

Window Terminal User's Guide

Notice

The information contained in this document is subject to change without notice.

UNLESS EXPRESSLY SET FORTH IN A WRITTEN AGREEMENT SIGNED BY AN AUTHORIZED REPRESENTATIVE OF STRATUS COMPUTER, INC., STRATUS MAKES NO WARRANTY OR REPRESENTATION OF ANY KIND WITH RESPECT TO THE INFORMATION CONTAINED HEREIN, INCLUDING WARRANTY OF MERCHANTABILITY AND FITNESS FOR A PURPOSE. Stratus Computer, Inc., assumes no responsibility or obligation of any kind for any errors contained herein or in connection with the furnishing, performance, or use of this document.

Software described in Stratus documents (a) is the property of Stratus Computer, Inc., or the third party, (b) is furnished only under license, and (c) may be copied or used only as expressly permitted under the terms of the license.

Stratus manuals document all of the subroutines and commands of the user interface. Any other operating-system commands and subroutines are intended solely for use by Stratus personnel and are subject to change without warning.

This document is protected by copyright. All rights are reserved. No part of this document may be copied, reproduced, or translated, either mechanically or electronically, without the prior written consent of Stratus Computer, Inc.

Stratus, the Stratus logo, Continuum, VOS, Continuous Processing, StrataNET, FTX, and SINAP are registered trademarks of Stratus Computer, Inc.

XA, XA/R, Stratus/32, Stratus/USF, StrataLINK, RSN, Continuous Processing, Isis, the Isis logo, Isis Distributed, Isis Distributed Systems, RADIO, RADIO Cluster, and the SQL/2000 logo are trademarks of Stratus Computer, Inc.

Apple and Macintosh are registered trademarks of Apple Computer, Inc.
IBM PC is a registered trademark of International Business Machines Corporation.
Sun is a registered trademark of Sun Microsystems, Inc.
Hewlett-Packard is a trademark of Hewlett-Packard Company.
UNIX is a registered trademark of X/Open Company, Ltd., in the U.S.A. and other countries.
HP-UX is a trademark of Hewlett-Packard Company.
Manual Name: *Window Terminal User's Guide*

Part Number: R256
Revision Number: 00
VOS Release Number: 10.0
Printing Date: November 1990

Stratus Computer, Inc.
55 Fairbanks Blvd.
Marlboro, Massachusetts 01752

© 1990 by Stratus Computer, Inc. All rights reserved.

Preface

The Purpose of This Manual

The *Window Terminal User's Guide (R256)* documents the user interface to the window terminal software.

Audience

This manual is intended for users working on character-mode terminals.

Before using the *Window Terminal User's Guide (R256)*, you should be familiar with the following Stratus manuals.

- *Introduction to VOS (R001)*
- *VOS Commands User's Guide (R089)*
- *VOS Commands Reference Manual (R098)*
- The user's guide for your terminal

Revision Information

This is the first publication of this manual. For information on which release of the software this manual documents, see the Notice page.

Manual Organization

This manual has five chapters and two appendixes.

[Chapter 1](#) provides an overview of the window terminal software, including input requests, windows, and the window manager. It also notes the system requirements for the use of this software.

[Chapter 2](#) illustrates primary window layout and describes the input requests that are available when you issue commands.

[Chapter 3](#) explains how to manage command output using input requests.

[Chapter 4](#) describes the input requests used to enter and leave window manager mode and manipulate primary windows.

[Chapter 5](#) discusses how to obtain help system information and status displays and how to update and clear the status area.

[Appendix A](#) summarizes the input requests described throughout this manual.

[Appendix B](#) provides key maps for the Stratus V101, V102, and V103 terminals. The key maps associate keystroke sequences with input requests.

Notation

Stratus documentation uses *italics* to introduce or define new terms. For example:

The *command area* initially contains one line for the `ready` prompt and one line for a command.

Computer font is used to represent text that would appear on your display screen or on a printer. (Such text is referred to as *literal* text.) For example:

To change the tab settings, issue the `set_terminal_parameters` command and specify the new settings in the `-tabs1` and `-tabs2` arguments.

Slanted font is used to represent general terms that are to be replaced by literal values. In the following example, the user must supply an actual value to replace the slanted-font term.

The status display contains the *person_name* portion of your user name.

Boldface is used to emphasize words within the text. For example:

In Overlay mode, any character you type **replaces** the character on which the cursor is positioned.

Related Manuals

Refer to the following Stratus manuals for related documentation.

- *Introduction to VOS (R001)*
- *VOS Commands User's Guide (R089)*
- *VOS Commands Reference Manual (R098)*
- *VOS Communications Software: Asynchronous Communications (R025)*
- *VOS Communications Software: Defining a Terminal Type (R096)*
- *Window Terminal Programmer's Guide for Asynchronous Communications (R194)*
- *VOS System Administrator's Guide (R012)*
- The user's guide for your terminal

Online Documentation

You can find additional information by viewing the system's online documentation in `>system>doc`. The online documentation contains the latest information available, including updates and corrections to Stratus manuals.

A Note on the Contents of Stratus Manuals

Stratus manuals document all the subroutines and commands of the user interface. Any other commands and subroutines contained in the operating system are intended solely for use by Stratus personnel and are subject to change without warning.

How to Comment on This Manual

You can comment on this manual by using the command `comment_on_manual`, described in the *VOS System Administrator's Guide (R012)*. Type `comment_on_manual`, press `[RETURN]`, and then complete the form that appears on your screen. You must fill in this manual's part number, *VOS System Administrator's Guide (R012)*. When you have completed the form, press `[ENTER]`. Your comments are sent to Stratus over the Remote Service Network. Note that the operating system includes your name with your comments.

Stratus welcomes any corrections and suggestions for improving this manual.

Contents

1. Window Terminal Overview	1-1
System Requirements	1-1
Input Requests.	1-3
Groups of Input Requests	1-3
Format of Input Requests	1-4
Request Names Shown in Key Outlines	1-4
Request Names Shown in Computer Font	1-4
Windows and the Window Manager	1-4
Primary Windows and the Window Manager	1-5
Multiple Primary Windows	1-6
Subwindows	1-7
 2. Using Input Requests When Issuing Commands.	 2-1
Screen Layout	2-1
The Command Area	2-2
The Output Area	2-2
The Status Area.	2-3
Issuing Commands	2-4
Issuing Commands on the Command Line	2-4
Issuing Commands Using the Display Form	2-4
Moving from Field to Field in a Display Form	2-8
Adding or Changing Field Values in a Display Form.	2-9
Executing and Canceling the Display Form of a Command	2-10
Retrieving Input After Issuing a Command	2-10
The Insert-Saved Buffer	2-10
The Insert-Default Buffer	2-12
Command-Line Behavior in the Command Area.	2-13
Interrupting Command Execution.	2-15
Moving the Cursor within a Command Line or a Display-Form Field	2-16
The ← and → Requests	2-16
The GOTO Requests	2-16
The WORD Requests.	2-16
The TAB Requests	2-17
The BLANKS Requests	2-18
Editing a Command Line or a Display-Form Field Value	2-20
The INSERT/OVERLAY Request	2-20
The DEL and DELETE Requests	2-21
The BACK_SPACE Request.	2-22

The <code>CHANGE_CASE</code> Requests.	2-22
The <code>CHANGE_CASE</code> <code>↑</code> and <code>CHANGE_CASE</code> <code>↓</code> Requests	2-22
The <code>WORD</code> <code>CHANGE_CASE</code> Requests	2-23
Miscellaneous Requests	2-25
The <code>REPEAT_LAST</code> Request	2-25
The <code>REDISPLAY</code> Request	2-25
The <code>INTERRUPT</code> Request	2-26
3. Managing Output	3-1
How Output Is Displayed	3-2
Displaying and Discarding Paused Output.	3-4
The <code>↓</code> and <code>NEXT_SCREEN</code> Requests	3-4
The <code>DISCARD</code> Request	3-6
Disabling the Pause Mechanism.	3-7
4. The Window Manager	4-1
Entering and Leaving Window Manager Mode	4-2
Entering Window Manager Mode.	4-2
Leaving Window Manager Mode.	4-2
Creating a New Primary Window	4-3
Moving among Primary Windows	4-5
Canceling Activity in the Current Primary Window	4-7
Breaking a Process in the Current Primary Window	4-7
Stopping the Login Process Associated with the Current Primary Window	4-8
Logging Out of the Current Primary Window	4-8
Redisplaying the Current Primary Window	4-8
5. Getting Help System and Status Information	5-1
Getting Help System Information	5-1
Getting Help about a Command	5-1
Getting Help about an Argument	5-2
Getting Status Information	5-3
The Contents of the Status Area	5-3
Updating and Clearing the Status Area.	5-5
Appendix A. Summary of Input Requests	A-1
Appendix B. Function/Special Key Maps for Stratus Terminals	B-1
Glossary	Glossary-1
Index.	Index-1

Figures

Figure 1-1. A Sample Screen Display	1-5
Figure 1-2. Three Primary Windows	1-6
Figure 1-3. A Primary Window Containing Two Subwindows	1-7

Tables

Table A-1. Command-Line Requests	A-2
Table A-2. Display-Form Requests	A-4
Table A-3. Window Manager Requests	A-7
Table B-1. Keystroke Sequences for the V101 Keyboard	B-1
Table B-2. Keystroke Sequences for the V102 Keyboard	B-3
Table B-3. Keystroke Sequences for the V103 ASCII Keyboard.	B-5
Table B-4. Keystroke Sequences for the V103 EPC Keyboard	B-7

Chapter 1:

Window Terminal Overview

The window terminal software is a data communications software package designed to provide a user-friendly interface between a Stratus module and a character-mode terminal. It provides these features:

- clearly delineated command and output areas
- “typeahead” capability that allows you to enter additional commands while the current command is executing
- extensive line-editing capabilities
- keystroke sequences to manage output display
- a window manager that allows you to manage multiple processes on one terminal
- access to help and status information.

This chapter provides an overview of the window terminal software. First, it notes the system requirements that must be met in order to use the software. Second, it describes how you issue input requests. Third, it explains the window manager feature that lets you create and use multiple processes on your terminal.

In this chapter and throughout the manual, the examples use operating system commands to illustrate application programs that are written to run with the window terminal software. Other application programs that are written for the window terminal software function similarly.

The window terminal software described in this manual is different from the asynchronous software described in the manual *VOS Communications Software: Asynchronous Communications (R025)*.

System Requirements

Your module and terminal must meet certain requirements before you can use the window terminal software.

First, your module must be running Release 10.0 or a subsequent release of the VOS operating system. To determine the operating system release, issue the following command.

```
display_line (module_info system_release)
```

Second, your terminal must be a character-mode terminal. Examples of character-mode terminals are the V101, V102, and V103 terminals available from Stratus. Block-mode terminals, intelligent workstations, and other intelligent devices cannot use the window terminal software (except by emulating character-mode terminals). Check with your system administrator if you are not sure what type of terminal you have.

