the TSM and CDH processes that an existing application in the directory **/vs/trans** has changed.

newscript

reinitLog

The **reinitlog** command is the control program and is used to inform **logdaemon** that a new config file is to be used.

Synopsis

reinitLog

Description

The **reinitlog** command is used during the procedure of creating a new logger message destination, as it sends a message to the **logdaemon** that informs the logdaemon that a new config file is to be used. The **reinitlog** command causes the **logdaemon** to reread the configuration file and reopen the various logging files.

Files

$LOGROOT/Config	**The configuration file which defines destinations.**
$LOGROOT/data	**The directory in which logging files are created.**
$LOGROOT/logpipe	**The FIFO which logdaemon reads.**

See Also

ckConfig
"logCat"
"logit"

remove

The **remove** command places a unit in the manual-out-of-service state.

Synopsis

remove *<unit> <number> <immed> <min_delay> [-i] [-n]*

rem *<unit> <number> <immed> <min_delay> [-i] [-n]*

Description

The **remove** command is used to remove a unit from service when its temporary state is idle. It changes the permanent state of the unit to manual-out-of-service (MANOOS). It does not remove a unit that has a temporary state of busy. If a unit must be interrupted immediately or appears to be stuck busy, use the **rem *<unit> <number> <immed>*** command.

The parameters for the **remove** command are:

- *<unit>* — Identifies the unit. The choices are “channel” or “card.”
- *<number>* —Specifies the channel or card number, a range of channel or card numbers in the form **m n**, or the word **all** for all the channel or card numbers. Card numbers are in the form ***card#[.port#]*** where *port#* is a port of card *card#*. If *port#* is not given, all ports of the card specified are removed. If no card number or channel number is given, a syntax message is displayed.
- *-n* — Disables prompting from the system whether to wait until a conflict has been resolved (see the *-i* option below) or to terminate the request to remove.
- *-i* — Enables secondary command registration. If T1 diagnostics are being run, this option allows the removing of another card. If -i is used and another maintenance command is being run (**"remove"**, **"detach"**, **"attach"**, **"restore"**, **diagnose**), the request to **"remove"** card is blocked and a message is printed to the screen. If *-i* is not used and any maintenance command is being run, the request to **remove** card is blocked and a message is printed to the screen.

  If the command is permitted to run, a check is made to see if the command is in conflict with another. A command is in conflict if the card or card associated with it:

  1. Is the T1 card being diagnosed
  2. Will cause a change in the existing TDM bus master assignment
  3. Has an interdependency with the T1 card being diagnosed (for example, PRI)

If one of the above conflicts exist and *-n* is not used, the user is asked whether to wait until the conflict is resolved or to terminate the request. If T1 diagnostics are executing on-line tests and a conflict is detected, the **remove card** command is blocked. If T1 diagnostics are executing off-line tests and a conflict is detected, the user is asked whether to wait until the conflict is resolved or to terminate the request to remove.

- *immed* — Removes a card or channel even if it is in use. Active calls are likely to be dropped when this option is specified. This option is necessary when the card or channel must be removed from service as soon as possible, and you are willing to terminate any active calls. You may also want to use this option to get control of a channel that is hung and not providing useful service.
- *min_delay* — Used to avoid waiting for channels to be granted. This option applies to **remove chan** and **remove card** requests that are removing network interface channels (for example, T1 and T/R). This option specifies to minimize the delay in removing channels from service by not waiting for the channel to be granted. This option speeds up execution of the **remove** command, especially when a large number of channels are currently active.

  When using this option, you must display the status of the channels with the **"display card"** command to determine when they are in the MANOOS state. This option can be used with or without the *immed* option and improves the response time in either case.

  CAUTION:
  Removing a large number of channels from service with the min_delay option may cause momentary load problems on the switch.

To delete out of the command, press DELETE. If this does not stop the command, you may need to press CTRL and backslash simultaneously. If, while running **remove**, you wish to abort the command, a message similar to the following may appear:

At the user's request, administration of the following cmd(s) has been interrupted.

CARD NUMBERS: <card numbers>

To assure proper operation of the identified card(s), run diagnostics at the earliest opportunity.

When **remove** is aborted, you should run diagnostics on all cards being administered to ensure they are returned to a fully functional state.

Example

The following example removes card 0 from service.

rem card 0

The following example removes channels 0 through 2 and channel 4 from service.

rem channel 0-2,4

The following example removes all cards from service.

rem card all

See Also

"attach"
"detach"
"restore"

remove_appl

The **remove_appl** command removes an application.

NOTE:
This command is valid only if the Enhanced File Transfer package is installed.

Synopsis

remove_appl ***[-d|s|t|f] -n <application name>***

Description

The **remove_appl** command is used to remove an application. Only one of the following options is allowed at one time.

-d This option removes the database tables.

-s This option removes the speech.

-t This option removes the transaction.

-f This option removes the installed files.

If no option is specified, the whole application is removed.

Return Values

If the **remove_appl** command is successful, a 0 value is returned.If any value other than 0 is returned, the **remove_appl** command failed. The following are the possible reasons for failure for the **remove_appl** command:

- The hard disk is low in space.
- The command syntax is incorrect.
- The voice system is not running.
- The command to remove the database tables failed.
- The command to remove the installed files failed.
- The command to remove the speech failed.
- The command to remove the transaction failed.

Example

The following example removes the application “bank_balance.”

remove_appl -n bank_balance

See Also

"backup_appl"
"install_appl"
"restore_appl"

remove_device

The **remove_device** command removes a device from the /vs/data/device_data file.

Synopsis

/vs/bin/util/remove_device

Description

The **remove_device** command is an interactive menu driven program that allows a user to remove a device entry from the **/vs/data/device_data** file. Only those devices which have been added may be deleted. The standard set of default devices with which the file is initially populated may not be removed.

Removing a device from the **device_data** file ends support of that device by the **/vs/bin/util/configure** program. The removed device is no longer available for possible configuration by the **configure** program.

Files

/vs/data/device_data

See Also

"add_device"
"change_device"
"configure"
"get_config"
"save_config"
"show_config"
"show_devices"

remove_sw

The **remove_sw** command removes an installed package.

NOTE:
This command is valid only if the Enhanced File Transfer package is installed.

Synopsis

remove_sw *<package name>*

Description

The **remove_sw** command is used to remove any of the installed software package.

The *<package name>* argument is the name that appears when a displaypky is executed and should be enclosed in double quotes (“ “).

Return Value

If the **remove_sw** command is successful, a 0 value is returned. If any value other than 0 is returned, the **remove_sw** command failed. The following are the possible reasons for failure of the **remove_sw** command:

- The hard disk is low in space.
- You are not logged in as root or super user.
- The package name is not specified.
- The package does not exist.
- The command can not find the removal script for the package.

Example

The following example removes the Intuity CONVERSANT VIS Script Builder Version 5.0 software package.

remove_sw “Intuity CONVERSANT Script Builder Version 5.0”

See Also

"install_sw"

removepkg

The **removepkg** command removes a software package.

Synopsis

removepkg

Description

The **removepkg** command removes a software package from the hard disk. You must have superuser permissions to execute the **removepkg** command.

When you enter the **removepkg** the VIS displays a list of the software packages installed on that machine and prompts you to specify the package you want to remove. Type the number associated with the software package, then press ENTER. The VIS displays messages when the software has been removed.

Example

The following example executes the program to remove a software package from the hard disk.

removepkg

See Also

"installpkg"

restore

The **restore** command restores a unit to the in-service state.

Synopsis

restore *<unit> <number> [-i] [-n]*

Description

The **restore** command is used to change the permanent state of a unit from manual-out-of-service (MANOOS) to in service (INSERV). The specified unit is placed in the INSERV state unconditionally, unless its current state is not MANOOS.

The parameters for the **restore** command are:

- *<unit>* — Identifies the unit. The choices are "channel" or "card."
- *<number>* — Specifies the channel or card number, a range of channel or card numbers in the form m n, or the word "all" for all the channel or card numbers. Card numbers are in the form ***card#[.port#]*** where *port#* is a port of card *card#*. If *port#* is not given, all ports of the card specified are restored. If no card number or channel number is given, a syntax message is displayed.
- *-n* — Disables prompting from the system whether to wait until a conflict has been resolved (see the *-i* option below) or to terminate the request to restore.
- *-i* — Enables secondary command registration.If T1 diagnostics are being run, this option allows "restoring" of another card to be performed. If *-i* is used and another maintenance command is being run (**"remove"**, **"detach"**, **"attach"**, **"restore"**, **diagnose**), the request to **"restore"** card is blocked and a message is printed to the screen. If *-i* is not used and any maintenance command is being run, the request to **restore** card is blocked and a message is printed to the screen.

  If the command is permitted to run, a check is made to see if the command is in conflict with another. A command is in conflict if the card or card associated with it:

  1. Is the T1 card being diagnosed
  2. Will cause a change in the existing TDM bus master assignment
  3. Has an interdependency with the T1 card being diagnosed (for example, PRI)

  If one of the above conflicts exist and *-n* is not used, the user is asked whether to wait until the conflict is resolved or to terminate the request. If T1 diagnostics are executing on-line tests and a conflict is

detected, the restore command is blocked. If T1 diagnostics are executing off-line tests and a conflict is detected, the user is asked whether to wait until the conflict is resolved or to terminate the request to restore.

To delete out of the command, press DELETE. If this does not stop the command, you may need to press CTRL and backslash simultaneously. If, while running restore, you wish to abort the command, a message similar to the following may appear:

At the user's request, administration of the following cmd(s) has been interrupted.

CARD NUMBERS: <card numbers>

To assure proper operation of the identified card(s), run diagnostics at the earliest opportunity.

It is recommended when **restore** is aborted, diagnostics be run on all cards being administered to ensure they are returned to a fully functional state.

Example

The following example restores card 0 to service.

restore card 0

The following example restores channels 0, 1 and 5 to service.

restore channel 0-1,5

The following example restores all cards to service.

restore card all

See Also

"attach"
"detach"
"remove"

restore_appl

The **restore_appl** command restores an application.

NOTE:
This command is valid only if the Enhanced File Transfer package is installed.

Synopsis

restore_appl -n *<application name> [-d <database file>] [-t <transaction file>] [-s <speech file>] [-p <path>]*

Description

The **restore_appl** command is used to restore an application from backed up files existing on the same machine or from backed up files sent from the host. The files are cpio files. If the file names are not specified, default file names are used and all three components (database tables, speech, transaction) are restored. The following are the default file names for each component:

database	**Dbase**
speech	**Spch**
transaction	**Trans**

The default path to restore all three components is **/tmp/sb/BkUpAppl/<*application name*>**.

NOTE:
You must use the **restore_appl** command before using the **"install_appl"** command.

Return Values

If the **restore_appl** command is successful, a 0 value is returned.If any value other than 0 is returned, the **restore_appl** command failed. The following are the possible reasons for failure of the **restore_appl** command:

- The hard disk is low on space.
- You are not logged in as root or super user.
- The command syntax is incorrect.
- The command to restore the database tables failed.
- The command to restore the speech failed.
- The command to restore the transaction failed.

Example

The following example restores the application “bank_balance” from backed up files.

```
restore_appl -n bank_balance
```

See Also

"backup_appl"
"install_appl"
"remove_appl"

retire

The **retire** command is used to retire critical and major alarm indications on the ARU.

Synopsis

retire

Description

Use the **retire** command to send appropriate control sequences to the ARU to retire any critical or major alarm indications.

Example

retire

See Also

"load_aru"

rmdb

The **rmdb** command displays the state of the resource manager (RM) and modifies the debug levels.

Synopsis

rmdb ***[-l]*** **[-s] [-u] [-d** ***[<range>]]*** **[-g** ***[<range>]*** **] [-f** ***[<range>]*** **] [-p** ***[<range>]*** **] [-C** ***[<range>]*** **] [-T** ***[<range>]*** **] [-P** ***[<range>]*** **] [-i** ***[<interval>]*** **] [-tL** ***[<levelMask>]*** **] [-tA** ***[<levelMask>]*** **] [-tc** ***<channel>*** **] [-tC** ***<channel>*** **]**

Description

The **rmdb** displays the state of the resource manager and modifies the debug levels. The valid syntax for ranges is as follows:

value [-value] [, value] | [value-value]*

Specifying a value and odd number of times indicates it will be displayed. Specifying a value an even number of times indicates it will not be displayed. For example, **7-10,9** will display the items associated with values 7,8, and 10. The 9th entry would be excluded since it was specified two times.

The **rmdb** command accepts the following arguments:

Variable	Definition
-l	Takes the rmLOCK while sampling data structures. This ensures that the sample is internally consistent. However, if the RM data structures are left in a locked state, this causes the rmdb to block until they are unlocked. (Leaving the rm data structures locked is a system fault). Also, other processes that attempt to use the RM data structures are temporarily blocked until rmdb completes its query.
-s	Prints the values of the RM parameters and debug variables
-u	Prints function's usage statistics
-d	Prints device table entries by device number
-g	Prints out group lists (by index in the group table)
-f	Prints out the function table (by index in the function table)
-p	Prints out the packfile table (by index in the packfile table)
-c	Prints out the card table (by card # in the card table)
-C	Prints out the channel table (by channel number)

Variable	Definition
-T	Prints out the channel touch-tone queues (by channel #)
-P	Prints out channel profiles (by channel #)
-i	Repeats the display, with a sleep interval of the specified number of seconds between samples.

The **-tL *<levelMask>*** sets the trace level mask. Supported masks are as follows:

Mask	Value
RM_TL_ERROR	0x1
RM_TL_GENERAL	0x2
RM_TL_ENTEREXIT	0x4

The **-tA *<levelMask>*** sets the trace area mask. Supported masks are as follows:

Mask	Value
RM_TA_TIMER	0x1
RM_TA_RESOURCE	0x2
RM_TA_INPUT	0x4
RM_TA_PROFILE	0x8
RM_TA_MTC	0x10
RM_TA_MSG	0x20
RM_TA_INTERNAL	0x40

The **-tc *<channel>*** sets the trace channel low end.

The **-tC *<channel>*** sets the trace channel high end.

rs_appl

The **rs_appl** command restores the speech or transaction component of a Script Builder application.

Synopsis

rs_appl *<a/s/t>* *<application_name>* [0/1/2/f=filename].

Variable	Definition
a	Used for restoring both the speech and transaction component of an application
s	Used for restoring the speech component of an application
t	Used for restoring the transaction component of an application
0	Indicates floppy drive 0 is to be used for restoring
1	Indicates floppy drive 1 is to be used for restoring
2	Indicates tape drive is to be used for restoring
f	Indicates information is to be restored to a file; a filename must be designated

Description

The **rs_appl** command is restore a Script Builder application from a type of media on the local machine. This command can be used to restore either a single component (for example, speech or transaction) or both of the components (speech and transaction).

The **rs_appl** command supports restoring from floppy diskettes, magnetic tapes, and to a file. Two separate sets of floppy diskettes or a set of magnetic tapes is required in order to restore both of the components of an application. Only a single component can be restored from a file.

NOTE:
When a file is used as a restore media, the file must be available either in the specified directory (file name is given with the full path name) or in the current working directory (only the filename is given) when this command is invoked.

See Also

"bk_appl"

save_config

The **save_config** command saves the **/vs/data/conf_data** to floppy disk.

Synopsis

/vs/bin/util/save_config

Description

The **save_config** command is used to save the **/vs/data/conf_data** file to floppy disk. The **/vs/data/conf_data** file is the file representing the configuration of a VIS machine as determined by the **/vs/bin/util/configure** program.

The **save_config** command should be used after upgrading a VIS machine in the field to store the newly determined configuration file to the CONFIGURATION DATA floppy for that particular machine.

Files

/vs/data/conf_data

See Also

"add_device"
"change_device"
"configure"
"get_config"
"remove_device"
"show_config"
"show_devices"

sb_backup

The **sb_backup** command backs up a Script Builder application.

Synopsis

sb_backup ***<a/d/s/t> <application_name> [<table1><table2>...<tablen>]***

Variable	Definition
a	Used for backing up all components of an application
d	Used for backing up an application database only
s	Used for backing up speech only
t	Used for backing up an application transaction only

Description

The **sb_backup** command is used to back up a Script Builder application to a media on the local machine. This command can be used to backup either a single component (for example, database, speech, or transaction) or all three components. This command supports backing up on either floppy diskettes or magnetic tapes.

NOTE:
If flloppy diskettes are used for backup, a separate set of floppy diskettes will be needed for each component of an application when you select a component or all components of an application. However, a single set of tape cartridge is enough to backup all components of an application.

See Also

"sb_restore"

sb_restore

The **sb_restore** command restores a Script Builder application.

Synopsis

sb_restore *<a/d/s/t> <application_name>*

Variable	Definition
a	Used for restoring all components of an application
d	Used for restoring an application database only
s	Used for restoring speech only
t	Used for restoring an application transaction only

Description

The **sb_restore** command is used to restore a Script Builder application from a media on the local machine. This command can be used to restore either a single component (for example, database, speech, or transaction) or all three components. This command supports restoring either from floppy diskettes or from magnetic tapes.

NOTE:
This command allows you to restore applications from any previous CONVERSANT release.

See Also

"sb_backup"

sb_te

The **sb_te** command invokes the 3270 Terminal Emulator.

Synopsis

sb_te ***<session number>***

Description

The **sb_te** command is used to invoke the 3270 terminal emulator and interact as a terminal to a host. This is used to first prove that a host communications link has been established. It can also be helpful in verifying that there have not been any changes to the host application screens. Sometimes changes can occur on the host end that are not passed down to the VIS development end. The *<session number>* chosen must be released from the host interface process before invoking **sb_te**. This can be accomplished by stopping the custom data interface process (DIP) for non-Script Builder applications or by using the **"hdelete"** command for Script Builder applications.

Sessions are mapped to logical unit (LU) numbers, with sessions numbered from 0 to 127 mapped to LUs that are numbered from 2 to 128. For example, session number 0 corresponds to the first LU number specified in the Configure Host Link screen for Link 0, while session number 1 corresponds to the second LU number in the Host Configure Link screen. LU's are configured dynamically. However, it is possible that LU's on a single connection may be non-contiguous.

A range of session numbers (for example, 5–38) can be specified to sequentially emulate each session in turn. Press CTRL Y to emulate the next session in the specified range. The CTRL Y command may only be used for multiple sessions.

If a session is not specified, the system assumes the value "all" for sessions 0–63 for both cards in a two card installation. If the first session the first card is not configured, **sb_te** automatically proceeds to the first session on the next card. For example, if session 0 on card 0 is specified and that session is not configured, a failure message is displayed and the **sb_te** command proceeds to the first session on card 1.

Example

The following example invokes the 3270 terminal emulator for card 0 and session 0.

sb_te 0

The next example invokes the 3270 terminal emulator for sessions 35–40 for card 1.

sb_te 35-40

sb_trace

The **sb_trace** command displays the trace messages and the screens being sent between Script Builder applications and the 3270 host mainframe for the specified channel.

Synopsis

sb_trace all

sb_trace *<voice channel number>*

Description

The **sb_trace** command displays on its stdout (usually the terminal screen) messages about what actions the Script Builder applications are executing on specified channels or sessions. It also captures screens sent between the 3270 host mainframe and Script Builder applications assigned to voice channels on the voice cards (T/R and T1) and/or sessions on the host interface card.

Once it starts running, **sb_trace** outputs the screens and trace messages that are generated from that time until the user terminates **sb_trace** by pressing DELETE. The **sb_trace** command outputs messages and screens when the specified voice channels or sessions are executing their Script Builder actions and/or sending or getting screens while logging-in, recovering, or in-transaction. The **sb_trace** command invokes the **trace** command when it is run.

The **sb_trace** command displays high level trace messages from the DIP and TSM on its stdout. The following is an example of the possible messages that appears:

Tracing started on channel 0
DIP0: CH 0 get screen form
DIP0: CH 0 save_bal =
TSM: CH 0 STEP: 0. VALUE: 10
TSM: CH 0 STEP: 1.
DB: Read Table
DB: index 0

The “step” refers to the corresponding action step in the transaction definition outline. “Value” refers to whatever value is given with the indicated action step.

Certain Script Builder external actions and functions may generate trace messages when they are passed invalid data or when they encounter other failures. These messages are recognizable by the fact that the “step” number is out of the range of normal action numbers that appear in the transaction definition.

If the buffer (storage area) where information is stored gets re-used before the information is completely shown on the screen, trace information may not get reported by **sb_trace**. The information you see may be incomplete. To see any missing information, place a "play message" action in the transaction to play a long silence. Insert it before the critical action whose trace you are interested in.

The **sb_trace** command accepts a voice channel as argument to output only messages and screens that relate to the specified voice channel and associated session number. For example, "sb_trace 15" limits its output to the following:

- The actions being executed from the Script Builder application on channel 15.
- If channel 15 is handling a call that interacts with the host, **sb_trace** also outputs the screen's name, fields being sent and received, and the entire screen's contents (24x80 bytes) for the "in-transaction" session associated with channel 15 to the file **/vs/trans/hostdata/*chan#***.
- If channel 15 is not handling a call that interacts with the host, **sb_trace** treats the channel number directly as a session number, and outputs the actions being executed and the screens sent and received by session number 15. Session 15 must be assigned to an application via **"hassign"** before **sb_trace** is invoked or else **sb_trace** quits without tracing. Also **sb_trace** only outputs the actions and screens while session 15 is "logging-in" or "recovering."

Note that the voice channel must exist in the system or **sb_trace** quits without tracing.

The **sb_trace** command also accepts the keyword "all" to mean that all channels and sessions will generate output. However to trace sessions that are "logging-in" or "recovering" the corresponding session numbers must fall in the range of existent voice channels. For example, a system with 25 voice channels can only trace the first 25 sessions if they're "logging-in" or "recovering". The 26th, 27th, and so on sessions can only be traced when "in-transaction" and associated with a voice channel.

The screen dumps are useful for debugging Script Builder applications while they interact with the 3270 host. A file with screen dumps is created for each voice channel or session number being traced. Screens appear in chunks of 24 text lines, appended to the files in the order which they are sent or received along with their name and time of transmission or reception. They can be viewed using the standard "cat" or "pg" commands.

The files are stored in **/vs/trans/hostdata** and are named as ***chanX*** or ***chanXX***, where *X* or *XX* is a one- or two-digit channel or session number. If they exist, sb_trace moves these files to ***chanX*.old** or ***chanXX*.old** before starting the trace.

NOTE:
These files tend to be voluminous requiring lots of disk space. If it is necessary to remove these files, it is recommended that they be removed after stopping the voice system. Otherwise if they are removed while the voice system is running **sb_trace** stops dumping the screens until the voice system is restarted.

See Also

db_pr
db_put
"trace"

sccsDaemon

The **sccsDaemon** command is the daemon process which distributes messages to the SCCS or CompuLert systems and to the alarm relay unit (ARU). It also can be used to send commands from the user to the daemon process.

Synopsis

sccsDaemon *[-v NNN]*	# Run as daemon process
sccsDaemon -c *{cmd}*	# Send command to daemon process
sccsDaemon -h	# Print help and usage message

Description

The **sccsDaemon** process is normally a continuously running daemon process which reads messages from ***${LOGROOT}*/sccsCtlPipe** where *${LOGROOT}* is normally **/usr/spool/log**. Messages are sent to this pipe by the logdaemon process and the message originating processes, which use the rules in *${LOGROOT}*/msgDst.rules to determine where to send messages. The *-v* option sets the initial verbosity level within the daemon process. This a development debugging aid only.

As a daemon process, sccsDaemon has the following responsibilities:

- Logging messages received are reformatted in accordance with the MML requirements of the SCCS/CompuLert systems and transmitted to the specified device. The format is as follows:

  031 n{machine} yy-mm-dd hh:mm:ss r 033 n{pri} {process} {msd} n 031

  The control characters embedded in the message delimit the start, end, and end-of-header locations of the message.

- If output to the console is active, a copy of the message is sent to the console as well.

  NOTE:
  This form of the message is substantially different from the VIS message format and, therefore, looks different from what would appear on the console if normal message administration was used to direct the message to the console. Use only one method to direct a message to the console.

- Every 15 minutes a “heartbeat” message is sent to the SCCS/CompuLert system, consisting only a message header. This lets the SCCS/CompuLert system know the VIS is still “alive” if there is no other message activity.

- If the alarm relay unit (ARU) is active and the priority of the message received is minor (*), major (**), or critical (*C), an alarm activation message is sent to the ARU unit to start the alarm.
- If the ARU is active and the watch dog timer is running as well, a watch dog reset command is sent to the ARU every 1 minute and 50 seconds to prevent the 2-minute watch dog timer from expiring and starting the ARU alarm.

The **sccsDaemon** command can also be used as a means to send commands to the **sccsDaemon** process. The *-c* flag indicates that it is being used to send commands to the daemon process. The commands are:

- check_console_flag — This command rereads the state of the console output flag. All output being sent to the SCCS/CompuLert destinations can be duplicated to **/dev/console** as well. The commands **"console_off"** and **"console_on"** use this command to inform the daemon that the state of such output has changed.
- check_devices — This command rereads the destinations for output to the SCCS/CompuLert systems and for the ARU. The command **"assign_tty"** uses this command to alert the daemon process that the destinations have been altered.
- quit — This command causes the daemon process to exit gracefully.
- verbosity {value} — This command changes the verbosity level of the daemon process. There are some messages that are only logged when the verbosity level is increased. This is a development debugging aid only.

The **sccsDaemon** process takes option settings from **/vs/etc/default/sccsDaemon**. The following parameters are understood:

CONSOLE_OUTPUT_DEFINED
: If this parameter is set, messages are sent to the **sccsDaemon** can be sent to **/dev/console** in addition to the SCCS device. The default is FALSE.

CONSOLE_FLAG
: This parameter enables output to console if CONSOLE_OUTPUT_DEFINED. Its value can be changed during operation via the **"console_on"** and **"console_off"** commands. The default is FALSE.

CONSOLE_DEVICE
: This parameter specifies the name of the console device. The default is **/dev/console**.

ARU_ENABLED
: This parameter enables the sending of alarm messages to the ARU. The default is TRUE.

SCCS_DEVICE
: This parameter specifies the alternate device name for the SCCS. This parameter is only used if the file **/vs/data/Sccs_tty** is not set. The default is <NONE>.

ARU_DEVICE
: This parameter specifies the alternate device name for the ARU. This parameter is only used if the file **/vs/data/Aru_tty** is not set. The default is <NONE>.

SCCS_PIPE
: This parameter specifies the name of the **sccsDaemon** pipe. The default is **/usr/spool/log/sccsCtlPipe**.

MACHINE_NAME
: This parameter specifies the alternate machine name. This parameter is only used if the file **/vs/data/Machname** is not set. The default is <NONE>.

VERBOSITY
: This parameter sets the initial value of the internal verbosity flag for the **sccsDaemon**. The default is 0.

SCCS_MODES
: This parameter uses the stty command to condition the SCCS device. The default is **stty sane 9600 "erase" b echoe echok**.

ARU_MODES
: This parameter uses the stty command to condition the ARU device. The default is **stty sane 9600 erase b echoe echok**.

Files

```
/vs/bin/vrs/sccsDaemon      # Executable program
/vs/data/Sccs_tty           # Contains SCCS/CompuLert device
/vs/data/Aru_tty            # Contains ARU device
/vs/data/Machname           # SCCS/CompuLert name of system
/vs/data/console_stat       # Current status of output to console
/vs/data/wdog_stat          # Current status of ARU watch dog timer
/usr/spool/log/sccsCtlPipe  # Name of daemon pipe
```

See Also

See *Intuity CONVERSANT VIS Version 5.0 Communications Development*, 585-310-229, for additional information about the following commands.

"assign_tty"
"chg_machname"
"console_off"
"console_on"
"wdog_off"
"wdog_on"

show_config

The **show_config** command displays and prints to file the valid Intuity CONVERSANT VIS system configuration represented by the /vs/data/conf_data file or the incomplete configuration represented by the /vs/data/fail_data file.

Synopsis

/vs/bin/util/show_config ***[fail/filename]***

Description

The output of the **/vs/bin/util/configure** program is either the **/vs/data/conf_data** file or the **/vs/data/fail_data** file depending respectively on whether the program arrives at a complete and valid configuration, or an incomplete, conflicting configuration based on the user's input. In either case, the data in these files is compressed and cannot be easily understood. The **/vs/bin/util/show_config** command formats the data in these files and displays it in tabular form to the screen and also writes it to a file in the current directory.

A message from the **configure** program instructs the user whether to use the *fail* option with **show_config** or not.

If the **configure** program was successful, executing **show_config** with no argument creates a **./configuration** file, by expanding the contents of **/vs/data/conf_data**. This file can then be printed for hard copy of the successful configuration.