Third, your terminal must be configured as a window terminal. Ask your system administrator to do this; configuring terminals requires special access to certain files.

- If your terminal has not yet been configured, ask your system administrator to specify `window_term` for the `device_type` field in the table input file `devices.tin` before creating the table file `devices.table`.
- If your terminal is currently configured as another device type (for example, `terminal`), ask your system administrator to specify `window_term` for the `device_type` field for your terminal's entry in the `devices.tin` file. Then, re-create the table file `devices.table`. To determine how your terminal is currently configured, issue the `display_device_info` command (described in the *VOS Commands Reference Manual (R098)*).

The next time your module is shut down and rebooted, your terminal will be configured as a window terminal.

Finally, the input requests you use that are specific to window terminals must be defined as generic input requests in a terminal type definition file for your terminal. Check with your system administrator to confirm that this has been done.

The manual *VOS Communications Software: Defining a Terminal Type (R096)* explains how to prepare and use terminal type definition files.

If you are using the window terminal software on an unsupported terminal, you must define the following sequences in the input section of the `terminal.ttp` file that defines the terminal you are using.

```
abort-output
next-screen
```

For supported terminals, Release 10.0 of the VOS operating system provides a new format for the terminal type definition files

(`>system>sample_programs>supported_ttps>*.ttp`). If you are using the window terminal software on an unsupported terminal, Stratus recommends that you convert the `terminal.ttp` file for the terminal to the new format. (See Revision 01 of the manual *VOS Communications Software: Defining a Terminal Type (R096)* for a description of the new format.)

Specifically, when converting to the new format, add the following input section to the `terminal.ttp` file for an unsupported terminal. Note that in the following example, the

value of *fkey* is the name of the key mapped to the CANCEL request in the terminal type definition file for your terminal.

```

input      $window_manager
  leave-window-manager  enter-key
  leave-window-manager  fkey-key      {cancel}
  login-process         f1-key
  break-process         f6-key
  stop-process          f7-key

```

If you do not make these changes, you will still be able to use the window terminal software; however, you will not be able to use the window manager feature.

Input Requests

As you work at your terminal, numerous input requests are available to you. To issue an input request, you press a certain key or sequence of keys. RequestsInput requests

For example, suppose that while entering the `print` command, you mistakenly type `prng`. You have not actually invoked the command, so you can correct the error on the command line. To do so, you issue the `BACK_SPACE` request by pressing the `BACK_SPACE` key. This moves the cursor one space to the left and deletes the character in that position (the `g`). You can now correctly type the last letter of the command name and issue the command.

Note that if a keystroke sequence contains any of the following keys, you must hold that key down while you press the next key in the sequence.

- `SHIFT` or `Shift`
- `CTRL` or `Ctrl`
- `Funct`

Groups of Input Requests

There are several groups of input requests. A basic group of requests is available at command or application level. These requests are described in [Chapter 2](#), “Using Input Requests When Issuing Commands,” [Chapter 3](#), “Managing Output,” and [Chapter 5](#), “Getting Help System and Status Information.” Some requests in the basic group are also available when you are using Stratus-supplied or other application programs, though their functions may not be the same as at command level or in display forms. (Display forms are described briefly later in this chapter. For a detailed discussion on using display forms, see [Chapter 2](#), “Using Input Requests When Issuing Commands.”)

Additional groups of input requests may be available only while using specific application programs or window terminal features. [Chapter 4](#), “The Window Manager,” describes the basic input requests that can be used when in window manager mode and the input requests that are specific to window manager mode. For information on input requests associated with other Stratus-supplied application programs (such as the Emacs text editor), see the appropriate Stratus manual.

Format of Input Requests

In this manual, names of input requests appear in one of two formats: either in key outlines or in computer font. This subsection describes both of these formats.

Request Names Shown in Key Outlines

If the request name appears either on the template for the terminal keyboard or on the key itself, the request name is shown in a key outline (for example, `DISPLAY_FORM` or `RETURN`). Each entry on a template corresponds to the function key below it. For example, the `DISPLAY_FORM` entry on the V102 template corresponds to the `F21` key.

Most requests fall into this category. To issue a request of this type, you simply press the key associated with the request. For example, to issue a `DISPLAY_FORM` request, you press the key identified by the words `DISPLAY FORM` printed on the template. To issue a `RETURN` request, you press the key with the word `RETURN` printed on it.

Note: The keyboard layouts, key legends, and the input requests issued when you press keys vary from terminal to terminal. For example, to issue a `CANCEL` request, you press the `F17` key on the V101 terminal, the `F20` key on the V102 terminal, the `F18` key on the V103 terminal with the ASCII keyboard, or the `*` key on the V103 terminal with the Enhanced Personal Computer (EPC) keyboard. Unless otherwise noted, the keystroke sequences presented in the examples in this manual show the key legends for the V102 terminal. [Appendix B](#), “Function/Special Key Maps for Stratus Terminals,” contains tables listing the keystroke sequences you use to issue input requests on the V101, V102, and V103 terminals.

Request Names Shown in Computer Font

If the request name does not appear on either the template or the key, the request name is shown in computer font and in uppercase letters with the words separated by underline characters (`_`) instead of spaces (for example, the `NEXT_SCREEN` request).

To issue a request of this type, use the keystroke sequence defined for that request. To determine the correct keystroke sequence for a V101, V102, or V103 terminal, see the tables in [Appendix B](#), “Function/Special Key Maps for Stratus Terminals.” For example, you issue a `NEXT_SCREEN` request by using the keystroke sequence `SHIFT ↓`.

To determine the keystroke sequences for other terminals, see the user’s guide and terminal type definition file for the specific terminal. For more information on terminal type definition files, see the manual *VOS Communications Software: Defining a Terminal Type (R096)*.

Note: You **cannot** issue these requests by typing their names as if they were operating system commands. For example, you cannot issue a `NEXT_SCREEN` request by typing the string `NEXT_SCREEN` at command level. You must press the correct key or sequence of keys.

Windows and the Window Manager

The window terminal software allows you to interact with multiple processes at the same terminal. Each process is associated with a *window*, which is a portion of the terminal screen. The *window manager* provides you with input requests that you use to manipulate these windows.

To use window manager requests, you must enter window manager mode. After performing the desired window-related activity, you leave window manager mode and resume your activities. For example, you can enter window manager mode, issue the `[CYCLE]` request to move to another window, then leave window manager mode and issue commands or run an application program in the window to which you cycled.

Note that when your terminal is in window manager mode, you are communicating with the window manager, not with the application program from which you entered window manager mode. You do not resume communication with the application program until you leave window manager mode.

See [Chapter 4](#), “The Window Manager,” for an explanation of how to enter and leave window manager mode and how to use window manager requests.

Primary Windows and the Window Manager

A *primary window* comprises the entire terminal screen and displays the output of a single process or application program. When you log in from a window terminal, the window terminal software provides a primary window in which your user process runs. You can now issue commands.

[Figure 1-1](#) illustrates how your screen might look after you issue a `list` command.

Display Form

```
w      1 announce_shutdown.cm
w      1 contents.cm
w      1 create_alias_links.cm
w      1 delete_alias_links.cm
w      1 delete_carefully.cm
w      1 display_both_access_lists.cm
w      1 display_reminders.cm
w      1 dk2.cm
w      1 do_multiple.cm
w      1 edit_abbreviations.cm
w      1 edit_nonwords.cm
w      1 edit_phone_list.cm
w      1 edit_reminders.cm
w      1 find_future_day.cm
w      1 get_rid_of_accesses.cm
w      1 get_titles.cm
w      1 get_titles.ufi
w      1 keys_to_chars.pm
w      1 print_multiple.cm
w      1 reminders.cm
w      1 run_print_test.cm
ready 09:33:13
█
06-08 09:57 | 21% | Lee_Smith | ready | Insert
```

Figure 1-1. A Sample Screen Display

Multiple Primary Windows

The window manager allows you to open additional primary windows, each of which is associated with a different process. Multiple primary windows overlap, and only one primary window, called the *current primary window*, is visible on the screen at a time. You can think of primary windows as a stacked deck of cards: the window on top is the current primary window, and the other windows are stacked below it. Any terminal input is associated with the activity in the current primary window.

The other primary windows, although not visible, are always available to you unless you delete them. The processes in the other primary windows run concurrently with the process in the current primary window. The window manager allows you to cycle forward or backward through all of the primary windows until you reach the desired window. This window becomes the current primary window, and all of your terminal input is associated with it.

Figure 1-2 illustrates the primary window shown in Figure 1-1, with two other primary windows behind it.



Figure 1-2. Three Primary Windows

Subwindows

Each primary window contains one or more subwindows. A *subwindow* is a section of a primary window. While you may frequently use subwindows, you do not create them. Subwindows are provided by the application programs that you run, whether they are standard Stratus-supplied programs or programs developed specifically for your site.

Subwindows may or may not overlap one another. [Figure 1-3](#) illustrates a primary window containing two subwindows that overlap. In this example, the first subwindow contains the output of a previous `list` command. The contents of this subwindow are partially obscured by the second subwindow, which contains the display form of the current `list` command. (The *display form* of a command provides a list of the available arguments, along with fields into which you enter argument values. See [Chapter 2](#), “Using Input Requests When Issuing Commands,” for a detailed description of display forms.) When the second subwindow is removed, the first subwindow’s obscured output is redisplayed.

Display Form

```
w          1 contents.cm
w          1 create_alias_links.cm
w          1 delete_alias_links.cm
w          1 delete_carefully.cm
w          1 display_both_access_lists.cm
w          1 display_reminders.cm
w          1 dk2.cm
w          1 do_multiple.cm
w          1 edit_abbreviations.cm
w          1 edit_nonwords.cm
w          1 edit_phone_list.cm
w          1 edit_reminders.cm
w          1 find_future_day.cm
----- list -----
path_name:  █
-files:     yes
-dirs:      no
-links:     no
-sort:      name
-full:      no
-names_only: no
-totals:    no
-header:    no
-exclude:
06-08 09:57 | 21% | Lee_Smith | ready | Insert
```

Figure 1-3. A Primary Window Containing Two Subwindows

Chapter 2:

Using Input Requests When Issuing Commands

The first part of this chapter describes primary window layout: the appearance of the primary window and the areas that it comprises. The remainder of this chapter describes the input requests available to you when issuing commands. The requests are grouped according to the tasks they perform.

- Issuing commands
- Moving the cursor within text
- Editing text
- Performing miscellaneous tasks

[Appendix A](#), “Summary of Input Requests,” provides tables that summarize **all** of the requests described in this manual.

Screen Layout

A primary window is divided into three areas.

- The command area contains the `ready` prompt and any commands to be executed.
- The output area contains command output.
- The status area can contain either information about the primary window, or a message from another user, an application program, or the operating system.

This section provides a detailed description of each of these areas.

Figure 1-1, reproduced here for your reference, illustrates the three areas of a primary window.

Display Form

```

w      1 announce_shutdown.cm
w      1 contents.cm
w      1 create_alias_links.cm
w      1 delete_alias_links.cm
w      1 delete_carefully.cm
w      1 display_both_access_lists.cm
w      1 display_reminders.cm
w      1 dk2.cm
w      1 do_multiple.cm
w      1 edit_abbreviations.cm
w      1 edit_nonwords.cm
w      1 edit_phone_list.cm
w      1 edit_reminders.cm
w      1 find_future_day.cm
w      1 get_rid_of_accesses.cm
w      1 get_titles.cm
w      1 get_titles.ufi
w      1 keys_to_chars.pm
w      1 print_multiple.cm
w      1 reminders.cm
w      1 run_print_test.cm
ready  09:33:13

```

```
06-08 09:57 | 21% | Lee_Smith | ready | Insert
```

The Command Area

The *command area* initially contains one line for the `ready` prompt and one line for a command. After entering one command, you can enter additional commands while the first command is executing—this is the “typeahead” feature of the window terminal software. You enter the additional commands in the command area. For each line of command input that you enter, one line of the output area scrolls off the top of the screen.

The command area in the previous example is shown here.

```
ready 09:33:13
```

Note that if you are using the short form of the `ready` prompt, the command appears on the same line as the prompt; therefore, the command area contains only one line. To use the short form of the `ready` prompt, issue the `VOS set_ready` command with the `-format brief` argument. The `set_ready` command is described in the *VOS Commands Reference Manual (R098)*.

The Output Area

When a command begins to execute, both the line containing the `ready` prompt and the command line move up to the output area. (If you are using the short form of the `ready` prompt, the line containing both the prompt and the command moves up to the output area.)

The first line of command output appears at the bottom of the output area. If there is more output, the contents of the output area move up to make room for it. If there is enough room in the output area for all of the output, the `ready` prompt appears and the next command in the command area (if any) executes.

The output area in the previous example is shown here.

```
w      1 announce_shutdown.cm
w      1 contents.cm
w      1 create_alias_links.cm
w      1 delete_alias_links.cm
w      1 delete_carefully.cm
w      1 display_both_access_lists.cm
w      1 display_reminders.cm
w      1 dk2.cm
w      1 do_multiple.cm
w      1 edit_abbreviations.cm
w      1 edit_nonwords.cm
w      1 edit_phone_list.cm
w      1 edit_reminders.cm
w      1 find_future_day.cm
w      1 get_rid_of_accesses.cm
w      1 get_titles.cm
w      1 get_titles.ufi
w      1 keys_to_chars.pm
w      1 print_multiple.cm
w      1 reminders.cm
w      1 run_print_test.cm
```

If there is not enough room for all of the output, the display of output pauses. See [Chapter 3](#), “Managing Output,” for information on how to continue the display of output, delete output, and control the pause mechanism.

The Status Area

The bottom portion of the screen is the *status area*. You can request a display of information about your current primary window and receive messages in the status area.

The status area in the previous example is shown here.

```
06-08 09:57 | 21% | Lee_Smith | ready | Insert
```

See [Chapter 5](#), “Getting Help System and Status Information,” for a description of the input requests you can use to produce and update status displays.

Issuing Commands

Executing commandsIssuing commands

You can issue an operating system command either on the command line or using a display form. In either case, you provide information (arguments and their values) that the command processor uses to execute the specified command. This section explains how to use input requests to issue and cancel operating system commands and to redisplay previous commands. It also describes command-line behavior in the command area of the screen, and explains how to interrupt the execution of a command.

This section is not intended to provide a detailed description of command usage or operating system commands. For this information, see the *VOS Commands User's Guide (R089)* and the *VOS Commands Reference Manual (R098)*, respectively.

Issuing Commands on the Command Line

To issue a command on the command line, enter the name of the command and any arguments you want. If you make an error while entering the command name or arguments, you can correct the error by using the requests described later in this chapter under “Moving the Cursor within a Command Line or a Display-Form Field” and “Editing a Command Line or a Display-Form Field Value.”

After you have entered the command name and arguments, use the **RETURN** or **ENTER** request to issue the command.

For example, suppose that you enter the following command and argument.

```
list *.cm -sort size
```

To issue this command, press the **RETURN** or **ENTER** key.

If you decide that you do not want to issue the command after entering it, press the **CANCEL** key to cancel the command instead of pressing the **RETURN** or **ENTER** key.

Issuing Commands Using the Display Form

The display form of a command provides a list of the available arguments, along with the fields into which you enter values. To invoke the display form of a command, enter the command name and then issue the **DISPLAY_FORM** request.

For example, suppose that you enter the `list` command and then press the `DISPLAY_FORM` key. The `list` display form appears on your screen, overlaying the output from previous commands. The screen might look like this.

```
m      12 admin
m      1 backup_files
m      1 cal_1989
m      1 cust_act
m      1 cust_inact
m      1 database
m      2 eng
m      1 system_events
m      2 forms
m      5 macros
m      6 misc
m      1 mtg_minutes
m      1 outputs
----- list -----
path_name:  █
-files:     yes
-dirs:      no
-links:     no
-sort:      name
-full:      no
-names_only: no
-totals:    no
-header:    no
-exclude:
```

Note that some of the arguments have a default value (yes for `-files`; no for `-dirs`, `-links`, `-full`, `-names_only`, `-totals`, and `-header`; and `name` for `-sort`). These default values are used unless you supply other values.

You can also enter arguments on the command line **before** you press the `DISPLAY_FORM` key. In this case, the arguments are inserted into the corresponding fields in the form, replacing the default values (if any). Suppose that you enter the following command and then press the `DISPLAY_FORM` key.

```
list *.cm -sort size
```

The screen now looks like this.

```
m      12 admin
m      1 backup_files
m      1 cal_1989
m      1 cust_act
m      1 cust_inact
m      1 database
m      2 eng
m      1 system_events
m      2 forms
m      5 macros
m      6 misc
m      1 mtg_minutes
m      1 outputs
```

```
----- list -----
path_name:  *.cm
-files:     yes
-dirs:      no
-links:     no
-sort:      size
-full:      no
-names_only: no
-totals:    no
-header:    no
-exclude:
```


As mentioned earlier, a display form can overlay output from a previous command. The obscured output reappears when you issue or cancel the command whose display form you invoked. Any output from this command is appended to the output from the previous command. (Some of the output from the previous command may scroll off the top of the screen to make room for the new output.) For example, when the command in the previous example finishes executing, the display form disappears and the screen looks like this.

```
m          1 cust_act
m          1 cust_inact
m          1 database
m          2 eng
m          1 system_events
m          2 forms
m          5 macros
m          6 misc
m          1 mtg_minutes
m          1 outputs
m          1 safety_net
m          4 sysadmin
m          1 tape_dumps
m          1 tapes
m          1 waste_basket
m          4 workshop
m          1 writing_group
m          1 xtra
ready 09:33:13
list *.cm -sort size
list:  No match for star name. %s1#m2>Sales>Smith>*.cm
ready 09:34:26
█
```

Note that the previous command output has reappeared, the output from the `list` command (an error message) has been appended, and three lines of output have scrolled off the top of the screen.

You can also invoke the display form of a command by entering the keyword `-form` after the command name and then pressing the `RETURN` or `ENTER` key.

Moving from Field to Field in a Display Form

Within the display form of a command, you can use various cursor-movement requests to move the cursor from field to field. The descriptions of the cursor-movement requests reference the following sample display form, which appears on the screen when you issue the `print` command.

```
----- print -----
file_names:
-queue:                standard
-title:
-destination:
-module:
-device:
-header:
-footer:
-index:
-defer_until:
-interpret_tabs:
-exception_handling:   replace
-copies:               1          -line_numbers:           no
-delete:               no         -raw:                  no
-page_breaks:          yes        -use_fortran_controls: no
-indentation:          0          -page_size:
-top_margin:           3          -bottom_margin:        3
-line_length:
-queue_priority:
-pass_thru:            no         -notify:               no
```

You have the following cursor-movement options.

- Next field: `TAB` or `TAB_STOP` `→`

The `TAB` or `TAB_STOP` `→` request moves the cursor to the next field in a display form. For example, if the cursor is positioned on the `file_names` field, these requests move the cursor to the `-queue` field. If the cursor is positioned on the `-copies` field, these requests move the cursor to the `-line_numbers` field. If the cursor is positioned on the `-pass_thru` field, these requests move the cursor to the `file_names` field.

- Previous field: `BACK_TAB` or `TAB_STOP` `←`

The `BACK_TAB` or `TAB_STOP` `←` request moves the cursor to the previous field in a display form. For example, if the cursor is positioned on the `-line_numbers` field, these requests move the cursor to the `-copies` field. If the cursor is positioned on the `-copies` field, these requests move the cursor to the `-exception_handling` field. If the cursor is positioned on the `file_names` field, these requests move the cursor to the `-pass_thru` field.

- First field on next line: `RETURN`

The `RETURN` request moves the cursor to the first field on the next line. For example, if the cursor is positioned on the `-exception_handling` field, this request moves the cursor to the `-copies` field. If the cursor is positioned on the `-copies` field, this

request moves the cursor to the `-delete` field. If the cursor is positioned on the `-pass_thru` field, this request moves the cursor to the `file_names` field.

- Closest field on previous or next line: `↑` and `↓`

The `↑` request moves the cursor to the closest field on the previous line. For example, if the cursor is positioned on the `-raw` field, this request moves the cursor to the `-line_numbers` field. If the cursor is positioned on the `-line_numbers` field, this request moves the cursor to the `-exception_handling` field. If the cursor is positioned on the `file_names` field, this request moves the cursor to the `-pass_thru` field.

The `↓` request moves the cursor to the closest field on the next line. For example, if the cursor is positioned on the `-exception_handling` field, this request moves the cursor to the `-copies` field. If the cursor is positioned on the `-copies` field, this request moves the cursor to the `-delete` field. If the cursor is positioned on the `-pass_thru` field, this request moves the cursor to the `file_names` field.

- First or last field: `GOTO ↑` and `GOTO ↓`

The `GOTO ↑` request moves the cursor from its current position to the first field in the display form. For example, if the cursor is positioned on the `-line_numbers` field, this request moves the cursor to the `file_names` field.

The `GOTO ↓` request moves the cursor from its current position to the last field in the display form. For example, if the cursor is positioned on the `-index` field, this request moves the cursor to the `-pass_thru` field.

Adding or Changing Field Values in a Display Form

Once you have positioned the cursor on a field, you can add or change the value in that field. If the field is not a cycle field, you enter the desired value. If you make an error when entering argument names or values, you can correct the error before issuing the command. To do so, use the requests described later in this chapter in “Moving the Cursor within a Command Line or a Display-Form Field” and “Editing a Command Line or a Display-Form Field Value.”

If the field is a cycle field, only a predetermined set of values is allowed. You must issue the `CYCLE`, `CYCLE_BACK`, `→`, or `←` request to cycle among the choices. (In the previous example, the following are cycle fields: `-exception_handling`, `-line_numbers`, `-delete`, `-raw`, `-page_breaks`, `-use_fortran_controls`, `-wrap`, `-notify`, and `-pass_thru`.)