If the **configure** program was unsuccessful at determining a configuration, executing **show_config fail** creates a **/vs/data/failed_config** file by expanding the contents of **/vs/data/fail_conf**. The **/vs/data/failed_config** file may be examined to determine what conflicting resource caused the configuration to fail.

The **show_config** command always checks for the presence of **./configuration** or **/vs/data/failed_config** and asks the user whether it is acceptable to overwrite the current file by that name if it exists.

When the **configure** program is used to upgrade an existing machine, the current **/vs/data/conf_data** file is saved in **/vs/data/conf_*MMDDYY***, where *MM* = month, *DD* = day and *YY* = year. It may be desirable at times to see what configuration is represented by these saved configuration files. The **show_config** command may be used to expand the contents of a saved configuration file by specifying the filename as the first argument. The user is prompted for an output file name whenever the first argument is an input configuration filename.

Files

/vs/data/conf_data
/configuration (show sample output, describe each HEADING)
/vs/data/fail_data
/vs/data/failed_config

See Also

"add_device"
"change_device"
"configure"
"get_config"
"remove_device"
"save_config"
"show_devices"

NOTE:
The **show_config** command takes only one argument. The *fail* and *filename* arguments are mutually exclusive.

show_devices

The **show_devices** command displays and prints to file all devices and their attributes as represented in the **/vs/data/device_data** file.

Synopsis

/vs/bin/utilshow_devices

Description

The **/vs/bin/util/show_devices** command uncompresses the database of devices and their attributes contained in the **/vs/data/device_data** file and displays the information to the screen. At the same time, a **./devices** file is created so that hard copy of this information may be generated. If a **./devices** file already exists, the user is prompted as to whether it is acceptable to overwrite the file.

Files

/vs/data/device_data
./devices (show sample output)

See Also

"add_device"
"change_device"
"configure"
"get_config"
"remove_device"
"save_config"
"show_config"

show_sys

The **show_sys** command allows you to retrieve configuration and administration information from customer sites.

Synopsis

/vs/bin/tools/show_sys ***[-l]***

Description

The following information can be retrieved with the **show_sys** command:

- UNIX version machine type
- Installed software
- Memory
- Configuration of hard disk(s)
- Free space in UNIX file system
- Tunable parameter changes
- Free space in swap
- Free space in speech file system
- Free space in Oracle database
- Oracle database tables
- Directory files in **/oracle/dbs**
- Cron information for root
- Local/remote database information
- SP driver (speech card) version
- DNIS information (if T1s are present)
- T1 card information (if T1s are present)
- Device Information
- SAR Snapshot
- Parallel Printer Information
- UUCP information
- Devices file
- Permissions file
- Systems file
- Analog transfer parameters

- Installed CONVERSANT cards
- Parameter file(s) for assigned applications
- Databases used in each application
- Status of Host LU's
- CCA report for the previous week
- Call data report for a specific day of the previous week
- Traffic report for a specific day of the previous week

The *-l* option prints details about each of the information that can be retrieved with the **"show_sys"** command.

Example

"show_sys"

soft_disc

The **soft_disc** command sends a disconnect to a script on a channel or channels.

Synopsis

soft_disc *<channel>*

soft_disc *<channelStart-channelEnd>*

Description

The **soft_disc** command sends a message or messages to TSM requesting that the script running on *<channel>* or the range of channels *<channelStart-channelEnd>* be sent interrupt messages. If no script is running on the channel or if TSM does not own the channel, no action is taken for the channel.

The **soft_disc** command waits for a response from TSM. When it exits, TSM has acted on all the requests for all the channels by sending disconnects to the scripts or rejecting the requests. Scripts running on the channel receive the ESOFTDISC event.

Return Values

If the **soft_disc** is successful, a 0 value is returned. If any other value than 0 is returned, the **soft_disc** command completely or partially failed. If **soft_disc** returns a value of 2, then **"dip_int"** command failed due to temporary condition. In this case, the user should attempt the **"dip_int"** command again.

Example

The following example requests that TSM send interrupt messages to channel 2.

soft_disc 2

The following example requests that TSM send interrupt messages to channels 1 through 32.

soft_disc 1-32

See Also

"dip_int"

soft_szr

The **soft_szr** command starts a script on a channel.

Synopsis

soft_szr *<channelStart-channelEnd> <script>*

Description

The **soft_szr** command can be used to start a script on a channel. The **soft_szr** command sends a message to TSM requesting that a script be started on a channel. If the channel is in use, the script is not started. **Soft_szr** waits for a response from TSM. When **soft_szr** exits, TSM has either accepted the request and started the script or rejected the request.

There are two arguments to the **soft_szr** command: *<channel>* and *<script>*. The *<channel>* argument specifies the channel or range of channels on which you want to start the script. The *<script>* argument specifies the script to be started. The script does not have to be in the table of assigned scripts.

The channel number(s) must be valid and the channel(s) must not be busy, and the channel(s) must be in the inserv state. If you specify a channel that is busy, the command fails. If you specify a range of channels and one or more of the channels is busy, the command seizes the idle channels but fails for the busy channels.

Example

The following example starts the script “sodapop” on channels 0 through 4.

soft_szr 0-4 sodapop

The following example starts the script “test1”’ on channel 10.

soft_szr 10 test1

Return Values

If the **soft_szr** is successful, a 0 value is returned. If any value other than 0 is returned, the **soft_szr** command completely or partially failed. If **soft_szr** returns a value of 2, then **soft_szr** command failed due to temporary condition. In this case, the user should attempt the **"dip_int"** command again.

spCtlFlags

The **spCtlFlags** command sets and clears flags used to control the behavior on SP Executive pack files as they run on an SP card.

Synopsis

spCtlFlags ***[-b SP-index] [-t] [[+|-]flag]***

Description

The CTL flags provide a means to alter the behavior of code running on the SP from the PC without distracting it from the job at hand. At the current time the CTL flags integer is divided into three parts, the upper 16 bits, which are general purpose flags to be used to turn on and off code and printfs, the bottom 8 bits, which are reserved for the SP Executive functions, and bits 8 15, which are currently not used by anyone officially. There is an unofficial use of these bits to prime the verbosity level for layer 3 of PRI.

The following are the options that can be used with the spCtlFlags command:

-b SP-index	Index of the SP card to be examined
-t	Terse - only output hex value of flag

NOTE:
The **spCtlFlags** only works with SP executive applications (currently, the PRI and CCA pack files).

With no flag argument, **spCtlFlags** just prints the current value. With a flag argument, it either resets the value (no '+' or '-'), logically ORs in the flag ('+'), or logically and compliments out the flag ('-'). A flag can either be a number or use one of the following symbolic names:

printf	This flag controls whether printfs from within an SP card actually generate output or not.
letters	This flag contains executive trace flag of letters arriving from the PC.
terminations	This flag generates reports on all process and action terminations.
dbgpanics	If this flag is set, panics by SP executive go to debugging monitor. If not set, panics go immediately to ROM for reloading.
timefcns	This flag enables timing of TDM and DSP functions.
checkmem	This flag enables checking of the "malloc" arenas to insure that they have not been corrupted. (This is fairly expensive in terms of CPU cycles expended per allocation reference.)
enabledbg	This flag enables various general purpose debugging code if it is compiled into the executive.
dbg{1-16}	This is a general purpose flag that can be used for debugging.

Symbolic and numerical flags can be combined with the "+" sign between them, that is, "+dbg1+printf" or "-0x20+printf."

Notice that the current value of the flags is printed if no other arguments are specified and that by starting the flags with a '+' causes them to be added to those already in place rather than just replacing the current flags with the new ones. The following is additional information for each of the symbolic names:

printf	If this flag is not set, all **printf()** operations from within the SP Executive are essentially NOPs. This flag must be set for any print information to be sent to the PC and logged.
letters	If this flag is on, the SP Executive attempts to report, via **printf()** the arrival of each letter that it is processing from the PC.
terminations	If this flag is on, the SP Executive sends a termination letter to the SP whenever a process or an action completes. This, in turn, is logged.
dbgpanics	If this flag is set and the SP Executive calls the **panic()** routine, it stops and waits for a debugger to examine what has happened. If this flag is not set and **panic()** is called, the SP Executive returns immediately to the ROM for reloading.
timefcns	If this flag is set, the SP Executive starts timing operations on each of the following four things, TDM interrupt servicing, the length of time between TDM interrupts, the length of time DSP loading is taking, and the length of time DSP servicing is taking. This information is requestable in the future via a letter from the PC. Currently it must be examined via a debugger.
checkmem	If this flag is set, each attempt to **malloc()**, **realloc()**, or **free()** memory causes the malloc arena to be checked for consistency. If the define symbol SM_FULLCHECK is set when the **spaceMngr.c** file is compiled, this check is very complete (though more time consuming) and detects problems sooner. If it is compiled without SM_FULLCHECK, the check is more cursory in nature.
enabledbg	Much of the special history keeping code is conditional upon this flag being set. If it is not set, the overhead of saving and timing is avoided. If it is set, then whatever history mechanism has been compiled in, saves its form of history information for debugging purposes.
dbg[1-16]	The use of these flags is up to each task. It is assumed that they will be used during debugging phases, but not be in use for final distribution. Code using them does tests of the following form: **if (spcon->status[SPS_CTL_FLAGS] & SPCF_DBGnn)** to determine whether a certain section of code or not should be executed.

spres

The **spres** command restores speech from a backup.

Synopsis

spres *-l <file> [-v] -t [talkfile <list>] [phrase <list>] [listfile <list>]*

Description

The **spres** command restores the specified talkfile number, phrase number, listfile, or phrase and talkfile of the speech. Only speech that is backed up using the **"spsav"** command can be restored with the **spres** command.

The parameters for the **spres** command are as follows:

-l file This parameter specifies the input device. Typically, this is cartridge tape (**/dev/rmt/c0s0**).

-v This parameter is the verbose flag that gives a running commentary of the "restore" procedure.

-t This parameter is the tape flag. This is required for restore from cartridge tape.

[talkfile <list>] This parameter specifies the list of talkfiles to be restored, specified as a single digit, a range m-n, or the word **all**. If no value is given, the default is **all**.

[phrase <list>] This parameter specifies the list of phrases to be restored, specified as a single digit, a range m-n, or the word **all**. If no value is given, the default is **all**.

[listfile <list>] This parameter specifies the list of listfiles and associated speech to be restored (for example, **listfile list.cabnt**)

The **spres** command invokes an interactive program asking you to insert and remove cartridge tapes periodically. If the *-v* option is used, the system displays information about each step of the recovery.

Example

The following example restores listfile “list.cabnt” verbosely from cartridge tape:

spres -l /dev/rmt/c0s0 -v -t listfile list.cabnt

spsav

The **spsav** command backs up speech.

Synopsis

spsav ***-O <file> [-v] -t [talkfile <list>] [phrase <list>] [listfile <list>]***

Description

The **spsav** command backs up the specified talkfile number, phrase number, listfile, or phrase and talkfile of the speech. Only speech in the speech file system can be backed up using the **spsav** command.

The parameters for the **spsav** command are as follows:

-O file This parameter specifies the output device. Typically, this is cartridge tape (**/dev/rmt/c0s0**).

-v This parameter is the verbose flag that gives a running commentary of speech being saved.

-t This parameter is the tape flag. This is required for back up to cartridge tape.

[talkfile <list>] This parameter specifies the list of talkfiles to be backed up, specified as a single digit, a range m-n, or the word **all**. If no value is given, the default is **all**.

[phrase <list>] This parameter specifies the list of phrases to be backed up, specified as a single digit, a range m-n, or the word **all**. If no value is given, the default is **all**.

[listfile <list>] This parameter specifies the list of listfiles and associated speech to be backed up (for example, **listfile list.cabnt**)

The **spsav** command invokes an interactive program asking you to insert and remove cartridge tapes periodically. If the *-v* option is used, the system displays information about each step of the back up.

Example

The following example saves listfile “list.cabnt” from cartridge tape:

spsav -O /dev/rmt/c0s0 -v -t listfile list.cabnt

spStatus

The **spStatus** command displays information about the pack file running on an SP card.

Synopsis

spStatus ***[-b SP-index] [-i interval] [-c count] [-r] [-B]***

Description

A substantial amount of information about the state of an SP Executive pack (PRI and CCA pack files) is available via shared memory and the program **spStatus**, which displays the information. The information is defined in **include/spStatus.h**.

The following are the options that can be used with the **spStatus** command:

-b SP-index	Index of the SP card to be examined
-i interval	Interval between examinations of SP status Minimum: 2 seconds. Default: 60 seconds.
-c count	Number of times SP status is to be examined. Default: 1
--r	Reset the executive and task counts before starting.
-B	No bell when running in iterative mode.

The **spStatus** command can be run in a one-shot mode, which is the default, or an iterative mode. In the iterative mode, it prints the changes between each successive examination of the values stored in the **spcon** structure in shared memory.

Sample Format

The following is an example of the sample output if **"spStatus"** is against the CCA pack.

```
        Fri Dec 7 13:06:03 1990
Romstate: 0x0 Romcmd: 0x0 Romargs: 0x0 0x0
Ramstate: 0x245 Pack Features: C      Pack Type: SP executive
 Bootcnt: 0x0 SPtime: 0x6a5 SPusage: 0x0
Debug ID: 0 spFreeMemory: 1,164,152
<< Status Information >>
          Free Actions: 46
          Busy Actions: 4
        Active Letters: 4
            Free DSPs: 2
          Broken DSPs: 0
            Busy DSPs: 0
      Run Queue Length: 0
    Sleep Queue Length: 4
    Running Process ID: 5
   Running Action Index: 3
         DSP Requests: 4
     RPC Requests Done: 0
   RPC Requests Queued: 0
   RPC Requests Discard: 0
          Exception #: 0x0 0-Reset
        Exception Adr: 0x0
              Routine: 0x0
    PC at last TDM Intr: 0x99f05b62
    PC at last DSP Intr: 0x99f05b46
            DSP Count: 2247
            CTL Flags: 0x0
       Timer Requests: 0
        Active Timers: 0
      Completed Timers: 0
        Killed Timers: 0
Work Search Loops * 1000: 53
          TDM Overruns: 0
 TDM Servicings Deferred: 0
      Letters Received: 8
          Letters Sent: 0
      Letters Deferred: 0
     Letters Discarded: 0
        Executive Time: 57
            Idle Time: 489
Task[6]: 1156
<< Mailbox Information >>
Index   1st Empty   1st Full
0 -> PC     00
```

```
1 <- PC    00
2 <- PC    4040
3 <- PC    00
4 <- PC    00
5 <- PC    00
6 <- PC    00
7 <- PC    00
8 -> PC    00
==========================================
1      Fri Dec 7 13:06:09 1990
 Bootcnt: 0x0 SPtime: 0x7df SPusage: 0x0
<< Status Information >>
       Run Queue Length: 1(+1)
     Sleep Queue Length: 3(-1)
     Running Process ID: 4(-1)
   Running Action Index: 2(-1)
    PC at last TDM Intr: 0x99f14024
    PC at last DSP Intr: 0x99f08154
           DSP Count: 2710(+463)
Work Search Loops * 1000: 57(+4)
        Executive Time: 58(+1)
            Idle Time: 537(+48)
Task[6]: 1421
```

The following is a brief description of each element of the display:

Romstate, Romcmd, Romargs
These three values are active if either the ROM is in control of the SP card or a debugger is in charge.