You can also type the first letter of a cycle-field value to display that value. For example, the `-exception_handling` argument of the `print` command has the following cycle-field values: `replace` (the default), `ignore`, and `abort`. If you position the cursor on the `-exception_handling` field and type `i`, the value `ignore` appears; if you type `a`, the value `abort` appears.

If more than one cycle-field value begins with the specified letter, the first value in the list that begins with the letter appears. Typing the letter a second time displays the next value in the list that begins with the letter, and so on. For example, the `-sort` argument of the `list` command has the following cycle-field values: `name` (the default), `size`, `date_created`,

date_modified, date_used, and date_saved. If you position the cursor on the -sort field and type d, the value date_created appears. If you type d a second time, the next value in the list beginning with the letter d, date_modified, appears.

Executing and Canceling the Display Form of a Command

Once you have specified the argument values you want the command to use, issue the **ENTER** request to execute the command.

If you decide before issuing the **ENTER** request that you do not want to execute the command, simply issue the **CANCEL** request.

Retrieving Input After Issuing a Command

Two buffers are used to store input: the insert-saved buffer and the insert-default buffer. You can retrieve input stored in these buffers by issuing the **INSERT_SAVED** or **INSERT_DEFAULT** request, respectively. These requests are useful for repeating a command or a portion of a command without re-entering the entire command string.

There is one insert-saved buffer for each terminal. You can move the input stored in the insert-saved buffer from one primary window to another. However, each primary window has its own insert-default buffer. The contents of an insert-default buffer are not affected by input in other primary windows. Therefore, when you return to a specific primary window, the insert-default buffer contains the last stored input.

Input that is copied into the insert-saved or insert-default buffer replaces any input that was previously stored in the buffer.

The Insert-Saved Buffer

Three types of operations store input in the insert-saved buffer.

- Issuing a delete request. When you issue the **DELETE** **←**, **DELETE** **→**, or **DELETE** **WORD** editing request, the deleted input is copied into the insert-saved buffer. This allows you to “cut” a portion of a command line and “paste” it into another location on the same command line or a command line in another primary window. You can also cut a value from a display-form field and paste it into another display-form field.
- Issuing a command. Commands that you issue on the command line are copied into the insert-saved buffer. If you issue a command using the display form, only the portion of the command that you type on the command line before pressing the **DISPLAY_FORM** key is copied into the insert-saved buffer. Argument values that you set or modify within the display form are **not** copied into this buffer.
- Responding to a request for input by a command or an application program. If a command or application program requests input, your response is copied into the insert-saved buffer. A break-level prompt is one example of a request for input.

Suppose that you want to list all of the files in the directory

%sl#ml>Sales>Smith>macros, **except** those whose names end in the suffix .pm. Using the **INSERT_SAVED** request (as described in the following steps), you can issue the appropriate

list command without entering the name of the directory twice. The following example illustrates this procedure.

1. Invoke the display form of the list command (using the `DISPLAY_FORM` request) and enter the path name `%s1#m1>Sales>Smith>macros>*`, which represents all of the files in the directory.

```
----- list -----
path_name:    %s1#m1>Sales>Smith>macros>*
-files:      yes
-dirs:       no
-links:      no
-sort:       name
-full:       no
-names_only: no
-totals:     no
-header:     no
-exclude:
```

2. Issue the `DELETE` `<` request. This deletes the path name from the display form and stores it in the insert-saved buffer.
3. Issue the `INSERT_SAVED` request to re-enter the path name in the `path_name` field.
4. Move the cursor to the `-exclude` field (using the `↓` request, for example).
5. Issue another `INSERT_SAVED` request to enter the path name in the `-exclude` field.

```
----- list -----
path_name:    %s1#m1>Sales>Smith>macros>*
-files:      yes
-dirs:       no
-links:      no
-sort:       name
-full:       no
-names_only: no
-totals:     no
-header:     no
-exclude:    %s1#m1>Sales>Smith>macros>*
```

6. Now type `.pm` at the end of the path name.

```
----- list -----  
path_name:    %s1#m1>Sales>Smith>macros>*  
-files:       yes  
-dirs:        no  
-links:       no  
-sort:        name  
-full:        no  
-names_only:  no  
-totals:      no  
-header:      no  
-exclude:     %s1#m1>Sales>Smith>macros>*.pm
```

You can now issue the `list` command and generate the desired output.

The Insert-Default Buffer

The insert-default buffer usually contains the most recently issued command and any arguments that you specified.

- If you issue a command on the command line, the insert-default buffer contains the command exactly as you entered it.
- If you issue a command using the display form, the insert-default buffer contains the command name and any argument values that you modified.
- If you cancel the display form of a command (using the `CANCEL` request), the insert-default buffer contains the input stored from the command you last issued; that is, input from the command that you canceled is **not** copied into the insert-default buffer.

To prevent input that you are entering on the command line from overwriting the current contents of the insert-default buffer, you must issue the command using the `ENTER` request rather than the `RETURN` request.

You can retrieve the contents of the insert-default buffer at any time **unless** you are working in a display form. Issuing the `INSERT_DEFAULT` request while working in a display form restores the current field to its default value.

Suppose that you issue the `list` command using the display form and specify the values shown here.

```
----- list -----
path_name:    start*
-files:       yes
-dirs:        no
-links:       no
-sort:        date_used
-full:        yes
-names_only:  no
-totals:      no
-header:      yes
-exclude:     start_up.cm
```

Now suppose that you want to issue the `list` command again. Issuing the `INSERT_DEFAULT` request retrieves the following command line.

```
list start* -sort date_used -full -header -exclude start_up.cm
```

The `INSERT_DEFAULT` request retrieves the command name and only those arguments for which you specified nondefault values: `-sort date_used`, `-full`, `-header`, and `-exclude start_up.cm`. (The default value of the `-exclude` field is the null string.)

Command-Line Behavior in the Command Area

All command lines that you enter appear in the command area. When a command begins to execute, it moves from the command area to the output area, and the command area disappears (unless there are other commands pending in the command area). If a status display appears in the status area, the name of the command that is currently executing is displayed.

If a command cannot be executed immediately (because the previous command has not finished executing), it is displayed in the command area until it can be executed. (The command is displayed in half intensity, if your terminal supports this feature.) You can enter multiple command lines in the command area while the previous command is executing. The command area expands to provide room for pending command lines. If necessary, output lines scroll off the top of the screen to accommodate the pending command lines.

The following example illustrates a screen in which a number of commands have accumulated in the command area while the `list` command is executing.

```

w          1 announce_shutdown.cm
w          1 contents.cm
w          1 create_alias_links.cm
w          1 delete_alias_links.cm
w          1 delete_carefully.cm
w          1 display_both_access_lists.cm
w          1 display_reminders.cm
w          1 dk2.cm
w          1 do_multiple.cm
w          1 edit_abbreviations.cm
w          1 edit_nonwords.cm
w          1 edit_phone_list.cm
w          1 edit_reminders.cm
w          1 find_future_day.cm
w          1 get_rid_of_accesses.cm
w          1 get_titles.cm
w          1 get_titles.ufi
w          1 keys_to_chars.pm
--PAUSE--
display_reminders
edit_abbreviations
change_current_dir work_in_progress
pll test_prog
bind test_prog
06-08 09:57 | 21% | Lee_Smith | ready | Insert

```

When the `list` command has finished executing, the `display_reminders` command is executed and is moved up out of the command area. The screen now looks like this.

```

w          1 get_titles.cm
w          1 get_titles.ufi
w          5 keys_to_chars.pm
w          1 print_multiple.cm
w          1 reminders.cm
w          1 run_print_test.cm
w          1 send_future_message.cm
w          1 set_remote.cm
w          1 staff
w          1 store.cm
w          3 test_prog.pll
w          2 tester.pll
w          1 unpropagate_access.cm
ready 09:57:01
display_reminders
>>--> Department meeting at 11 AM
>>--> Submit status report
>>--> 1030: Meet with manager
ready 09:57:23
edit_abbreviations
change_current_dir work_in_progress
pll test_prog
bind test_prog
06-08 09:57 | 21% | Lee_Smith | ready | Insert

```


When the `display_reminders` command has finished executing, the `edit_abbreviations` command is executed and moves up out of the command area. This process continues until all commands in the command area are executed.

Interrupting Command Execution

In some cases, you may want to interrupt a command or program before it finishes executing. For example, the command that you issued is displaying more output than you want to see, or you may have entered the wrong command name.

To interrupt a command or program, you issue a `BREAK` request. A `BREAK` request places the terminal in window manager mode. (See [Chapter 4](#), “The Window Manager,” for a detailed discussion of window manager mode.) If you issue a `BREAK` request again, it interrupts command or program execution and displays the following break prompt, requesting instructions from you on how to proceed.

```
BREAK
Request?  (stop, continue, debug, keep, login, re-enter)
```

By default, a `BREAK` request discards all pending input (typeahead). An application program can change this behavior using the `s$control` subroutine. See the *Window Terminal Programmer's Guide for Asynchronous Communications (R194)* for information about the `s$control` subroutine.

Note that the `BREAK` request name is not shown in a key outline. In this manual, the absence of a key outline generally indicates that the corresponding key (as identified by words on the keyboard template or by the key legend) does not exist. A `BREAK` request is slightly different. A key with the legend `BREAK` commonly exists on a keyboard, but it does not always produce a `BREAK` request. Typically, you issue a `BREAK` request by using one of the following keystroke sequences.

- BREAK
- CTRL BREAK
- CTRL C

For the keystroke sequences that begin with CTRL, you must hold down the CTRL key while you press the second key.

Moving the Cursor within a Command Line or a Display-Form Field

This section describes the requests that you can use to move the cursor within a command line or a display-form field. (Note that you cannot edit cycle fields in a display form.) You can move the cursor using the following requests.

- and
-
-
-
-

These requests may also be available in Stratus-supplied or other application programs. For more information, see the documentation for the specific application program.

The and Requests

The and requests move the cursor one character to the left or one character to the right, respectively. If you issue the request at the beginning of the current command line or display-form field, or the request at the end of the command line or display-form field, the cursor does not move.

The and requests, along with the and requests, are commonly used with other requests (such as and) to modify the actions of those requests.

The Requests

There are four requests in this category: , , , and .

The request positions the cursor at the beginning of the current command line or display-form field. If the cursor is already at the beginning of the line or field, it does not move.

The request positions the cursor at the end of the current command line or display-form field (that is, one position to the right of the last character in the line or field). If the cursor is already at the end of the line or field, it does not move.

The and requests are available only in a display form. These requests move among fields rather than within text. The requests are described earlier in this chapter under “Issuing Commands Using the Display Form.”

The Requests

There are two requests in this category: and .

The request moves the cursor to the first character of the current word or, if the cursor is already positioned on the first character of the current word or on a space between words, moves the cursor to the first character of the word to the left of the current word. If the cursor is at the beginning of a command line or display-form field, it does not move.