Ramstate, Pack Features, Pack Type
If a packfile is running or being debugged Ramstate contains the ID of the pack. If the pack is an SP Executive type pack, the Pack Features indicate which tasks are available in this pack. The Pack Type is either “SP executive” or “Original.”

Bootcnt, SPtime, SPusage
Bootcnt is incremented each time the ROM restarts. Only diagnostics currently alter it in any other way. SPtime is the time in 16 msec increments since the pack started. If **"spStatus"** is running in recursive mode, this value is not changing, and the debugger is not active, the following warning is generated:

```
“No Clock! Check TDM master,”
```

One of two things is happening, either there is no TDM master and hence no TDM interrupts, or the pack file is stuck at priority level 6 or 7 and so all interrupts are blocked. In the former situation, check your T1 or Tip/Ring (T/R) cards and make sure that one of them is the TDM master. In the latter case, you have a bug. Use msdb and examine the pack file. **SPusage** is the current load factor on the SP card. This is the last value of meaning if the pack is an original-style pack. The remaining information applies only to SP Executive packs.

Debug ID, spFreeMemory
Debug ID is set to the pid of the UNIX process currently debugging this SP card. It is used to avoid collisions between people attempting to debug code running on a card. spFreeMemory is the amount of memory free in the memory allocation arenas, which are managed by malloc(), realloc(), and free().

Free Actions
The number of Action structures not currently assigned to a time slot. This value is initially 50.

Busy Actions
The number of **Action** structures currently assigned to time slots.

Active Letters
The number of letters being carried in Chainmail structures for long time processing via **Action** structures.

Free DSPs, Broken DSPs, Busy DSPs
The number of DSP processors available to do work, broken, and assigned to work.

Run Queue Length, Sleep Queue Length
The number of processes waiting to run and the number waiting for some event to wake them up.

Running Process ID, Running Action Index
The process ID of the SP Executive process currently running and the index of the Action structure currently active.

DSP Requests
The number of DspRequest structures active.

RPC Requests Done, RPC Requests Queued, RPC Requests Discard
The number of remote procedure call requests that have been performed, the number that are waiting to be done, and the number of requests that had to be discarded before the backlog was too large.

Exception #, Exception Adr
The 680X0 hardware exception number and the name of the exception that has stopped the 680X0 processor and either sent it to the ROM or to the debugger and the address where the exception took place. These can be very valuable in case of a fatal error.

Routine
Currently not used.

PC at last TDM Intr, PC at last DSP Intr
Addresses at which the TDM and DSP last interrupted.

Info Flags
Currently there are two pieces of information conveyed by these flags, whether the processor is currently within a DSP interrupt and whether it is within a TDM interrupt. Both, neither, or any combination could be true.

DSP Count
The number of DSP interrupts processed.

CTL Flags
The current value of the CTL flags. These are used to control optional code within a pack. See "spCtlFlags" for further information.

Timer Requests
The number of timer requests that have been made.

Active Timers, Completed Timers, Killed Timers
The number of timer requests currently outstanding, the total number of timer requests that have run to completion, the number of timer requests that were removed prior to execution. If these values do not total up properly, there is also a warning indicating that there is trouble.

Work Search Loops
This is the number of times divided by 1000 that the SP Executive has gone through its base level work search loop, trying to find something productive to do. The change in the number goes down as the SP Executive becomes busier and busier doing productive work.

TDM Overruns
This number should always be zero. If it is not, it indicates that some activity is taking too long and blocking the processing of a TDM interrupt before it rolls over and starts overwriting data. This is serious.

TDM Servicings Deferred
This is the number of times that a TDM servicing was deferred because the TDM interrupt came in on top of a DSP interrupt for a time slot. It is not serious. It just indicates that the hardware is busy and conflicts are being resolved. It can be a potential area of difficulty if the DSP routine is too slow and the TDM overruns while it is waiting to be serviced.

Letters Received, Letters Sent, Letters Deferred, Letters Discarded
This is the number of letters received from the PC, the number of letters sent to the PC, the number of letters going to the PC that had to be temporarily stored in the overflow area because the PC was not keeping up, and the number of letters that even the overflow area could not handle and had to be discarded. Going into the overflow area is an indication of potential trouble, but is not bad if the duration is short. If the SP code continues to generate too many letters in too short of a period of time, then it is real trouble. The same thing can happen if the PC gets bogged down and cannot keep up.

Executive Time, Idle Time, Task[]
These counts indicate the load being placed on each portion of the system. The executive time is the number of times the TDM interrupted some activity of the SP Executive that was what is considered to be the idle look-for-work activity. The idle time is the number of times the TDM interrupted the look-for-work activity. When tasks are active, a line appears for each task. The index of the task is its position in the **tasks[]** array found in the associated **sp/config/taskTbl*.c** file.

Mailbox Information — Index, Empty, Full
The mailbox information is rudimentary information about activity within each mailbox. It does not tell you how many letters have been sent via each mailbox, though that may come eventually, but it does tell you whether the mailbox is empty (1st Empty == 1st Full) and if the values are changing from one display to the next, you know mail is passing through that mailbox. Keep in mind that mailbox 1, from the PC, is now reserved by the kernel and is used by the ioctl() form of mail sending for all processes other than the limited number of processes that directly own mailboxes.

In iterative mode, only those lines whose values have changed since the list display are listed. On decimal entries, the delta value since the last time is also printed.

spVrsion

The **spVrsion** command prints the version of the SP driver currently installed on a machine.

Synopsis

spVrsion

Description

The **spVrsion** command prints which version of the SP driver has been installed. The two versions that can be installed are the 12-Mbyte version and the 44-Mbyte version.

start_hi

The **start_hi** command starts the 3270 host interface software.

Synopsis

start_hi

Description

The **start_hi** command starts the 3270 host interface software appropriately for voice system use.

Example

The following example starts the 3270 host interface software:

start_hi

See Also

"stop_hi"

start_vs

The **start_vs** command brings the system up to a fully operational state.

Synopsis

start_vs

Description

The **start_vs** command returns the voice system software to fully operational state. If you use the **"stop_vs"** command to stop the system, you should use the start_vs command to start it again. The **start_vs** also should be used if the system was rebooted or powered down after **"stop_vs"** was used.

The **start_vs** command checks to see if the user stopped the system with the **"stop_vs"** command. The **start_vs** command places all cards placed in the manual-out-of-service (MANOOS) state with the **"stop_vs"** command in the in-service (INSERV) state.

You must be logged on to the system console as root to use the **start_vs** command.

Since the **/vs/data/spchconfig** file cannot be edited while the voice system processes are running, it is a good idea to check the value of nbufs in the **/vs/data/spchconfig** file before executing the **start_vs** command. The value of nbufs defines the number of speech buffers. In order for the voice system to operate properly, nbufs must be set to 2.5 times the number of active channels.

Example

The following example starts the voice system software.

start_vs

See Also

"stop_vs"

stop_hi

The **stop_hi** command stops the 3270 host interface software.

Synopsis

stop_hi

Description

The **stop_hi** command stops the 3270 host interface software.

Example

The following example stops the 3270 host interface software:

stop_hi

See Also

"start_hi"

stop_vs

The **stop_vs** command command gracefully stops the voice system software.

Synopsis

stop_vs ***[time_out] [-n]***

Description

The **stop_vs** command gracefully stops the voice system software. If the system is receiving calls, **stop_vs** waits for approximately three minutes before it unconditionally stops the software. By waiting, the system allows callers to finish their transactions. The **stop_vs** command disables incoming call recognition on all cards to prevent them from being reactivated by an incoming call.

The *time_out* option is the time to wait before the voice system is stopped. The default value for this option is 180 seconds. The *-n* option prompts you with a message that another maintenance command ("restore", **"remove"**, **"attach"**, **"detach"**, **diagnose**) is being performed. It asks you if you wish to continue or to terminate the **stop_vs** command. The **stop_vs** command terminates another maintenance command in progress when initiated. The default value for this option is Yes.

If you use **stop_vs** to stop the system, you should use **start_vs** to reactivate it. If you use **stop_vs** to stop the software and then reboot the machine, be sure to execute **start_vs** after logging in as root. This ensures that the system is returned to the state it was in before it was rebooted.

If an active host link is established, the **stop_vs** command checks the LUs and logs out the application(s). The command waits up to 60 seconds (6 series of the 10 seconds each), then continues stopping the voice system.

Example

The following example stops the voice system software.

stop_vs

See Also

start_vs

striphdr

The **striphdr** command strips voice or code headers from a speech file.

Synopsis

striphdr ***[voice/code]***

Description

Striphdr is a filter that removes either the voice or code headers from a speech file. Voice headers are required for files being edited by the GSE, and code headers are required for speech that is to be used with the VIS.

See Also

"addhdr"

sysmon

The **sysmon** command executes a program that monitors incoming telephone lines and the associated cards to see that they are functional.

Synopsis

sysmon *<page number>*

Description

The **sysmon** command verifies that each incoming telephone line and its associated card are functional. Before initializing the test, locate a touch-tone telephone close to the system controller and get a telephone number to be used for dialing into the system. Use the **assign channel** command to assign to a group any channels you want to test. Then, use the **"assign service/startup"** command to assign a script to the same group.

Once the channels and service are assigned, enter the sysmon command followed by the number of pages, or screens, you want to see. Each page displays 120–140 channels.

The resulting display shows all channels and their current states. Note that only equipped channels can be in the IDLE or MOOS state, while unequipped channels are followed by dashes (--).

Enter the telephone number for the touch-tone phone. Watch the display on the monitor and note the channels that receives the call. Follow the instructions provided by the voice system. Enter 0000 to end the test.

Example

The following example shows page four of the system monitor display.

sysmon 4

tas

The **tas** command executes the transaction assembler (tas) program to assemble script instructions.

Synopsis

tas ***[-e] [-I<include_directory> -T<talk_directory> -U<name> -D<name> -D<name_def> -Y<dir> -H] -o<output_file> <application_name>*.t**

Description

The **tas** command is used to assemble script instructions recorded in an ***application-name*.t file**. It produces an executable file designated ***application-name*.T**, which is stored in a table as a list of executable script instructions.

The *-e* option requires exact string matches for speech phrases.

The arguments must be in the order given above for the command to work properly. The directory search specified by the arguments are: *I* (include file) and *T* (listfile). No space is allowed between the *-I* and *-T* flags and their pathnames, but space is allowed after the *-e* flag. Note that the *-I* option to **tas** is interpreted by cpp(1).

The remaining arguments are:

- *-U <name>* — Remove any initial definition of name, where name is a reserved symbol that is predefined by the particular preprocessor (this option is interpreted by cpp(1)).
- *-D <name>* and *-D <name-def>* — Define *name* with value *def* as if by a #define. If no *-def* is given, *name* is defined with value 1. The *-D* option has lower precedence than the *-U* option. That is, if the same name is used in both a *-U* option and a *-D* option, the name is undefined regardless of the order of the options (this option is interpreted by cpp(1)).
- *-Y <dir>* — Use directory *dir* in place of the standard list of directories when searching for #include files (this option is interpreted by cpp(1)).
- *-H* — Print, one per line on standard error, the path names of included files (this option is interpreted by cpp(1)).
- *-o <output_file>* — The name of the output file. The default is **out.T**.

Note that the maximum number of literals per script allowed by the **tas** command is 450. If there are more than 450 literals in a script, the error message "literal table overflow" is displayed. Additional limitations enforced by the **tas** command are (whichever occurs first in a list file):

— 1,000 phrases

— 4,000 words

— 40,000 characters

If more phrases are needed by an application, use multiple list files and tfile instructions within the script.

NOTE:
If your script contains a large number of define statements, **tas** may report messages such as the following during compilation:

```
script.t: 1068: too much defining
```

where *script.t* is the script source file and *1068* is the line in which the define appears. The limit to the number of define statements that a script may have depends on the number of defined macros and their size. If this type of message appears, reduce the number of define statements in your script.

Files

/vs/bin/tas

Example

tas example.t

The program includes applicable header files and replaces literal definitions with corresponding numbers to produce an assembled version of the script. The assembled code is stored on disk under the label **example.T**. The unassembled instructions are found in the file **/var/applN/trans/example.t**.

tas example.t -I/var/include -T/var/speech

In addition to performing the same functions described for the previous example, tas checks the files in **/var/include** when processing include statements and the file in **/var/speech** when processing T-file statements.

trace

The **trace** command outputs trace messages to standard output, while the system is taking calls, for specified processes and channels.

NOTE:
This information may be useful for debugging applications and dips.

Synopsis

trace ***[name]...[chan <,range>]...[card <card #[.port#>]]...[area [,area...]...] [level [,level...]...][date] [tracelog | startlog]...[sleep <sleeptime>]***

Description

The **trace** command prints trace messages to the standard output device (stdout) according to specified options. Executing trace also causes trace output to be logged to the trace shared memory buffer or to the trace log.

When trace is specified with ***name***, all process-specific trace messages from process ***name***, are printed. Process-specific trace messages are printed regardless of which channels that process may own or on which are operating.

When trace is specified with **chan** or **card** options, all channel-specific messages, from any process are printed. The **card** option is applicable only to network interface cards (that is, cards that have channels). The **card** option is a special case of the channel option.

A combination of the *name* variable and **chan** options prints trace messages from both the *name* and **chan** options. The *name* and **chan** options act collectively rather than selectively.

If **area** is specified, only the process or channel messages associated with **area** are printed. The **area** option is, therefore, selective. Areas may be integers ranging from 1 to 32. Areas 1 through 16 are available for user applications. The voice system reserves areas 17 through 32. The current internal voice system areas in use (areas 17 through 32) may be identified through mnemonics defined as follows:

AS (area 17)	Trace advanced service operations such as TTS and speech recognition
EM (area 18)	Trace event management operations
IN (area 19)	Trace caller input operations including touchtone and speech recognition
PM (area 20)	Trace parameter management operations

RM (area 21)	Trace resource management operations
SE (area 22)	Trace script execution. This includes trace entries made implicitly by Script Builder applications and through *tas(1)* scripts via the *trace(3TSM)* command.
ST (area 23)	Trace call and application initialization and completion operations
TS (area 24)	Trace telephony service operations
VS (area 25)	Trace voice code and play operations
ER (area 26)	Trace error processing operations
IL (area 27)	Trace internal library operations
SI (area 28)	Trace script instructions. Every TSM script instruction displays a trace message.
AD (area 29)	Trace administration operations
BM (area 30)	Trace bus management operations
OT (area 32)	Trace old trace instructions. All old trace messages are placed in this area
ALL (area 1-32)	Trace all areas

The default, if **area** is omitted, is **all** areas except **SI** (area 28). Trace areas may also be specified numerically with lists and ranges. For example, the following is legal:

```
trace chan 5 area 1-7,10,TS
```

A **level** argument may also be specified. Levels range from 1 through 32, where level 1 indicates the least amount of detail and level 32 indicates the greatest level of detail. Levels may be specified as a single number, comma-separated list, or ranges. The current internal voice system levels in use (levels 17 through 32) may be identified through mnemonics. A complete list of area and level mnemonics can be displayed by executing the **trace** command with no arguments. The current voice system levels (areas 1 through 32) are defined as follows:

U (levels 1-16)	Trace all user levels
AE (level 17)	Trace internal application error messages
AG (level 18)	Trace internal application general messages
AX (level 19)	Trace internal application enter/exit messages
A (levels 17-19)	Trace all internal application levels
FE (level 20)	Trace user-callable function error messages
FG (level 21)	Trace user-callable function general messages
FX (level 22)	Trace user-callable function enter/exit messages
F (levels 20-22)	Trace all user-callable function levels

PE (level 23)	Trace process interface function error messages
PG (level 24)	Trace process interface function general messages
PX (level 25)	Trace process interface function enter/exit messages
P (levels 23-25)	Trace all process interface function levels
IE (level 26)	Trace error processing operations
IG (level 27)	Trace internal library operations
IX (level 28)	Trace script instructions. Every TSM script instruction displays a trace message.
I (levels 26-28)	Trace script instructions. Every TSM script instruction displays.
RH (level 29)	Trace RM Helper function enter/exit messages
RE (level 30)	Trace RM Helper function error messages
RG (level 31)	Trace RM Helper function general messages
RX (level 32)	Trace RM function enter/exit messages
R (levels 29-32)	Trace all RM Helper and RM function messages
S (level 17-32)	Trace all irAPI system levels
ALL (levels 1-32)	Trace all levels

The default, if level is omitted, is levels **U**, **A**, **AE**, **FE**, **PE**, **IE**, and **RE**. Trace levels may also be specified numerically with lists and ranges.

If the **tracelog** option is specified, all trace messages are logged to the trace log file and sent to *stdout*. If **startlog** is specified, tracing is done to the trace log but no trace output is sent to *stdout*. The trace log file may be queried for data deposited from prior executions of the **trace** command by using the **display** command with the tracelog option.

Trace messages may be printed with or without the date and time when they are generated. If **date** is specified, the date and time are printed with each trace message. The date and time are always printed for messages in the trace log file.

If the **sleep** argument is specified, trace will sleep *sleeptime* milliseconds between reading the trace buffer. The default is 200ms.

The **trace stop** command clears any active trace settings, ensuring that no trace output is generated to the trace log.

By default, all trace messages are saved in a trace shared memory buffer. The trace buffer is a circular buffer. If trace messages are written to the trace buffer faster than the trace command can read them, eventually the trace buffer will overflow and trace messages will be lost. When this happens, trace will print the message `TRACE: ***** LOST XXX RECORDS`, where XXX is the number of trace messages lost. Two ways to minimize the number of trace messages lost exists:

- Use the **sleep** argument of the trace command to decrease the time that trace sleeps between reading the buffer (default sleep time = 200 ms)
- Increase the size of the trace buffer by adding or modifying the line `TRACE_BUFFER_SIZE=X` in the **/vs/data/irAPI.rc** file, where X is the number of messages that the trace buffer can hold. (default = 256) Increasing the value of X should reduce the chance of losing trace messages.

CAUTION:
If you change the size of the trace buffer, you must stop and restart the VIS (stop_vs and start_vs). Otherwise, you will not be able to run trace.

Examples

The following are examples of valid level lists and ranges:

1,2 Trace levels at 1 and 2

1-4,FE Trace at levels 1, 2, 3, 4, and 20

all Trace at levels 1-32.

NOTE:
Levels are not hierarchically inclusive. That is, level 3 does not imply that tracing at levels 1 and 2 also occurs, which could be achieved by using a range starting from 1. For example, **1-3** for levels 1, 2, and 3.

Note that a user input (touchtone and speech recognition) log can be implemented by the following trace command:

```
trace chan all area IN level F
```

Files

/usr/spool/log/data/trace*
/vs/data/irAPI.rc

trarpt

The **trarpt** command generates a call traffic report.

Synopsis

trarpt *<hours> <summarize> <date>*

Description

The **trarpt** command generates a call traffic report. Information in this traffic report includes the number of calls coming in to the system during a specified time period, average holding time, and the percentage of time the channel was occupied for a certain hour. This report is stored in standard out (stdout). Before this can be done, the database system must be up and running, but the voice system does not need to be up.

The parameters for the **trarpt** command are:

- *<hours>* — Specifies the hours in which the traffic data was collected. The valid options can be a range between 0 to 23 (with 0 representing midnight and 23 representing 11 p.m.), or "all."
- *<summarize>* — Indicates a traffic report or a traffic summary report to be generated. If the option is "n", the report provides information on the total traffic volume for each channel in one-hour increments. If the option is "y," the report is a summary report that provides information on the total traffic volume for each channel for the whole period specified in the *<hours>* parameter.
- *<date>* — Specifies the date the data was collected in the system. This parameter must be in the mm/dd/yy format.

Example

The following example generates a traffic summary report for data collected on date August 24, 1993 between 8 a.m. and 5 p.m. on multiple entries per channel.

trarpt 8-17 y 08/24/93

The following example generates a traffic report for data collected on date August 24, 1993, one entry per channel)

trarpt all n 08/24/93

unassign_permissions

The **unassign_permissions** command removes VIS security permissions for a specific user.

Synopsis

unassign_permissions *<user login>*

Description

The **unassign_permissions** command removes VIS security permissions for a specific user.

The *<user login>* argument represents the user for which VIS permissions are to be removed. The user login will still exist; however, the user will not be able to access the VIS system. For information on removing the user login, see *Intuity CONVERSANT VIS Version 5.0 Operations*, 585-310-550.

Example

The following example executes the command to remove VIS security permissions.

unassign_permissions brown

See Also

"display_permissions"
"assign_permissions"

upg

The **upg** command provides automated assistance in upgrading an Intuity CONVERSANT VIS machine to Version 5.0.

Synopsis

/usr/lib/upgrade/bin/upg ***[-c] [-h]***

Description

The **upg** command and all of its associated files and functions are installed as part of the Software Upgrade Assistance Package delivered as part of an upgrade to Intuity CONVERSANT VIS Version 5.0. This command should be run with the full path name, **/usr/lib/upgrade/bin/upg**. The command assists with the procedure of upgrading a CONVERSANT VIS machine with Version 3.0 or later software on a MAP platform to a Version 5.0. It provides online guidance in implementing the upgrade procedures. In combination with *Intuity CONVERSANT VIS Version 5.0 Software Upgrade*, 585-310-152, the tool simplifies the upgrade procedure and eliminates many sources of potential errors in upgrades.

The procedures implemented by this command are intended to be self-documenting. Read and carefully follow all instructions that appear on the screen during the upgrade.

Before executing the command, perform all of the actions defined in "Getting Started" in Chapter 1, "Introduction to Software Upgrades," of *Intuity CONVERSANT VIS Version 5.0 Upgrade*s, 585-310-152, which applies to the particular customer scenario being upgraded. Help is available for most requests made by the upgrade package by entering **?**.

NOTE:
Help messages and graceful user exit from the questions which are part of each software package's individual removal or [re]installation are not available at this time.

Enter **q** to exit the upgrade procedure at any of these prompts. Exiting the procedure this way before the first phase of the upgrade (saving files, speech and database tables to be migrated to the target system), implies that the procedure must be redone from the beginning. When ready to begin the upgrade procedure again after exiting anytime after the end of the first phase, enter **upg** again to pick up the procedure where it left off.

Only the options intended for use in the field by customers are listed below. A full set of options may be accessed using the *-h* option. However, those options not listed either do not apply to Version 5.0 upgrades or are intended for use in the development environment and should not be used in the field by customers.

- *-c* — Introduces an additional confirmation step between each of the phases of the upgrade procedure, allowing you to abort the upgrade procedure (by entering **q**) between each phase.
- *-h* — Prints a help message defining the **upg** command syntax. All options are included in this message. Use this option if you wish to see a definition of the other available options, even though they are intended for use only in the development environment.

Files

All files associated with the Software Upgrade Assistance tool are stored/written in the **/usr/lib/upgrade** directory during the upgrade procedure. Save these files if you choose to remove the Software Upgrade Assistance package from the system before verifying *all* functionality of the upgraded system. If an upgrade includes a disk change of any kind (disk upgrade, repartition, etc.), then the first three files should be saved to floppy disk before any modifications are made to the disk.

- **output.list** — This file is a record of most screen interactions during the entire upgrade procedure.
- **CVIS.info** — This file is a survey of some hardware and software parameters and configuration settings on the system *before* it is upgraded. This file contains most of the information needed to install or reinstall newer versions of the software packages on the system.
- **rmvPkgsList** — This file contains an ordered list of the “older” software packages on the system. These are the packages that need to be removed and/or installed or reinstalled on the system to accomplish the upgrade.
- **insPkgsList** — This file contains an ordered list of the “new” software packages to be installed to accomplish the upgrade. These are the packages that need to be installed or reinstalled on the system to accomplish the upgrade.
- **fileList** — This file contains a list of the files encounted on the system which will be preserved in some form during the upgrade procedure. Many, but not all, will be restored on or merged into the target system during the upgrade.
- **ignoreList** — This file contains a list of files which are to be ignored, even if a later operation indicates they should be preserved.

vfyLogMsg

The **vfyLogMsg** command verifies the information associated with a specific logging message format.

Synopsis

"lComp" ***<msgnum>***

Description

The **vfyLogMsg** command, given a message number or symbolic message name, recomposes the message format from the information stored in the *cmpLogFmt* files generated by *"lComp"*.

NOTE:
You cannot use the **vfyLogMsg** command to look up a message format for a message class that you have just created, but not yet installed.

The *msgnum* argument can be in any of the following four formats:

- absolute message number

  The absolute message number would be if you were examining compressed logging files with an editor, for example, **238**

- symbolic name

  The symbolic name is found in the associated *log{CLASS}.h* header file, for example, **SYSMSG**

- message class/relative index in class pair

- **logGEN(2)** or **GEN.2**

  This last format can be specified in two ways: **logGEN(2)** or **GEN.2** Two forms exist because the log{CLASS}(index) form must be enclosed in quotes when used from the command line because the '(' and ')' are shell meta-characters, which is difficult to type.

The output of the **vfyLogMsg** command contains up to five different types of information about the message format:

- Interpretations of the message number

  The first block of information contains the three interpretations of the message number.

- Restored message format

  The second block of information includes the restored message format without any SQL field names that might have been specified in the original format.

- SQL field name information

  This information is the SQL field name information either as specified in the original format or as generated by "lComp" for those fields that did not have specifications in the input description. One description line exists for each argument on the machine.

- Current message priority

  The fourth block of information describes the current priority assigned to this message in that shared memory and the destination bit mask. This block of information is available only if the logging destination/priority shared memory exists on the machine.

- Description of each destination bit

  The fifth block of information describes each destination bit specified in the destination bit mask, starting with the lowest order bit.

Example

"lComp"

See Also

"logCat"
"logDstPri"

vsdisable

The **vsdisable** command disables the automatic restarting of the voice system.

Synopsis

vsdisable

Description

The **vsdisable** command is used to prevent the voice system from being started when the Intuity CONVERSANT VIS is rebooted. Running **vsdisable** allows you to log into the system before the voice system is started. The voice system may be started manually at any time with the **start_vs** command.

Example

vsdisable

See Also

"vsenable"

vsenable

The **vsenable** command enables the automatic starting of the voice system at system reboot.

Synopsis

vsenable

Description

When the **vsenable** command is run, UNIX system files are modified to allow the voice system to be automatically started when the Intuity CONVERSANT VIS is rebooted. By default, the voice system is installed with the automatic startup enabled. If there were any non-fatal problems during installation, the voice system is still installed but it has not enabled for automatic startup at system reboot. After the installation problems have been cleared, use **vsenable** to enable automatic voice system startup at reboot.

Example

vsenable

See Also

"vsdisable"

vusage

The **vusage** command displays the current load on the voice system.

Synopsis

vusage

Description

The **vusage** command enables the voice system administrator to determine the load on the voice system. It queries the voice system and prints the response on the screen, indicating the maximum number of channels in the system and the number of channels playing or coding, and the maximum number of buffers and the number in use.

Example

The following is an example of the **vusage** command and sample output.

```
$ vusage

Max (Current) Speech Buffers used:   0   (0)
Max (Current) Chans playing/coding:  0   (0)

$
```

WARNING:
The voice system must be running to execute this command.

See Also

"display channel"
"sysmon"

wdog_off

The **wdog_off** command disables the watchdog timer function of the ARU.

Synopsis

wdog_off

Description

The **wdog_off** command sends control sequences to the ARU that disable the watchdog timer function. When the watchdog is disabled, an asterisk (*) does not appear on the LED display next to the word “NORMAL.”

The watchdog timer should be reenabled with the **"wdog_on"** command prior to placing the system back into service. The timer should not be disabled for an operational system.

Example

wdog_off

See Also

"wdog_on"

wdog_on

The **wdog_on** command enables the watchdog timer function of the ARU.