For example, suppose that the cursor is positioned as shown in this example.

```
print cust_list memo.smith
```

Issuing the **[WORD]** **[←]** request re-positions the cursor on the first `m` in `memo.smith`.

Suppose instead that the cursor is positioned as shown in either of the following examples.

```
print cust_list memo.smith
```

```
print cust_listmemo.smith
```

In both cases, issuing the **[WORD]** **[←]** request re-positions the cursor on the `c` in `cust_list`.

The **[WORD]** **[→]** request moves the cursor one word to the right in the current command line or display-form field. This request moves the cursor to the first character of the word to the right of the current word. If the cursor is positioned on the last word of a command line or display-form field, the request moves the cursor one position to the right of that word.

For example, issuing the **[WORD]** **[→]** request when the cursor is positioned as shown in any of the following examples moves the cursor to the first `m` in `memo.smith`.

```
print cust_list memo.smith
```

```
print cust_list memo.smith
```

```
print cust_listmemo.smith
```

The **[TAB]** Requests

There are two requests in this category: **[TAB]** and **[BACK_TAB]**.

When you issue the **[TAB]** request on the command line, the cursor moves to the next tab stop. You can determine the current tab settings for your terminal by issuing the `display_terminal_parameters` command and looking for the line that begins with the

word Tabs. To change the tab settings, issue the `set_terminal_parameters` command and specify the new settings using the `-tabs1` and `-tabs2` arguments.

The `BACK_TAB` request does **not** perform the opposite action of the `TAB` request. When you issue the `BACK_TAB` request on the command line, the cursor moves to the beginning of the command line.

When you issue the `TAB` and `BACK_TAB` requests while working in a display form, the cursor moves from field to field. These requests are described earlier in this chapter under “Issuing Commands Using the Display Form.”

The `BLANKS` Requests


There are two requests in this category: `BLANKS` `←` and `BLANKS` `→`.

The `BLANKS` `←` request moves the cursor to the left to the space immediately following the first nonblank character. If no blank characters immediately precede the cursor, the cursor does not move.


The `BLANKS` `→` request moves the cursor to the right to the first nonblank character. If no blank characters immediately follow the cursor, the cursor does not move.

Suppose that you enter the value `phone_list` minutes in the `file_names` field of the display form of the `print` command. You then issue the `WORD` `←` request to move the cursor to the `m` in `minutes`. The resulting display looks like this.

```
----- print -----
file_names:          phone_list  m minutes
-queue:              standard
-title:
-destination:
-module:
-device:
-header:
-footer:
-index:
-defer_until:
-interpret_tabs:
-exception_handling:  replace
-copies:              1          -line_numbers:          no
-delete:              no         -raw:                  no
-page_breaks:         yes        -use_fortran_controls: no
-indentation:         0          -page_size:
-top_margin:          3          -bottom_margin:       3
-line_length:
-queue_priority:
-pass_thru:           no         -wrap:                 no
                                -notify:                no
```

Issuing the **BLANKS**  request moves the cursor to the space following the `t` in `phone_list`, as shown here.

```
----- print -----
file_names:          phone_list minutes
-queue:              standard
-title:
-destination:
-module:
-device:
-header:
-footer:
-index:
-defer_until:
-interpret_tabs:
-exception_handling:  replace
-copies:              1          -line_numbers:          no
-delete:              no          -raw:                  no
-page_breaks:         yes         -use_fortran_controls: no
-indentation:         0          -page_size:
-top_margin:          3          -bottom_margin:       3
-line_length:
-queue_priority:
-pass_thru:           no          -wrap:                  no
                                -notify:                no
```

Now suppose that you issue the **BLANKS**  request. The cursor moves to the first nonblank character to the right, the `m` in `minutes`.

```
----- print -----
file_names:          phone_list m minutes
-queue:              standard
-title:
-destination:
-module:
-device:
-header:
-footer:
-index:
-defer_until:
-interpret_tabs:
-exception_handling:  replace
-copies:              1          -line_numbers:          no
-delete:              no          -raw:                  no
-page_breaks:         yes         -use_fortran_controls: no
-indentation:         0          -page_size:
-top_margin:          3          -bottom_margin:       3
-line_length:
-queue_priority:
-pass_thru:           no          -wrap:                  no
                                -notify:                no
```

Editing a Command Line or a Display-Form Field Value

There are various requests that allow you to edit either command lines or display-form field values. (Note that you cannot edit cycle fields in display forms.) You can use these requests to delete and/or change characters in a command line or a field value. One request, `INSERT/OVERLAY`, allows you to change the behavior of some editing requests.

Note: The width of a character can vary depending on the default language that your terminal uses. For example, kanji characters are wider than the characters in Latin alphabet No. 1. The examples illustrating the editing requests are based on a terminal using Latin alphabet No. 1. If your terminal uses a language that has wider characters, the examples will look slightly different.

This section describes the following requests.

- `INSERT/OVERLAY`
- `DEL` and `DELETE`
- `BACK_SPACE`
- `CHANGE_CASE`

The `INSERT/OVERLAY` Request

The `INSERT/OVERLAY` request toggles your terminal between Insert mode and Overlay mode. For example, if your terminal is in Overlay mode and you issue the `INSERT/OVERLAY` request, your terminal enters Insert mode. The default setting is Insert mode.

You can determine whether your terminal is in Insert mode or Overlay mode by issuing the `UPDATE_STATUS` request. The word `Insert` or `Overlay` appears in the status display at the bottom of the screen. See “Getting Status Information” in [Chapter 5](#) for an explanation of the status area.

Insert mode and Overlay mode determine how the characters you type are added to a command line or other text. In Insert mode (the default), any character you type is inserted **before** the character at the current cursor position. The character at the current cursor position and any subsequent characters move one position to the right for each character you insert. In Overlay mode, any character you type **overwrites** the character at the current cursor position.

Suppose that you enter the following command line and issue the `←` request to position the cursor on the `l` in `cust_list`.

```
print cust_1list memo.smith
```

Now enter the string `phone`. If your terminal is in Insert mode, the command line looks like this.

```
print cust_phone1list memo.smith
```

However, if your terminal is in Overlay mode, the command line looks like this.

```
print cust_phone memo.smith
```

The **DEL** and **DELETE** Requests

This group of requests includes the **DEL**, **DELETE** **←**, **DELETE** **→**, **DELETE** **WORD**, and **DELETE** **BLANKS** requests. All of these requests perform essentially the same action, deleting one or more characters.

The **DEL** request deletes the **character** on which the cursor is positioned. All characters to the right of the cursor (if any) move one position to the left to fill the gap. This request performs the same action regardless of whether your terminal is in Insert mode or Overlay mode.

The **DELETE** **←** request deletes **all** characters on the current line that are to the left of the cursor. The deleted characters are copied into the insert-saved buffer. If your terminal is in Insert mode, all characters to the right of the cursor move to the left to fill the gap. In Overlay mode, the deleted characters are replaced with spaces. In either mode, the cursor is positioned at the beginning of the line.

The **DELETE** **→** request deletes **all** characters on the current line that are to the right of the cursor. The deleted characters are copied into the insert-saved buffer, and the cursor is positioned at the end of the line. This request performs the same action regardless of whether your terminal is in Insert mode or Overlay mode.

The **DELETE** **WORD** request deletes the **word** on which the cursor is positioned or, if the cursor is not positioned on a word, deletes the word that precedes the cursor. The deleted word is copied into the insert-saved buffer. If your terminal is in Insert mode, all characters to the right of the cursor move to the left to fill the gap. In Overlay mode, the deleted word is replaced with spaces.

For example, suppose that you enter the following command line and issue the **←** request to position the cursor on the first **n** in `-line_numbers`.

```
print backup.89-03-25 -line_numbers -defer_until '11:30 PM'
```

If your terminal is in Insert mode, issuing the **DELETE** **WORD** request deletes the string `line_numbers -` and produces this result.

```
print backup.89-03-25 -defer_until '11:30 PM'
```

However, issuing the **DELETE** **WORD** request when your terminal is in Overlay mode produces this result.

```
print backup.89-03-25 -
```

```
defer_until '11:30 PM'
```

The **DELETE** **BLANKS** request deletes the space on which the cursor is positioned and any spaces adjacent to the cursor. Any nonspace characters to the right of the cursor move to the left to fill the gap.

The **BACK_SPACE** Request

The **BACK_SPACE** request moves the cursor one position to the left and deletes the character in that position. If your terminal is in Insert mode, all characters to the right of the cursor move one position to the left to fill the gap. In Overlay mode, the request replaces the deleted character with a space.

The **BACK_SPACE** request and the **DEL** request are similar in that they both delete one character or space. However, the **BACK_SPACE** request deletes the character immediately preceding the cursor and moves the cursor one position to the left; the **DEL** request deletes the character on which the cursor is positioned and does not move the cursor.

The **CHANGE_CASE** Requests

The **CHANGE_CASE** requests change the case of one or more characters. There are two groups of **CHANGE_CASE** requests.

- The **CHANGE_CASE** **↑** and **CHANGE_CASE** **↓** requests affect only the character on which the cursor is positioned.
- The **WORD** **CHANGE_CASE** **↑**, **WORD** **CHANGE_CASE** **↓**, and **WORD** **CHANGE_CASE** **←** requests affect only the word on which the cursor is positioned.

The **CHANGE_CASE** **↑** and **CHANGE_CASE** **↓** Requests

The **CHANGE_CASE** **↑** request changes the case of the character on which the cursor is positioned from lowercase to uppercase. The **CHANGE_CASE** **↓** request changes the case of the character on which the cursor is positioned from uppercase to lowercase. The cursor then moves one position to the right.

These requests do not affect digits or special characters (for example, the percent sign (%)).

For example, suppose that you enter the following command line and issue the **←** request to position the cursor on the **d** in **testdata1**.

```
print testdata1 output1
```


Issuing the **CHANGE_CASE** **↑** request changes the letter d to uppercase and positions the cursor on the letter a, as shown here.

```
print testDatal output1
```

Positioning the cursor on the D in testData1 (using the **←** request) and issuing the **CHANGE_CASE** **↓** request returns the command line to its original appearance.

```
print testdata1 output1
```

The **WORD** **CHANGE_CASE** Requests

The **WORD** **CHANGE_CASE** **↑** request changes to uppercase all lowercase characters in the word on which the cursor is positioned. The **WORD** **CHANGE_CASE** **↓** request changes to lowercase all uppercase characters in the word on which the cursor is positioned. The cursor then moves to the first character of the next word.

The **WORD** **CHANGE_CASE** **↑** and **WORD** **CHANGE_CASE** **↓** requests do not affect those characters in the word that are already in the desired case, nor do they affect digits or special characters (for example, the number sign (#)).