Synopsis

wdog_on

Description

The **wdog_on** command sends control sequences to the ARU to enable the watchdog timer function. When the timer is enabled, the ARU shows an asterisk (*) next to the word “NORMAL” in the LED display.

Use the **wdog_on** command to reenable the watchdog timer after being disabled with the **"wdog_off"** command.

NOTE:
The watchdog timer should always be enabled for any operational system.

Example

wdog_on

See Also

"wdog_off"

wl_copy

The **wl_copy** command copies FlexWord wordlists to disk.

Synopsis

wl_copy ***<wordlist file>***

Description

The **wl_copy** command copies the wordlist files or directories given by names out to a floppy disk. Names should be relative pathnames, not absolute pathnames, since they will be used to load the vocabularies onto a FlexWord system.

NOTE:
Make sure change to the directory where your FlexWord wordlists are located, usually **/att/asr/wordlists/active**

If any of the names are directory names, the contents of the directories and any subdirectories are also copied to floppy disk.

Example

wl_copy database

wl_edit

The **wl_edit** command edits FlexWord wordlists.

Synopsis

wl_edit ***[-l <chan#>] [-s <TTS sp#>] [-D <directory>] [-O] [-I] [-?]***

Description

The **wl_edit** command invokes a Motif-based phoneme editor for wordlists. Wordlists are opened using a standard Motif interface, and then words can be added, changed, or deleted.

Audible playback is provided with Text-to-Speech so that you can hear the pronunciation of any word. You must dial into the channel specified with the *-l* option in order to hear the pronunciations.

Error messages are written to a small window at the bottom of the screen. Error messages are also logged to the file **/usr/tmp/wledit.output**.

The Tip/Ring [-l <*chan#*>] argument is used to specify which Tip/Ring channel to use for speech playback. If this argument is not specified, then playback will default to channel 0. Refer to Appendix C, “Computing Channel Numbers,” of *Intuity CONVERSANT VIS Version 5.0 Speech Development*, 585-310-228, for information about computing channel numbers.

NOTE:
Be sure to validate the Tip/Ring channel number before executing the **wl_edit** command.

The [-s <*TTS sp#*>] argument is used to specify which TTS sp circuit card to use for speech playback. Circuit cards are referenced by O.S. index. Therefore, you need to include the O.S. index of the TTS SP card for this argument, rather than the TTS SP card number. If this argument is not specified, then playback will default to the sp circuit card with O.S. index 0.

NOTE:
Be sure to validate the O.S. index before executing the **wl_edit** command .

The [-D directory] argument is used to specify that the program will start in the given directory.

The [-O] option is for debugging purposes: it causes certain Tip/Ring events to be recorded in the output window.

The [-I] option inhibits forced initialization of the T/R and SP cards. The **wl_edit** command usually determines if the cards need to be reset and provides a forced initialization. The advantage of using this option is that the FlexWord Editor comes up faster. The disadvantage is that in some unusual situations you may not be able to hear the pronunciation of your words. If this happens, run **wl_edit** again without specifying the -I option.

WARNING:
If the voice system is running, wl_edit will prompt you to stop the voice system before continuing.

Example

To dial into Tip/Ring channel 0 and use the TTS SP with O.S. index 5, enter the following command: **wl_edit -l0 -s5 -D /att/asr/wordlists/active -O -I**

See Also

"wl_init"

wl_gen

The **wl_gen** command creates data files for a FlexWord vocabulary.

Synopsis

wl_gen

Description

The **wl_gen** command uses the wordlists in **/att/asr/wordlists/active** and the models in **/att/asr/models/seg.yy21** to create all of the data files needed for FlexWord recognition. In particular, **wl_gen** creates the following files:

- **/att/asr/grammar_hs/sw_grammar.h** — A header file for FlexWord recognition scripts
- **/att/asr/sr_files/sr_file.sw** — Reformatted active wordlists
- **/vs/pack/cmp.seg.sw** — Vocabulary and model information for companion card DSPs.
- **vs/pack/resource.sw** — Wordlist resource information for the resource manager.

The **wl_gen** command should be called whenever there is a change to an active wordlist or when a wordlist has been activated or deactivated. You need to diagnose the FlexWord SP circuit card after using the **wl_gen** command.

If more than 38 phonemes, including the underscore (_), are found with any word or phrase within a wordlist by the **wl_gen** command, you receive an error message. Once **wl_gen** finds an error within a wordlist, it quits looking at that wordlist, and the wordlist containing that word or phrase is not used when generating the FlexWord data files. Therefore, if you receive an error, you may have to run **wl_gen** several times to iteratively locate each error.

An example of the error message you would receive regarding the filename *ACCT_NUM* is as follows:

```
Generating the FlexWord data files ...

'ACCT_NUM' ignored: it contains a word with more than 38 phonemes
        (Six_Six_Six_Six_Six_Six_Six_Six_Six_)
```

The **wl_gen** command retains the wordlist numbers of any active wordlist. This means that scripts do not have to be recompiled unless they use a wordlist that was not active on the previous invocation of **wl_gen**.

Example

wl_gen

See Also

"wl_edit"
"diagnose card"

wl_init

The **wl_init** command generates an initial wordlist from a set of words.

Synopsis

wl_init *<file>*

Description

The **wl_init** command takes a file consisting of words and/or phrases and adds a phonetic pronunciation for each word or phrase. The pronunciation is determined by a dictionary lookup, and uses the phonetic alphabet “cecilbet.” The input file should consist of one word or phrase per line, with ‘_’ instead of white space between words of a phrase. For example, “call Rachel” should be written “call_Rachel.” Words are case insensitive. Each line of the file will be augmented with a tab followed by the cecilbet phonetic transciption of the line.

Example

wl_init database

WARNING:
The file names should be upper case for Script Builder compatibility.

See Also

"wl_edit"

wl_install

The **wl_install** command reads FlexWord vocabularies from floppy disk.

Synopsis

wl_install

Description

The **wl_install** command reads FlexWord vocabularies from a floppy disk, and copies them into **att/asr/wordlists/inactive**. It then asks whether any wordlists are to be activated, and if necessary, whether **"wl_gen"** should be run.

Example

wl_install

See Also

"wl_copy"
"wl_gen"

xferdip_off

The **xferdip_off** command deactivates the bridging capability.

Synopsis

xferdip_off

Description

The **xferdip_off** command deactivates the bridging capability. If the xferdip is running, this command stops it.

Example

xferdip_off

xferdip_on

The **xferdip_on** command activates the bridging capability.

Synopsis

xferdip_on

Description

The **xferdip_on** command activates the bridging capability. If the VIS is running, this command starts the xferdip.

Example

xferdip_on

Abbreviations

A

AC
Alternating current

ACD
Automatic call distributor

AD
Application Dispatch

AD-API
Application dispatch application programming interface

ADPCM
Adaptive differential pulse code modulation

ADU
Asynchronous data unit

AGL
Application generation language

ALERT
VIS Alerter process

ANI
Automatic number identification

API
Application programming interface

ARU
Alarm relay unit

ASAI
Adjunct/Switch Application Interface

ASCII
American Standard Code for Information Interchange

ASI
Analog switch integration

B

BB
Bulletin board

bps
Bits per second

BRDG
Call bridging process

BSC
Binary synchronous communication

C

CCA
Call classification analysis

CDH
Call data handler

CELP
Continuously Excited Linear Prediction

CGEN
Voice system general message class

CICS
Customer Information Control System

CMP
Companion circuit card

CMS
Call Management System

CO
Central office

CPE
Customer provided equipment or customer premise equipment

CPN
Calling party number

CPT
Call progress tones

CPU
Central processing unit

CSU
Channel service unit

CVS
Converse vector step

D

dB
Decibels

DB
Database

DBC
Database checking process

DBMS
Database management system

DC
Direct current

DCE
Data communications equipment

DCP
Digital communications protocol

DIO
Disk input and output process

DIP
Data interface process

DMA
Direct memory access

DNIS
Dialed number identification service

DSP
Digital signal processor

DTE
Data terminal equipment

DTMF
Dual tone multi-frequency

DTR
Data terminal ready

E

EBCDIC
Extended Binary Coded Decimal Interexchange Code

EIA
Electronic Industries Association

EISA
Extended Industry Standard Architecture

EMI
Electromagnetic interference

ESD
Electrostatic discharge

ESDI
Extended Serial Data Interface

ESS
Electronic Switching System

ET
Error tracker

EXTA
External alarms feature message class

F

FCC
Federal Communications Commission

FDD
Floppy disk drive

FEP
Front end processor

FFE
Form Filler Plus feature message class

FIFO
First-in-first-out processing order

foos
Facility out-of-service state

FTS
File transfer process message class

G

GEN
PRISM logger and alerter general message class

GSE
Graphical Speech Editor

GUI
Graphical user interface

H

HDD
Hard disk drive

HLLAPI
High Level Language Application Programming Interface

HOST
Host interface process message class

hwoos
Hardware out-of-service state

Hz
Hertz

I

IBM
International Business Machines

ICK
Integrity checking process message class

ID
Identification

IDE
Integrated Disk Electronics

IE
Information element

INIT
Voice system initialization message class

inserv
In-service state

IPC
Interprocess communication

IPC
Intelligent Ports Card (IPC-900)

IPCI
Integrated personal computer interface

IRAPI
Intuity Response Application Programming Interface

IRQ
Interrupt request

ISA
Industry Standard Architecture

ISDN
Integrated Services Digital Network

ISV
Independent Software Vendor

ITAC
International Technical Assistance Center

IVP4
Integrated Voice Processing card with 4 analog channels

IVP6
Integrated Voice Processing card with 6 analog channels

IVPSS
Integrated Voice Processing System Software

K

Kbps
Kilobites per second

Kbyte
Kilobyte

L

LAN
Local area network

LDB
Local database

LED
Light-emitting diode

LIFO
Last-in-first-out processing order

LN
Load number

LOG
VIS logger process message class

LST1
Line side T1

LU
Logical unit

M

manoos
Manually out-of-service state

MAP/100
Multi-Application Platform 100

MAP/100C
Multi-Application Platform 100C

MAP/40
Multi-Application Platform 40

Mbps
Megabits per second

Mbyte
Megabyte

ms
Millisecond

msec
Millisecond

MHz
Megahertz

MTC
Maintenance process

N

NCP
Network Control Program

NEBS
Network Equipment Building Standards

NEMA
National Electrical Manufacturers Association

netoos
Network out-of-service state

NFAS
Non-Facility Associated Signaling

NFS
Network file sharing

NMVT
Network Management Vector Transport

NM-API
Network Management - Application Programming Interface

nonex
Nonexistent state

NRZ
Non Return to Zero

NRZI
Non Return to Zero Inverted

O

OEM
Original equipment manufacturer

OGA
Operator generated alert

P

PBX
Private branch exchange

PC
Personal computer

PCB
Printed circuit board

PCM
Pulse code modulation

PEC
Price element code

PRI
Primary rate interface

PSTN
Public switch telephone network

PS&BM
Power supply and battery module

R

RAM
Random access memory

RECOG
Speech recognition feature message class

RDBMS
ORACLE relational database management system

REN
Ringer equivalence number

RFS
Remote file sharing

RM
Resource manager

RMB
Remote maintenance board

RTS
Request to send

S

SBC
Sub-band coding

SCCS
Switching Control Center System

SCSI
Small Computer System Interface

SDLC
Synchronous Data Link Control

SDN
Software Defined Network

SID
Station identification

SIMM
Single inline memory module

SLIP
Serial Line Interface Protocol

SNA
Systems Network Architecture

SNMP
Simple Network Management Protocol

SP
Signal processor circuit card

SPIP
Signal processor interface process

SPPLIB
Speech processing library

SQL
Structured Query Language

SR
Speech recognition

SYS
UNIX system calls message class

sysgen
System generation

T

tas
Transaction assembler

TCC
Technology Control Center

TCP/IP
Transmission control protocol/internet protocol

TDM
Time division multiplexing

TE
Terminal emulator

THR
Threshold message class

TKR
Token Ring

TLI
Transport layer interface

TLP
Transmission level plan

T/R
Tip/Ring circuit card

TRIP
Tip/Ring interface process

TSO
Technical Service Organization

TSO
Time Share Operation

TSM
Transaction state machine process

TTS
Text-to-Speech

TWIP
T1 interface process

U

UK
United Kingdom

USOC
Universal service ordering code

UVL
Unified Voice Library

V

VDC
Video display controller

VIS
Intuity CONVERSANT Voice Information System

VPC
Voice processing comarketer

VRU
Voice response unit

VROP
Voice response output process

Glossary

Numerics

3270 interface
A link between one or more Intuity CONVERSANT Voice Information System (VIS) machines and a host mainframe. In Intuity CONVERSANT VIS documentation, the 3270 interface means the link between one or more VIS machines and an IBM host mainframe.

4ESS
A large AT&T central office switch used to route calls through AT&T's telephone network.

A

ACD
See "automatic call distributor."

ADPCM
See "adaptive differential pulse code modulation."

adaptive differential pulse code modulation
A means of encoding analog voice signals into digital signals by adaptively predicting future encoded voice signals. This adaptive modulation method reduces the number of bits required to encode voice. See also "pulse code modulation."

adjunct products
Products (for example, Adjunct/Switch Application Interface) that the Intuity VIS administers via cut-through access to the inherent management capabilities of the product itself; this is in opposition to CONVERSANT VIS's ability to administer the switch directly.

Adjunct/Switch Application Interface
An optional feature package that provides an Integrated Services Digital Network-based interface between AT&T PBX's and adjunct processors.

affiliate
A business organization that AT&T controls or which with AT&T is in partnership.

alarm relay unit
A unit used in central office telecommunication arrangements that transmits warning indicators from telephone communications equipment (like the Intuity CONVERSANT VIS) to audio.

alerter
A system process that responds to patterns of events logged by the "logdaemon" process.

analog
An analog signal, such as voice or music, that varies in a continuous manner. An analog signal may be contrasted with a digital signal, which represents only discrete states.

application
Made of several components that provide an automated version of the communication between a caller and an attendant. The Intuity CONVERSANT VIS provides several methods for creating applications, including Script Builder, the Intuity Response Application Programming Interface (IRAPI), and transaction state machine (TSM) script language.

application administration
The component of the Intuity CONVERSANT VIS that provides access to the applications currently available on your system and helps you to manage and administer them.

application installation
A two-step process in which the Intuity CONVERSANT VIS invokes the TSM script assembler for the specific application name and files are moved to the appropriate directories.

application verification
A process in which the Intuity CONVERSANT VIS verifies that all the components needed by an application are complete.