For example, suppose that you enter the following command line and issue the **←** request to position the cursor on the d in testData1.

```
print testdata1 output1
```

Issuing the **WORD** **CHANGE_CASE** **↑** request changes the word testData1 to uppercase and positions the cursor on the o in output1, as shown here.

```
print TESTDATA1 cursoroutput1
```

Now suppose that you move the cursor back to the word TESTDATA1 using the **←** request. Issuing the **WORD** **CHANGE_CASE** **↓** request returns the command line to its original appearance.

```
print testData1 output1
```

The **WORD** **CHANGE_CASE** **↔** request changes the first character of the word on which the cursor is positioned to uppercase and all other alphabetic characters to lowercase. The cursor

then moves to the first character of the next word. This request does not affect digits or special characters (for example, the underline (_)). If the word on which the cursor is positioned begins with one or more nonalphabetic characters, the request ignores these characters and changes all alphabetic characters to lowercase.

The `[WORD][CHANGE_CASE][←]` request has no affect if the first character of the word on which the cursor is positioned is already uppercase and all other alphabetic characters are lowercase (that is, if the word is already in the form that this request produces).

Suppose that you enter the following command line and issue the `[←]` request to position the cursor on the `D` in `testDATA1`.

```
print testDATA1 output1
```

Issuing the `[WORD][CHANGE_CASE][←]` request changes the first `t` in `testDATA1` to uppercase and changes the remainder of the word to lowercase. The cursor then moves to the `o` in `output1`, as shown here.

```
print Testdata1output1
```

Now suppose that you enter the following command line and issue the `[←]` request to position the cursor on the `D` in `_testDATA1`.

```
print _testDATA1 output1
```

Issuing the `[WORD][CHANGE_CASE][←]` request changes the uppercase characters (the substring `DATA`) to lowercase and positions the cursor on the `o` in `output1`, as shown here.

```
print _testdata1output1
```

The first character of the string `_testDATA1` is not alphabetic; therefore, the request ignores it.

Miscellaneous Requests

This section describes the following requests, which do not fall into any of the categories presented earlier in this chapter.

- `REPEAT_LAST`
- `REDISPLAY`
- `INTERRUPT`

The `REPEAT_LAST` Request

The `REPEAT_LAST` request repeats the previous keystroke (or keystroke sequence). This is especially useful if you want to repeatedly issue a request that requires pressing several keys (for example, the `WORD` `CHANGE_CASE` `↓` request).

For example, suppose that you type the following command on the command line, but you inadvertently type it in uppercase.

```
PRINT CUST_PHONE_LIST -TITLE PHONE_LIST -QUEUE LOCAL -LINE_NUMBERS
+-NO_PAGE_BREAKS -NOTIFY
```

You cannot issue this command in uppercase because the command processor will not recognize the command name or any of the arguments. One way to change the command to lowercase is to issue the `GOTO` `←` request to position the cursor at the beginning of the command line, and then issue the `WORD` `CHANGE_CASE` `↓` request once for each word in the command line. This would require 29 keystrokes: two to position the cursor at the beginning of the command line, and 27 to change the case of each word.

Alternatively, you can delete the command line and retype it in lowercase. This would require even more keystrokes: two to position the cursor at the beginning of the command line, and 90 to retype the line---a total of 92 keystrokes.

A simpler approach is to use the `REPEAT_LAST` request. Issue the `GOTO` `←` request to position the cursor at the beginning of the command line, issue the `WORD` `CHANGE_CASE` `↓` request once for the first word in the command line, and then issue the `REPEAT_LAST` request seven times to perform the same action on the remaining words in the command line. This approach entails a total of 12 keystrokes.

You can also use the `REPEAT_LAST` request in the noncycle fields of a display form.

The `REDISPLAY` Request

The `REDISPLAY` request refreshes the contents of the screen. This request is useful, for example, if you are working at a remote terminal and a portion of the screen is obscured by one or more line errors. (A line error occurs when the module does not receive transmission from your terminal correctly. This results in random characters appearing on the screen that have nothing to do with your work.)

The `INTERRUPT` Request

You use the `INTERRUPT` request to enter window manager mode. See [Chapter 4](#), “The Window Manager,” for a detailed description of this request and for additional information about the window manager.

Chapter 3:

Managing Output

When you issue certain commands, a display of output appears on your screen. This output can be short (for example, an error message or a message indicating that a command has finished executing) or it can be long (for example, a display of the contents of a large file). The output from all commands that you issue is displayed in the output area, which is the top (and largest) area of the three areas into which the screen is divided.

If there is enough room in the output area for all of the command output, it is displayed completely. If there is **not** enough room for all of the output, what will fit is displayed, a pause message appears, and the remaining output waits until you decide what to do with it. You can choose to continue the display of output or you can discard the remaining output. You can also disable the pause mechanism so that subsequent output is displayed completely. This chapter describes the input requests that you use to perform these tasks.

How Output Is Displayed

This section provides two examples that illustrate how output is displayed at two points in the execution of the `list -dirs` command.

In the first example, output is just beginning to appear. Output from a previous `display` command appears at the top of the output area. Also, the user has requested a status display, which appears at the bottom of the screen.

```
first ub      by use_block
first uci     by update_channel_info
first ud      by use_dump
first ul      by unlink
first um      by use_module
first up      by use_partition
first vsa     by verify_system_access
first w       by where
subsequent -m* by -module *
subsequent -on by -module
subsequent -oon by -overseer_module
subsequent -pl by -pause_lines
subsequent -priv by -privileged
subsequent -ttp by -terminal_type
subsequent -st by -statistics
ready 14:34:53
list -dirs (home_dir)>*
Directories: 14
m          12 admin
m          1 cal_1989
█
06-08 14:35 | 21% | Lee_Smith | list | Insert
```

In the second example, the `list -dirs` command has finished executing. Note that there is enough room in the output area for all of the output.

```
subsequent -st      by -statistics
```

```
ready 14:34:53
```

```
list -dirs (home_dir)>*
```

```
Directories: 14
```

```
m      12 admin
```

```
m      1 cal_1989
```

```
m      1 database
```

```
m      2 eng
```

```
m      2 forms
```

```
m      5 macros
```

```
m      6 misc
```

```
m      1 outputs
```

```
m      1 safety_net
```

```
m      4 sysadmin
```

```
m      1 tape_dumps
```

```
m      1 tapes
```

```
m      1 waste_basket
```

```
m      4 workshop
```

```
ready 14:35:07
```

```
█
```

```
06-08 14:35 | 21% | Lee_Smith | list | Insert
```

Displaying and Discarding Paused Output

Suppose that you issue the `list` command to list all of the files in your current directory; however, the complete list does not fit in the window. The following example illustrates what your screen looks like.

```
Files: 25, Blocks: 39
m      1 acctg_memos
m      1 backup_files
m      1 cal_1989
m      1 cust_act
m      1 cust_inact
m      1 cust_phones
m      1 database
m      1 database_admin
m      1 database_user_questions
m      2 eng
m      2 forms
m      5 macros
m      1 memo.chan.89-06-22
m      1 memo.clark.89-03-14
m      1 memo.williams.89-03-14
m      6 misc
m      1 mtg_minutes
m      1 outputs
m      1 safety_net
m      4 sysadmin
--PAUSE--
06-10 09:12 | 21% | Lee_Smith | list | Insert
```

The default pause message is `--PAUSE--`. By default, the window terminal software displays 23 lines of output and then pauses. You can use the `set_terminal_parameters` command to tailor the pause message or the number of lines displayed before the output pauses. For instructions on how to do this, see the description of the `set_terminal_parameters` command in the *VOS Commands Reference Manual (R098)*.

There are three requests that allow you to proceed after the pause message appears. To continue the display of paused output, issue the `↓` or `NEXT_SCREEN` request; to discard paused output, issue the `DISCARD` request. The remainder of this section describes these requests.

The `↓` and `NEXT_SCREEN` Requests

The `↓` and `NEXT_SCREEN` requests, which continue the display of paused output, differ only in how much of the remaining output they display.

- The `↓` request displays the next line of output at the bottom of the output area. To make room for the new line of output, the display scrolls up one line, causing the first line of output to disappear.

- The `NEXT_SCREEN` request displays the next portion of output in the output area. The amount of output displayed is determined in the following manner.
 - If there is not enough room in the output area to display the remaining output, the request displays as much as it can and pauses again.
 - If there **is** enough room in the output area to display the remaining output, the request displays that output and the command finishes executing.

The old output scrolls off the screen to accommodate the new output.

Note: The `NEXT_SCREEN` request name is not shown in a key outline because it does not appear on either the template or a key. For example, on the V102 terminal, you issue the `NEXT_SCREEN` request by pressing `[SHIFT]` `[↓]`. You must hold the `[SHIFT]` key down while you press the `[↓]` key.

If you issue the `[↓]` request when the output pauses (see the previous example), the display changes, as shown here.

```
Files: 25, Blocks: 39
m      1 acctg_memos
m      1 backup_files
m      1 cal_1989
m      1 cust_act
m      1 cust_inact
m      1 cust_phones
m      1 database
m      1 database_admin
m      1 database_user_questions
m      2 eng
m      2 forms
m      5 macros
m      1 memo.chan.89-06-22
m      1 memo.clark.89-03-14
m      1 memo.williams.89-03-14
m      6 misc
m      1 mtg_minutes
m      1 outputs
m      1 safety_net
m      4 sysadmin
m      1 system_events
--PAUSE--
06-10 09:12 | 21% | Lee_Smith | list | Insert
```

The blank line at the top of the output area has scrolled off the screen, and a new line of output has appeared directly above the pause message. The pause message indicates that there is more output to be displayed.

Suppose that you issued the `NEXT_SCREEN` request instead of the `↓` request. The screen would look like this.

```
m      1 cust_inact
m      1 cust_phones
m      1 database
m      1 database_admin
m      1 database_user_questions
m      2 eng
m      2 forms
m      5 macros
m      1 memo.chan.89-06-22
m      1 memo.clark.89-03-14
m      1 memo.williams.89-03-14
m      6 misc
m      1 mtg_minutes
m      1 outputs
m      1 safety_net
m      4 sysadmin
m      1 system_events
m      1 tape_dumps
m      1 tapes
m      1 waste_basket
m      1 workshop
ready  09:12:58
█
06-10 09:12 | 21% | Lee_Smith | list | Insert
```

Because there were fewer lines of remaining output than would fill the output area, the display moved up only the number of lines necessary to display the remaining output and the ready prompt.

Note: You cannot use the `RETURN` request to continue the display of paused output. If you issue the `RETURN` request when there is paused output and the current command line is empty, a message appears in the status area indicating which key you should press to continue the display of paused output. If the message is `Press {next-screen}` to continue from a pause., the `next-screen` sequence is not defined in the input section of the terminal type definition file for your terminal. You should define this sequence in the terminal type definition file.

The `DISCARD` Request

The `DISCARD` request discards any remaining output and replaces the pause message with a new ready prompt and a command line. Anything that you type on the command line while the command is executing is not affected by the `DISCARD` request. For example, if you have begun to type a new command and you issue this request to discard the output from the command that is executing, the window terminal software executes the `DISCARD` request, displays the ready prompt, and then waits for you to finish typing the new command.