ASCII
An acronym for American Standard Code for Information Interchange, a standard for data representation. ASCII code represents alphanumeric characters as binary numbers. The code includes 128 upper- and lowercase letters, numerals, and special characters. Each alphanumeric and special character has an ASCII code (binary) equivalent that is 1 byte long.

asynchronous communication
A method of data transmission in which bits or characters are sent at irregular intervals and are spaced by start and stop bits and not by time. See also “synchronous communication.”

asynchronous data unit
An electronic communications device that allows computer systems to communicate over asynchronous lines more than 50 feet in length.

AUDIX Voice Power
A complete voice-mail messaging system accessed and operated by touch-tone telephones and integrated with a switch or “Private Branch Exchange.”

automatic call distributor
A telephone system that recognizes and answers incoming calls and completes these calls based on a set of instructions contained in a database. The Automatic Call Distributor can send the call to an operator or group of operators as soon as the operator has completed a previous call or after the system has played a message to the caller.

automatic number identification
A method of identifying the calling party by automatically receiving a string of digits that identifies the calling station of a particular customer.

B

back up
The preservation of the information in a file in a different location, so that the data is not lost in the event of hardware or system failure.

backing up an application
A utility that makes an archive copy of a completed application or makes an interim copy of an application in progress. The backup copy can be restored to the VIS if the online version is damaged, or if you make revisions and wish to go back to the previous version.

barge-in
A capability provided by WholeWord speech recognition that allow callers to speak their responses to the VIS prompt and have those responses recognized before the prompt has finished playing.

batch file
A file containing one or more lines, each of which is a command executable by the UNIX shell.

binary synchronous communications
A character-oriented synchronous link protocol.

blind transfer protocol
A protocol in which a call is completed as soon as the extension is dialed, without having to wait to see if the telephone is busy or if the caller answered.

bridging
The process of connecting one telephone network connection to another telephone network connection over the Intuity CONVERSANT VIS TDM bus. Bridging decreases the processing load on the system since an active bridge does not require speech processing, database access, host activity, etc., for the transaction.

BSC
See “binary synchronous communication.”

bundle
In the context of the Enhanced File Transfer package, this term is used to denote a single file, a group of files (package), or a combination of both.

byte
A unit of storage in the computer. On many systems, a byte is 8 bits (binary digits), the equivalent of one character of text.

C

call classification analysis
An optional feature package that allows application developers to classify the disposition of originated and transferred calls.

call data event
A parameter that specifies a list of variables that are appended to a call data record at the end of each call.

call data handler process
A software process that accumulates generic call statistics and application events.

called party number
The number dialed by someone making a telephone call. It can be used by telephone switching equipment to selectively route an incoming call to a particular department or agent.

caller
The party that calls for a service, gets connected to the Intuity CONVERSANT VIS, and interacts with the system. As the Intuity CONVERSANT VIS is also capable of making outbound calls for service, the caller can also be the person who responds to those outbound calls.

call progress tones
Standard telephony sounds that indicate the status of the call. These sounds include busy, fast busy, ringback, reorder, etc.

card cage
An area within a Intuity CONVERSANT VIS platform that contains and secures all of the standard and optional circuit cards used in the system.

cartridge tape drive
A high-capacity data storage/retrieval device that can be used to transfer large amounts of information onto high-density magnetic cartridge tape based on a predetermined format. This tape can be removed from the system and stored as a backup, or used on another system.

caution
An admonishment used when there is a possibility of a service interruption or a loss of data.

CCA
See “call classification analysis.”

CDH
See “call data handler process.”

central office
An office or location in which large telecommunication machines such as telephone switches and network access facilities are maintained. These locations follow strict installation and operation requirements.

central processing unit
A component of the Intuity CONVERSANT VIS that is based on either the Multi-Application Platform 100 (MAP/100), MAP/40, or MAP/100C.

channel
See “port.”

CICS
See “Customer Information Control System.”

circuit card upgrade
A new circuit card that replaces an existing one in the platform. Usually the replacement is an updated version of the other card, and the replacement is designed to deal with technology made obsolete by industry trends or a new VIS release.

cluster controller
A bisynchronous interface that provides a means of handling remote communication processing.

command
An instruction or request given by the user to the VIS software to perform a particular function. An entire command consists of the command name and options.

CompuLert/SCCS interface
An optional feature that enables remote or console monitoring of error messages generated from the Intuity CONVERSANT VIS. CompuLert is a centralized maintenance system for monitoring minicomputers, computer mainframes, etc. The Switching Control Center System (SCCS) is similar to the CompuLert system, but is used to support 4ESS local switching systems.

configuration
The arrangement of the software and hardware of a computer system or network. The Intuity CONVERSANT VIS configuration includes either a standard or custom processor, peripheral equipment (for example, printers, modems), and software applications. Configuration also refers to the way the switch network is set up; that is, the types of products that are in the network and how those products communicate.

configuration management
The component of the VIS that allows you to manage the current configuration of voice channels, host sessions, and database connections, assign scripts to run on specific voice channels or host sessions assign functionality to SP and T1 cards, and perform various maintenance functions.

Converse Data Return (conv_data)
A Script Builder action that supports the DEFINITY call vectoring (routing) feature by enabling the switch to retain control of vector processing in the VIS environment. It supports the DEFINITY "converse" vector command to establish a two-way routing mechanism between the switch and the VIS to facilitate data passing and return.

controller circuit card
A circuit card used on a computer system that controls its basic functionality and makes the system operational. These cards are used to control magnetic peripherals, video monitors, and basic system communications.

copying an application
A utility in which information from a source application is directed into the destination application.

coresidency
The ability of two products or services to operate and interact with each other on a single hardware platform. An example of this is the use of AUDIX Voice Power along with Intuity CONVERSANT on the same VIS platform.

CPU
See "central processing unit."

crash
An interactive utility for examining the operating system core and for determining if system parameters are being exceeded.

custom speech
Unique words or phrases to be used in Intuity CONVERSANT VIS voice prompts that AT&T records for a customer on a custom basis.

custom vocabulary
A specialized package of unique words or phrased created on a per-customer basis and used by WholeWord or FlexWord speech recognition.

Customer Information Control System
Part of the operating system that manages resources for running applications (for example, IND$FILE). Note that TSO and CMS provide analogous functionality in other host environments.

D

danger
An admonishment used when there is a possibility of personal injury.

data interface process
A software process that communicates with Script Builder applications.

database
A structured set of files, records, or tables.

database field
A field used to extract values from a local database and form the structure upon which a database is built.

database table
A structure, made up of columns and rows, that holds information in a database. Database tables provide a means of storing information that changes too often to "hard-code," or permanently store, in the transaction outline.

debug
The process of locating and correcting errors in computer programs. This process is also referred to as "troubleshooting."

default
The way a computer performs a task in the absence of other instructions.

default owner
The owner of a channel when no process takes ownership of that channel. The default owner holds all idle, in-service channels. In terms of the IRAPI, this is typically the Application Dispatch process.

diagnose
The process of performing diagnostics on Tip/Ring, T1, or SP circuit cards or a bus.

dialed number identification service
A service that allows incoming calls to contain information about the telephone number for which it is destined.

directory
A type of file used to group and organize other files or directories.

DNIS
See "dialed number identification service."

DIP
See "data interface process."

display errdata
A command that displays system errors sent to the logger.

DTMF
See “dual tone multi-frequency.”

dual 3270 links
A feature that provides an additional physical unit (PU) to allow a cost-effective means of connecting to two host computers. The customer can connect a VIS to two separate FEPs or to a single FEP shared by one or more host computers. Each link supports a maximum of 32 LUs.

dual tone multi-frequency
A touch tone.

dump space
An area of the disk that is fixed in size and should equal the amount of RAM on the system. The operating system “dumps” an image of core memory upon system crashes. The dump can be fetched after rebooting for analysis of what may have caused the crash.

E

editor system
A system that allows speech phrases to be displayed and edited by a user. See “Graphical Speech Editor.”

Enhanced File Transfer
A feature that allows the transferring of files automatically between the Intuity CONVERSANT VIS and a synchronous host processor on a designated logical unit.

Enhanced Serial Data Interface
A software- and hardware-controlled method used to store data on magnetic peripherals.

error message
A message on the screen indicating that something is wrong and possibly suggesting how to correct it.

Error Tracker process
See “etStub.”

Ethernet
A name for a local area network that uses 10BASE5 or 10BASE2 coaxial cable and InterLAN signaling techniques.

etStub
A system process that processes pre-Version 3.1 error message logging requests. These requests are transformed and passed on to the “logdaemon” process.

event
The notification given to an application when some condition occurs.

external actions
Specific tasks and interfaces controlled by Intuity CONVERSANT VIS software that allow a Script Builder application script to invoke processes and interact with other products or services. For example, a Intuity CONVERSANT VIS application script can invoke AUDIX Voice Power functionality through the used of an external action within an application script.

F

feature
A function or capability of a product or an application within the Intuity CONVERSANT VIS.

feature package
An optionally purchased package that may contain both hardware and software resources, which provides additional functionality to a standard system.

feature_tst script package
A standard CONVERSANT VIS software program that allows a VIS user to perform self-tests of critical hardware and software functionality.

field
A “slot” in a VIS window that holds one column of information in a row.

file
A collection of data treated as a basic unit of storage.

file transfer
An option that allows you to transfer files interactively or directly to and from UNIX using the File Transfer System.

filename
Alphabetic characters used to identify a particular file.

FlexWord speech recognition
A type of speech recognition based on subword technology that recognizes phonemes or parts of words of American English vocabularies. See “subword technology.”

Form Filler Plus
An optional feature package that provides the capability for application scripts to record caller’s responses to prompts for later transcription and review.

function key
A key, labeled F1 through F8, on your keyboard to which the Intuity CONVERSANT VIS software gives special properties for manipulating the user interface.

G

Graphical Speech Editor
A window-driven, X Windows/Motif based, graphical user interface (GUI) that can be accessed to perform different functions associated with the creation and editing of speech files to be used by VIS applications.

H

hard disk drive
A high-capacity data storage/retrieval device that is located inside a computer platform. A hard disk drive stores data on nonremovable high-density magnetic media based on a predetermined format for retrieval by the system at a later date.

hardware
The physical components of a computer system. The central processing unit, disks, tape and floppy drives, etc., are all hardware.

hardware upgrade
Replacement of one or more fundamental platform hardware components (for example, the CPU or hard disk drive), but the existing platform and other existing optional circuit cards remain.

High Level Language Applications Programming Interface (HLLAPI)
An application programming interface that allows user to write custom applications that can communicate with the host via an API.

HLLAPI
See “High Level Language Applications Programming Interface.”

host computer
A computer linked to a network providing a range of services, such as database access and computation. The host computer operates in a time-sharing manner with other computers linked to it via the network.

I

iCk
The system integrity checking process.

idle channel
A channel that either has no owner or is owned by its default owner and is onhook.

IND$FILE
The standard SNA file transfer utility that runs as an application under CICS, TSO, and CMS. IND$FILE is independent of link-level protocols such as BISYNC and SDLC.

indexed table
A table that, unlike a nonindexed table, can be searched via a field name that has been indexed.

initialize
To start up the system for the first time.

Integrated Services Digital Network
A network that provides end-to-end digital connectivity to support a wide range of voice and data services.

Integrated Voice Processing circuit card
The IVP4 or IVP6 circuit card.

intelligent transfer protocol
A transfer protocol that monitors the line after dialing is complete to determine whether a busy, reorder (fast busy), or other failure has been encountered. It also recognizes when the extension is answered or if the extension is not answered after a specified number of rings.

interface
The access point of a system. With respect to the Intuity CONVERSANT VIS, the interface is designed to provide you with easy access to the software's capabilities.

interrupt
The termination of voice and/or telephony functions when some condition occurs.

Intuity Response Application Programming Interface
A library interface the provides a standard development interface for voice-telephony applications.

ipcs
A command that reports interprocess communication facilities status.

IRAPI
See "Intuity Response Application Programming Interface."

ISDN
See "Integrated Services Digital Network."

K

keyboard mapping
In emulation mode, this feature enables the keyboard to send 3270 keyboard codes to the host according to a configuration table set up during installation.

keyword spotting
A capability provided by WholeWord Speech Recognition that allows the VIS to recognize a single word in the middle of an entire phrase spoken by a caller in response to a prompt.

L

LAN
See "local area network."

library states
The state information about channel activities maintained by the IRAPI.

line side T1
A digital method of interfacing a Intuity CONVERSANT VIS to a PBX or switch using T1-related hardware and software.

listfile
An ASCII catalog that lists the contents of one or more talkfiles. Each application script is typically associated with a separate listfile. The listfile maps speech phrase strings used by application scripts into speech phrase numbers.

local area network
A data communications network in a limited geographical area. The local area network provides communications between computers and peripherals.

local database
A database residing on the Intuity CONVERSANT VIS.

logical unit
A type of SNA Network Addressable Unit.

logdaemon
System information and error logging process.

logger
See “logdaemon.”

logging on/off
Entering or exiting the Intuity CONVERSANT VIS software.

LU
See “logical unit.”

M

magnetic peripherals
Data storage devices that use magnetic media to store information. Such devices include hard disk drives, floppy disk drives, and cartridge tape drives.

main screen
The Intuity CONVERSANT VIS VERSION 5.0 screen from which you are able to enter System Administration or Voice System Administration.

maintenance process
A software process that runs temporary diagnostics.

Manual Configurator Program
A software program that resolves or blocks the allocation of CPU and memory resources for controlling and optional circuit cards.

masked event
An event that an application can ignore (that is, the application can ask not to be informed of the event).

master
A board that provides clock information to the TDM bus.

megabyte
A unit of memory equal to 1,048,576 bytes (1024 x 1024). It is often rounded to one million.

Microsoft
A company that manufactures software products, primarily for IBM-compatible computers.

mirroring
A method of data backup that allows all of the data transactions to the primary hard disk drive to be copied and maintained on a second identical drive in near real time. If the primary disk drive crashes or becomes disabled, all of the data stored on it (up to 1.2 billion bytes of information) is accessible on the second mirrored disk drive.

MS-DOS
A personal computer disk operating system developed by the Microsoft Corporation.

MTC
See “maintenance process.”

multi-threaded application
A single process/application that controls several channels. Each thread of the application is managed explicitly. Typically this means state information for each thread is maintained and the state of the application on each channel is tracked.