Suppose that, after issuing the `list` command, you issue the `DISCARD` request rather than continuing the display of output. The screen now looks like this. (Compare this screen with the first screen in this section.)

```
Files: 25, Blocks: 39
m      1 acctg_memos
m      1 backup_files
m      1 cal_1989
m      1 cust_act
m      1 cust_inact
m      1 cust_phones
m      1 database
m      1 database_admin
m      1 database_user_questions
m      2 eng
m      2 forms
m      5 macros
m      1 memo.chan.89-06-22
m      1 memo.clark.89-03-14
m      1 memo.williams.89-03-14
m      6 misc
m      1 mtg_minutes
m      1 outputs
m      1 safety_net
m      4 sysadmin
ready  09:12:50
█
06-10 09:12 | 21% | Lee_Smith | list | Insert
```

The blank line at the top of the output area has scrolled off the screen, and the line containing the pause message has been replaced by the `ready` prompt and the command line.

Note: You cannot use the `CANCEL` request to discard paused output. If you issue the `CANCEL` request when there is paused output and the current command line is empty, a message appears in the status area indicating which key you should press to discard the paused output. If the message is `Press {abort-output} to continue from a pause.`, the `abort-output` sequence is not defined in the input section of the terminal type definition file for your terminal. You should define this sequence in the terminal type definition file.

Disabling the Pause Mechanism

You can disable the pause mechanism so that command output can scroll continuously up the screen. For example, you may want to save command output in a file for later analysis (using the `start_logging` and `stop_logging` commands) rather than view it on the screen. (The `start_logging` and `stop_logging` commands are described in the *VOS Commands Reference Manual (R098)*.)

To disable the pause mechanism, issue the `NO_PAUSE` request when the command begins to execute. The `NO_PAUSE` request disables the pause mechanism temporarily; it is disabled only while the command is executing and is then re-enabled. If you want to execute several

commands and prevent the output from pausing while **any** of the commands are executing, you can do one of two things.

- Issue the `NO_PAUSE` request for each command.
- Issue the `display_terminal_parameters` command to determine the number of pause lines set for your terminal. Then, issue the command `set_terminal_parameters -pause_lines 0`. Output from subsequent commands will scroll up the screen without pausing. When you are ready to stop the continuous scrolling, reset the number of pause lines to the original value by issuing the command `set_terminal_parameters -pause_lines n` (where *n* is the original number of pause lines).

Chapter 4:

The Window Manager

When you first log in at your terminal, you start one login process. This process opens one primary window in which you can execute commands, use forms, run application programs, and so on. The window terminal software allows you to start additional login processes on your terminal, which means that one terminal can function as though it were several terminals. Each new login process opens a primary window in which you can execute commands or perform other activities. The activities you perform in a primary window are, with one minor exception, independent from the activities in other primary windows. The exception concerns retrieving stored input; for details, see “Retrieving Input After Issuing a Command” in [Chapter 2](#).

Regardless of how many primary windows you have created, only one of them, the *current primary window*, appears on the screen at a time. You can work interactively only in the current primary window, even though other primary windows exist and may be active. For example, you can use a form interactively in the current primary window, while in a second primary window an application program is compiling, and in a third, an application program performs calculations.

Because you can have multiple primary windows open at the same time, the window terminal software provides a mechanism called the *window manager*. The window manager monitors all primary windows and provides a means for manipulating them. Among the actions you can request are opening new primary windows, cycling among a group of primary windows to make a different primary window the current window, and interrupting activities in primary windows in various ways (for example, breaking or stopping a process, and deleting a primary window). To use window manager requests, you must enter window manager mode, as described later in this chapter. After issuing the window manager request(s), you can leave window manager mode and resume your activities.

This chapter describes the requests you use to enter and leave window manager mode and to manipulate primary windows. There are three groups of requests.

- You can issue the `INTERRUPT` request at operating system command level to enter window manager mode. You can only issue this request **before** entering window manager mode.

- You can issue the following requests whether or not your terminal is in window manager mode. These requests may function differently when issued in window manager mode rather than on the command line or using a display form.
 - BREAK
 - `CYCLE` and `CYCLE_BACK`
 - `UPDATE_STATUS` and `CLEAR_STATUS`
 - `REDISPLAY`
- You can issue the following requests, which are available **only** in window manager mode.
 - `LEAVE_WINDOW_MANAGER`
 - `LOGIN_PROCESS`
 - `BREAK_PROCESS`
 - `STOP_PROCESS`

This chapter also briefly describes the operating system `logout` command that you use to log out of primary windows.

Entering and Leaving Window Manager Mode

This section describes how to enter and leave window manager mode.

Entering Window Manager Mode

To enter window manager mode, issue the `INTERRUPT` request from operating system command level or issue the `BREAK` request from operating system command level or from an application program. While your terminal is in window manager mode, you can use any of the requests described in this chapter, except for the `INTERRUPT` request.

When you enter window manager mode, a status display appears at the bottom of the screen (if one is not already present). To remind you that you are now in window manager mode, the display contains the words `Window Manager`.

```
06-08 09:57 | 21% | Lee_Smith | ready | Insert | Window Manager
```

Leaving Window Manager Mode

To leave window manager mode, issue the `LEAVE_WINDOW_MANAGER` request. On most terminal keyboards, you issue this request by pressing the `CANCEL` or `ENTER` key. Once you leave window manager mode, you return to your current primary window. The current primary window may be the primary window from which you entered window manager mode or it may be a different primary window, depending on the requests you issued while in window manager mode.

If there was no status display before you entered window manager mode, the status display created by the window manager disappears when you leave window manager mode. If there **was** a status display before you entered window manager mode, when you leave window manager mode, only the words `Window Manager` disappear from the status display.

Creating a New Primary Window

To create a new primary window, you must first enter window manager mode by issuing the `INTERRUPT` request from operating system command level or by issuing the `BREAK` request from operating system command level or from an application program. Then, you issue the `LOGIN_PROCESS` request by pressing the `(F1)` key (on most terminals). The `LOGIN_PROCESS` request starts a new process that opens a new primary window and displays it on the screen. The new primary window becomes the current primary window, and you **leave** window manager mode.

Issuing the `LOGIN_PROCESS` request is similar to issuing the operating system `login` command to log in to a subprocess. (See the *VOS Commands Reference Manual (R098)* for a description of the `login` command.) The primary difference is that in using the `LOGIN_PROCESS` request, the original process and the new process run concurrently; that is, you do not need to log out of the new process in order to work in the original process.

Notes:

1. If all of the primary windows are currently executing commands or other input, the `LOGIN_PROCESS` request must wait to be processed until a primary window is not busy.
2. Stratus recommends that you use the `LOGIN_PROCESS` request to start a new process rather than the operating system `login` command.

For example, suppose that you have entered window manager mode and the current primary window looks like this.

```

w      1 announce_shutdown.cm
w      1 contents.cm
w      1 create_alias_links.cm
w      1 delete_alias_links.cm
w      1 delete_carefully.cm
w      1 display_both_access_lists.cm
w      1 display_reminders.cm
w      1 dk2.cm
w      1 do_multiple.cm
w      1 edit_abbreviations.cm
w      1 edit_nonwords.cm
w      1 edit_phone_list.cm
w      1 edit_reminders.cm
w      1 find_future_day.cm
w      1 get_rid_of_accesses.cm
w      1 get_titles.cm
w      1 get_titles.ufi
w      1 give_full_access.cm
--PAUSE--
display_reminders
edit_abbreviations
change_current_dir work_in_progress
pll test_prog
bind test_prog
06-08 09:57 | 21% | Lee_Smith | list | Overly | Window Manager

```

Now issue the `LOGIN_PROCESS` request. The window manager starts a new login process for you and opens a new primary window, which looks like this. (You do not have to specify your user name or password when you issue the `LOGIN_PROCESS` request. The window manager obtains this information from your current login process and logs you in automatically.)

```
Lee_Smith.Sales logged in on %s1#d0 at 89-06-09 09:58:01 EDT.  
ready 09:58:01  
█
```

As part of the login process, the `start_up.cm` command macro in your home directory is executed. Therefore, if the `start_up.cm` command macro contains commands that produce output on the screen, you may see more than just the `ready` message when you create a new primary window. The current directory in the new primary window is the same as the current directory in the previous primary window.

The new primary window becomes the current primary window, and the window that was previously the current primary window moves to the pool of existing primary windows. Conceptually, the previous window is now **below** the new window. The primary windows available to you are maintained in order, like a stacked deck of cards. For example, suppose that you have three primary windows, A, B, and C. The current primary window is window B. Conceptually, window B is on top of the stack, so the order of windows in the stack is B, C, A. If you enter window manager mode and create a new primary window, D, that window is added to the stack on top of window B. The order of windows in the stack is now D, B, C, A.

There is no limit on the number of primary windows you can create at one time on your terminal; however, there is a limit on the amount of memory allocated to your terminal by the module. Each time you create a new primary window, the terminal uses more of this memory. When all of the memory has been used, you must log out of a window (as described later in this chapter) before you can create another window. The status display at the bottom of the screen contains an entry indicating how much memory has been used.

Suppose that you have only one primary window open and you enter window manager mode. The status display might look like this.

```
06-08 09:57 | 18% | Lee_Smith | ready | Insert | Window Manager
```

The 18% entry indicates that 18% of the memory allocated to your terminal has been used. Now suppose that you create a new primary window. The next time the status display is updated, it might look like this.

```
06-08 09:57 | 21% | Lee_Smith | ready | Insert
```

You have used an additional 3% of the allocated memory to create the new primary window, for a total of 21%. (Note that the words `Window Manager` have disappeared because creating the new primary window took you out of window manager mode.)

See [Chapter 5](#) for a detailed description of the status area.

The next section describes how to access a primary window other than the current primary window.

Moving among Primary Windows

As noted earlier, you can work interactively only in the current primary window, even though other primary windows exist and may be active. However, you may want to leave the current primary window and work in another primary window. For example, suppose that you start an application program that will take some time to process, and you do not want to wait until it finishes to work on a project in another primary window. Or, suppose that you want to check the status of a command that is executing in another primary window, issue additional commands, and then return to your original window.

In either case, you move to another primary window by following these steps.

1. Issue the `[INTERRUPT]` request or the `BREAK` request to enter window manager mode.
2. Issue the `[CYCLE]` or `[CYCLE_BACK]` request to cycle through the primary windows until you reach the desired primary window.
3. Issue the `[ENTER]` or `[CANCEL]` request to leave window manager mode.

The window you have selected is now the current primary window in which you can work interactively.

The `[CYCLE]` request cycles through the primary windows one window at a time, starting with the current primary window. As you cycle to another primary window, the previous window moves to the bottom of the stack of windows. For example, suppose that you have three primary windows, A, B, and C, and that window A is your current primary window. Issuing the `[CYCLE]` request brings you to window B and issuing the request again brings you to window C. Issuing the `[CYCLE]` request a third time brings you back to window A.