N

NetView
An optional feature package that transmits high-priority (major or critical) messages to the host as Operator-Generated Alerts (OGAs) over the 3270 host link. The NetView Alarm feature package does not require a dedicated LU.

new error logging environment
A more flexible and informative environment for logging errors and status messages (introduced in CONVERSANT VIS Version 3.1). Customer applications created earlier than V3.1 that log messages require conversion to this new environment.

new operating system
The UnixWare operating system being introduced in Intuity CONVERSANT VIS V5.0.

nonindexed table
A table that may be searched only in a sequential manner and that cannot be searched via a field name.

nonmasked event
An event that must be sent to the application. Generally, an event is nonmaskable if the applicaiton would likely encounted state transition errors by trying to ignore the event.

null value
An entry containing no value. A field containing a null value is normally displayed as blank and is different from a field containing a value of zero.

O

obsolete hardware
Hardware that is no longer supported on Intuity CONVERSANT VIS V5.0.

on-line help
Messages or information that appear on the user's screen when a "function key" (F1 through F8) is pressed.

Operator Generated Alerts
System monitoring messages transmitted from the CONVERSANT VIS or other computer system to an IBM host computer that are classified as critical or major.

option
An argument used in a command line to modify program output by modifying the execution of a command. When you do not specify any options, the command will execute according to its default options.

ORACLE
A company that produces Relational Database Management software. It is also used as a generic term that identifies a database residing on a local or remote system that is created and maintained using an ORACLE RDBMS product.

P

PBX
See "private branch exchange."

PCM
See "pulse code modulation."

peripheral (device)
Equipment such as printers or terminals that is in addition to the basic processor.

permanent process
A process that starts and initializes itself before it is needed by a caller.

phoneme
A single basic sound of particular spoken language. The English language contains 40 phonemes that represent all basic sounds used with the language. As an example, the word "one" can be represented with three phonemes, "w" - "uh" - "n." Phonemes vary between languages because of guttural and nasal inflections and syllable constructs.

phrase filtering
The rejection of unrecognized speech. The WholeWord and FlexWord speech recognition packages can be programmed to reprompt the caller if the spoken response was not recognized by the VIS.

phrase tag
A string of up to 50 characters that identify the contents of a speech phrase used by an application script.

platform migration
See “platform upgrade.”

platform upgrade
The process of replacing the existing platform with a new platform.

poll
A message sent from a central controller to an individual station on a multipoint network inviting that station to send if it has any traffic to send.

polling
A network arrangement whereby a central computer asks each remote location whether they wish to send information. This arrangement enables each user or remote data terminal to transmit and receive information on shared facilities.

port
A connection or link between two devices that allows information to travel to a desired location. See “telephone network connection.”

Primary Rate Interface
An optional feature package that provides a digital interface capable both of receiving and originating telephone calls directly from/to an AT&T 4ESS switch.

private branch exchange
A private switching system, either manual or automatic, usually serving an organization, such as a business or government agency, and usually located on the customer’s premises.

processor
In Intuity CONVERSANT VIS documentation, the computer on which UnixWare and Intuity CONVERSANT VIS software runs. In general, the part of the computer system that processes the data. Also known as the “central processing unit.”

ps
A command that shows active processes. This command displays the process table and can be used to determine which processes are consuming large amounts of system resources, such as CPU time.

pseudo driver
A driver that does not control any hardware.

pulse code modulation
A digital modulation method of encoding voice signals into digital signals. See also “adaptive differential pulse code modulation.”

R

recovery
The process of using copies of the VIS software to reconstruct files that have been lost or damaged. See also “restore.”

remote database
The component of the VIS that provides access to information not currently on the VIS.

remote maintenance board
A Intuity CONVERSANT VIS board that is equipped standard on all new MAP/100 and MAP/40 platform purchases. This card, available with a built-in modem, allows remote personnel (for example, field support) to access all Intuity CONVERSANT VIS machines with a standard simplified process.

reports administration
The component of the VIS that provides access to system reports, including VIS call classification reports, call data detail reports, call data summary reports, message log reports, and traffic reports. In addition, if AUDIX Voice Power R2.1.1 is installed on your system, the reports administration component gives you access to AUDIX Voice Power reports.

restore
The process of recovering lost or damaged files by retrieving them from available backup tapes or from another disk device. See also “recovery.”

restore application
A utility that replaces a damaged application or restores an older version of an application.

reuse
The concept of reusing an existing system component after a software upgrade or platform migration.

roll back
To cancel changes to a database since the point at which changes were last committed.

rollback segment
A portion of the database that records actions that should be undone under certain circumstances. Rollback segments are used to provide transaction rollback, read consistency, and recovery.

S

sar
A command that is associated with the system activity report package.

screen pop
A method of delivering a screen of information to a telephone operator at the same time a telephone call is delivered. This is accomplished by a complex chain of tasks that include identifying the calling party number, using that information to access a local or remote ORACLE database, and pulling a “form” full of information from the database using an ORACLE database utility package.

script
The set of instructions for the Intuity CONVERSANT VIS to follow during a transaction.

Script Builder
An optional software package that provides a menu-oriented interface designed to assist in the development of custom voice response applications on the VIS.

SCSI
See “Small Computer System Interface.”

shared database table
A database table that is used in more than one application.

shared speech
Speech that is a part of more than one application.

shared speech pools
A parameter that allows the user of a voice application to share speech components with other applications.

Single Inline Memory Modules
A method of containing random access memory (RAM) chips on narrow circuit card strips that attach directly to sockets on the CPU circuit card. Multiple SIMMs are sometimes installed on a single CPU circuit card.

single-threaded application
An application that runs on a single voice channel.

slave
A circuit card that depends on the TDM bus for clock information.

Small Computer System Interface
A disk drive control technology in which a single SCSI adapter card plugged into a PC slot is capable of controlling as many as seven different hard disks, optical disks, tape drives, etc.

software
The set or sets of programs that instruct the computer hardware to perform a task or series of tasks — for example, UnixWare software and the Intuity CONVERSANT VIS Version 5.0 software.

software upgrade
The installation of a new version of software. The existing platform and circuit cards are kept.

source system
The system from which you are upgrading (that is, your system as it exists *before* you upgrade).

speech energy
The amount of energy in an audio signal. Literally translated, it is the output level of the sound in every phonetic utterance.

speech envelope
The linear representation of voltage on a line. It reflects the sound wave amplitude at different intervals of time. This envelope can be plotted on a graph to represent the oscillation of an audio signal between the positive and negative extremes.

speech file
A file containing an encoded speech phrase.

speech filesystem
A collection of several talkfiles. The filesystem is organized into 16-Kbyte blocks for efficient management and retrieval of talkfiles. The Intuity CONVERSANT VIS speech filesystem is not consistent with standard UNIX filesystems, and can not be referenced with standard UNIX commands such as **ls**, **cat**, etc.

speech modeling
Creating WholeWord speech recognition algorithms by collecting thousands of different speech samples of a single word and comparing them all to obtain a statistical average of the word. This average is then used by a WholeWord speech recognition program to recognize a single spoken word.

speech phrase
A continuous speech segment encoded into a digital string.

speech space
An area that contains all digitized speech used for playback in the applications loaded on the system.

standard speech
The speech package containing simple words and phrases produced by AT&T for use with an Intuity CONVERSANT VIS. This package includes digits, numbers, days of the week, and months, each spoken with initial, medial, and falling inflection. The speech is in digitized files stored on the hard disk to be used in the voice prompts played by the VIS.

standard vocabulary
A standard package of simple word speech models provided by AT&T and used for WholeWord speech recognition purposes. These phrases include the digits "zero" through "nine," "yes," "no," and "oh."

string
A contiguous sequence of characters treated as a unit. Strings are normally bounded by white spaces, tabs, or a character designated as a separator. A string value is a specified group of characters symbolized by a variable.

Structured Query Language
A standard data programming language used with data storage and data query applications.

subword technology
A method of speech recognition that recognizes phonemes or parts of words of American English vocabularies. See "whole-word technology."

switch
A software and hardware device that controls and directs voice and data traffic. A customer-based switch is known as a "private branch exchange."

switch hook
The device at the top of most telephones that is depressed when the handset is resting in the cradle (on hook). The device is raised when the handset is picked up (the telephone is off hook).

switch hook flash
A signaling technique in which the signal is originated by momentarily depressing the "switch hook."

switch interface administration
The component of the VIS that enables you to define the interaction between the VIS and switches by allowing you to establish and modify switch interface parameters and protocol options for both analog and digital interfaces.

switch network
Two or more interconnected switching systems.

synchronous communication
A method of data transmission in which bits or characters are sent at regular time intervals, rather than being spaced by start and stop bits. See also "asynchronous communication."

System 75
An advanced digital switch supporting up to 800 lines that provides voice and data communications for its users.

System 85
An advanced digital switch supporting up to 3000 lines that provides voice and data communications for its users.

system administrator
The person assigned the responsibility of monitoring all VIS software processing, performing daily system operations and preventive maintenance, and troubleshooting errors as required.

system architecture
The manner in which the Intuity CONVERSANT VIS software is structured.

system message
An event or alarm generated by either a VIS or end-user process.

system monitor
A component of the VIS in which tests are performed to verify that each incoming telephone line and its associated tip/ring or T1 card is functional. Through the "System Monitor" component, you are able to see displays of the Voice Channel and Host Session Monitors.

T

T1
A digital transmission link with a capacity of 1.544 Mbps.

table
A collection of records that are logically grouped together.

talkfile
An ASCII file that contains the speech phrase tags and phrase tag numbers for all the phrases of a specific application. The speech phrases are organized and stored in groups. Each talkfile can contain up to 65,535 phrases and the speech filesystem can contain multiple talkfiles.

target system
The system to which you are upgrading (that is, your system as you expect it to exist *after* you upgrade).

TDM
See "time-division multiplex."

telephone network connection
The point at which a telephone network connection terminates on an Intuity CONVERSANT VIS. Supported telephone connections are Tip/Ring and T1.

Terminal Emulator
Software that allows the VIS to temporarily transform itself into a "look alike" of an IBM 3270 terminal. In addition to providing full 3270 functionality, the Terminal Emulator enables you to transfer files to and from UNIX.

Text-to-Speech
An optional feature that allows an application to play speech directly from ASCII text by converting that text to synthesized speech. The text can be used for prompts or for text retrieved from a database or host, and can be spoken in an application with prerecorded speech. Text-to-Speech application development is supported through Script Builder.

ThickNet
A 10-millimeter (10BASE5) coaxial cable used to provide InterLAN communications.

ThinNet
A 5-millimeter (10BASE2) coaxial cable used to provide InterLAN communications.

time-division multiplex
A method of serving a number of simultaneous channels over a common transmission path by assigning the transmission path sequentially to the channels, with each assignment being for a discrete time interval.

Tip/Ring
A term used to denote analog telecommunications using four-wire media.

Token/Ring
A ring type of local area network that allows any station in the network to communicate with any other station.

trace
A command that can be used to monitor the execution of a script.

traffic
The flow of information or messages through a communications network for voice, data, or audio services.

transaction
Comprised of the exchanges between the caller and the voice system. A transaction can involve one or more telephone network connections and voice responses from the Intuity CONVERSANT VIS. It can also involve one or more of the VIS optional features, such as speech recognition, 3270 host interface, FAX response, etc.

transaction state machine process
A multi-channel IRAPI application that runs applications driven by script information.

transient process
A process that is created dynamically only when needed.

troubleshoot
The process of locating and correcting errors in computer programs. This process is also referred to as debugging.

TSM
See “transaction state machine process.”

TTS
See “Text-to-Speech.”

U

UNIX Operating System
A multiuser, multitasking computer operating system developed by the Bell Telephone Laboratories division of AT&T.

UNIX shell
The command language that provides a user interface to the UNIX operating system.

upgrade image tape
A tape, optionally provided to you by the Technical Service Organization, containing the new operating system and Intuity CONVERSANT VIS V5.0 base software in a standard configuration which is compatible with your target system.

upgrade scenario
The particular combination of current hardware, software, application and target hardware, software, applications, etc.

V

vi editor
A screen editor used by the Intuity CONVERSANT VIS to create and change electronic files.

virtual channel
A channel that is not associated with an interface to the telephone network (Tip/Ring, T1, or PRI). Virtual channels are intended to run "data only" applications which do not interact with callers but may interact with DIPs. Voice or network functions (for example, coding or playing speech, call answer, origination, or transfer) will not work on a virtual channel. Virtual channel applications may be initiated only by a "virtual seizure" request to TSM from a DIP.

VIS
See "Voice Information System."

vocabulary
A collection of words that a VIS is able to recognize using either WholeWord or FlexWord speech recognition.

vocabulary activation
The set of active vocabularies that define the words and wordlists known to the FlexWord recognizer.

vocabulary loading
The process of copying the vocabulary from the system where it was developed and adding it to the target system.

voice channel
A channel that is associated with an interface to the telephone network (Tip/Ring, T1, or PRI). Any Intuity CONVERSANT VIS application can run on a voice channel. Voice channel applications may be initiated by being assigned to particular voice channels or dialed numbers to handle incoming calls or by a "soft seizure" request to TSM from a data interface process (DIP) or the **soft_szr** command.

Voice Information System
A computer connected to a telephone network that handles touch-tone input, voice response, and line transfer. The Voice Information System uses a screen-based, menu-driven user interface to interact with the system operator or administrator.

voice processing co-marketer
A company licensed to purchase voice processing equipment, such as the Intuity CONVERSANT VIS, to market and sell based on their own marketing strategies.

voice response output process
A software process that transfers digitized speech between system hardware (for example, Tip/Ring and SP cards) and data storage devices (that is, hard disk, etc.)

Voice System Administration
The means by which you are able to administer both voice- and nonvoice-related aspects of the system.

VROP
See “voice response output process.”

W

warning
An admonishment used when there is a possibility of equipment damage.

WholeWord speech recognition
An optional feature based on whole-word technology that provides speaker independence, connected digit recognition, key word spotting, prompt interrupt, and DTMF support functionality. See “whole-word technology.”

whole-word technology
The ability to recognize an entire word, not the phoneme or a part of a word. See “subword technology.”

wink signal
An interruption of current to a busy lamp indicating that there is a line on hold.

word
A unique utterance understood by the recognizer.

wordlist
A set of words identified by a wordlist name. If the wordlist is part of an active vocabulary, the wordlist name appears as a recognition type in the Prompt & Collect mode field.

word spotting
The ability to search past extraneous speech during a recognition.

Index

Numerics

A

B

C

D

E

F

G

H

I

L

M

N

R

S

T

U

V

W

X