The `CYCLE_BACK` request cycles through the primary windows in the opposite direction from the `CYCLE` request. Using the same example, if window A is your current primary window, issuing the `CYCLE_BACK` request brings you to window C.

Note: If all of the primary windows are executing commands or other input, the `CYCLE` or `CYCLE_BACK` request must wait to be processed until a primary window is not busy.

When you cycle among primary windows, some of the windows may appear to be blank, even though the status display indicates that an application program is executing. If the window is blank, it is because the application program executing in that window is not designed to handle the `REDISPLAY` request generated by the window terminal software. After leaving window manager mode, you should issue the application program's `REDISPLAY` request to repaint the screen. For example, for a window in which the Emacs text editor is executing, you press `CTRL V` to repaint the screen.

To illustrate how cycling to another primary window relates to other activities, consider the following examples.

Suppose that you start an application program in one of your primary windows that will take some time to execute, and you do not want to wait until it finishes to work on a project in another primary window. To move to this window, perform the following steps.

1. Issue the `INTERRUPT` request or the `BREAK` request to enter window manager mode.
2. Issue the `CYCLE` or `CYCLE_BACK` request to cycle to another primary window.
3. Issue the `LEAVE_WINDOW_MANAGER` request (if necessary) to leave window manager mode. Remember that if you create a new primary window, you leave window manager mode automatically.
4. Work in the new current primary window while the application program is running in the other primary window.

Suppose that you want to check the status of a command that is executing in another primary window, issue additional commands, and then return to your original window. To do so, perform the following steps.

1. Issue the `INTERRUPT` request or the `BREAK` request to enter window manager mode.
2. Issue the `CYCLE` or `CYCLE_BACK` request to move to the primary window in which the command is executing.
3. Issue the `LEAVE_WINDOW_MANAGER` request to leave window manager mode. The primary window to which you cycled is now the current primary window.
4. Check on the progress of the command and issue additional commands.
5. Issue the `INTERRUPT` request or the `BREAK` request to re-enter window manager mode.

6. Issue the `CYCLE` or `CYCLE_BACK` request to cycle to the original primary window.
7. Issue the `LEAVE_WINDOW_MANAGER` request to leave window manager mode. The original primary window is now the current primary window.

Canceling Activity in the Current Primary Window

You can cancel activity in the current primary window in one of three ways.

- Issue the `BREAK_PROCESS` request to suspend activity in the window.
- Issue the `STOP_PROCESS` request to stop the login process associated with the window.
- Issue the operating system `logout` command to log out of the window.

The following subsections describe each of these actions.

Breaking a Process in the Current Primary Window

Your terminal must be in window manager mode in order for you to use the `BREAK_PROCESS` request.

The `BREAK_PROCESS` request **suspends** activity in the current primary window. You issue the `BREAK_PROCESS` request by pressing the `[F6]` key (on most terminals). When you issue this request, you leave window manager mode and the standard break prompt is displayed.

```
BREAK
Request? (stop, continue, debug, keep, login, re-enter)
```

Your response to this prompt determines what action the operating system takes regarding the `BREAK_PROCESS` request.

- A response of `stop` tells the operating system to terminate the program abnormally, discard all pending input, and return the process to operating system command level.
- A response of `continue` tells the operating system to resume processing as if the interruption did not occur.
- A response of `debug` invokes the symbolic debugger, which allows you to examine the program and set breakpoints. You can restart the program from the debugger or you can terminate it.
- A response of `keep` saves an interrupted executable image of the program in a file that can be debugged later.
- A response of `login` creates a new login process for you, which is a subprocess of the current process.
- A response of `re-enter` returns control to the interrupted program at a predetermined execution point, provided that the program supports the `re-enter` request.

For more information on the debugger and on logging in to a subprocess, see the descriptions of the `debug` and `login` commands, respectively, in the *VOS Commands Reference Manual (R098)*. For more information about the `stop`, `continue`, `keep`, and `re-enter` requests, see the *VOS Commands User's Guide (R089)*.

Stopping the Login Process Associated with the Current Primary Window

Your terminal must be in window manager mode in order for you to use the `STOP_PROCESS` request.

To stop (rather than suspend) activity in the current primary window, issue the `STOP_PROCESS` request to stop the login process associated with that primary window. You issue the `STOP_PROCESS` request by pressing the `[F7]` key (on most terminals).

When you issue this request, the operating system stops the process and performs a number of cleanup activities. The primary window associated with the login process that you stopped is no longer available. The primary window immediately below the primary window whose process you stopped is now the current primary window.

Issuing the `STOP_PROCESS` request while in window manager mode is equivalent to issuing an operating system `stop_process` command while at operating system command level. The `stop_process` command is described in the *VOS Commands Reference Manual (R098)*.

Logging Out of the Current Primary Window

To log out of the current primary window, issue the operating system `logout` command from operating system command level. You **cannot** log out of a primary window while your terminal is in window manager mode.

The `logout` command logs you out of the current primary window **only**; it does not log you out of all primary windows. If you log out of the current primary window while other primary windows are still open, the window terminal software displays the primary window immediately below the window you logged out of and makes that window the current primary window.

Redisplaying the Current Primary Window

The `[REDISPLAY]` request refreshes the contents of the current primary window. If all of the primary windows are executing commands or other input, the `[REDISPLAY]` request must wait to be processed until a primary window is not busy. See “Miscellaneous Requests” in [Chapter 2](#) for more information about this request.

Chapter 5:

Getting Help System and Status Information

While working at your terminal, you can request two types of information:

- help system information about commands and arguments
- status information about the process running in the current primary window.

This chapter describes the input requests that you use to display and manage this information. It also describes the messages that you can receive in the status area from another user, an application program, or the operating system.

Getting Help System Information

The operating system provides a *help system*, which contains interactive online help messages. For information about a specific command or argument, you issue the `HELP` request from the command line or display form.

See the *VOS System Administrator's Guide (R012)* for more information about the help system.

Note: The `HELP` request is not available when your terminal is in window manager mode.

Getting Help about a Command

To get help about a command, type the command name (with or without arguments) on the command line, then issue the `HELP` request. This request provides a brief description of the command and returns you to command level.

You can also display help system information by issuing the `help` command on the command line. For a description of the `help` command, see the *VOS Commands Reference Manual (R098)*.

For example, suppose that you type `print` on the command line and then issue the `HELP` request. The help system displays the following information.

```
Print a file or files.
```

To get help about multiple commands, type the command names on the command line, using a semicolon (;) to separate each one. Then, issue the `HELP` request. A description of each command appears, in the order in which you typed them on the command line.

For example, suppose that you type the following command names on the command line and then issue the `[HELP]` request.

```
print; list
```

The following information is displayed.

```
Print a file or files.
```

```
Lists the contents of a directory.
```

Getting Help about an Argument

Getting help about an argument is easiest when you use the display form of a command. Simply invoke the display form of a command, position the cursor on the relevant field, and issue the `[HELP]` request. A brief description of the argument appears on the screen. If there is additional information about this argument, you are prompted to ask for it. When you have viewed all the information you want, you can return to the display form and issue or cancel the command.

Suppose that you want information about the `-queue` argument of the `print` command. To do this, follow these steps.

1. Type `print` on the command line.
2. Issue the `[DISPLAY_FORM]` request.
3. Position the cursor on the field for the `-queue` argument.
4. Issue the `[HELP]` request.

The display form of the command disappears, and the following information is displayed.

```
Type the name of the queue to the printer you want to use, if other
than the standard queue. The command uses the standard queue
unless you ask for a different one. Use the display_print_status
command
to find out the names of the queues on your system.
```

```
Press HELP for more detail or press RETURN to continue with the
command.
SELECTION:
```

If you issue another `[HELP]` request, the following additional information is displayed.

```
A print queue is a file that holds requests for print jobs.
Each printer takes print requests off a print queue. Each queue holds
requests for only one printer, although different queues may hold
requests
for the same printer.
```

(Continued on next page)

- o To find out how many print requests are in a queue: `list_print_requests`
- o To see where a print job is in the queue: `list_print_requests`
- o To change the position of a print request in the queue, cancel it with the `cancel_print_requests` command and issue the print command again with a higher priority option. (See the help information for the priority option.)
- o To get information about which queues go to which printers, and the state of the printers: `display_print_status`
- o To get information about a print queue: `display_print_defaults`

Press RETURN to continue with the command.

SELECTION:

At this point, issuing the `RETURN` request returns you to the display form of the print command.

You can also get help about a required argument by omitting the argument when you issue the command on the command line. When the software prompts you for an argument value, issue the `HELP` request to display help system information.

Note: This method of getting help applies **only** to required arguments. You cannot use this method to get help about optional arguments or arguments that have a default value.

Getting Status Information

The status area – the bottom line on the screen – can contain either a status display of information about the process running in the current primary window, or a message from another user, an application program, or the operating system.

All information and messages in the status area are displayed in inverse video (if your terminal supports this feature).

The Contents of the Status Area

A status display appears when you issue the `UPDATE_STATUS` request. A status display is provided automatically when you enter window manager mode.

Here is a sample status display.

```
06-08 09:57 | 21% | Lee_Smith | list | Overly | Window Manager
```

This status display contains the following information:

- an abbreviated version of the current date and time. The software displays the month and day (the year is omitted), and the hour and minutes (the seconds are omitted).
- a number followed by a percent sign (%), to indicate how much of the memory allocated to your terminal has been used. In this example, 21% of the memory allocated to your

terminal has been used. See “Creating a New Primary Window” in [Chapter 4](#) for more information about this field.

- the `person_name` portion of your user name. Your group name is not included.
- the name of the current primary window. This is the name of the command that is currently executing, or the word `ready` if no command is executing. In this example, the `list` command is executing.
- either the word `Insert` or the word `Overlay`, to indicate whether the terminal is in Insert mode or Overlay mode. In this example, the terminal is in Overlay mode.
- if your terminal is in window manager mode, the words `Window Manager`.

The status display is updated periodically by the window terminal software.

The messages that appear in the status area apply either to the device or to the current primary window. Device-specific messages include messages that are sent using the `send_message` command, notifications of print and batch job completion, and notifications of mail arrival. (See the *VOS Commands Reference Manual (R098)* for a description of the `send_message` command.) Here is an example of a message that was sent using the `send_message` command.

```
Robin_Clark: Department meeting in 10 mknutes in the conference room.
```

Window-specific messages include error messages, informational messages, and messages generated by the application running in the current primary window. For example, if you are working in the display form of a command and you issue a delete request from a cycle field, the following message is displayed.

```
You may only CYCLE through items in this field.
```

Device-specific messages take precedence over window-specific messages. For example, if a window-specific message is displayed in the status area and the terminal receives a device-specific message, that message overwrites the window-specific message. However, if a device-specific message is displayed in the status area and the terminal receives a window-specific message, the device-specific message is not overwritten. In this case, the window-specific message is **behind** the device-specific message.

If the terminal receives a message of the same type as the currently displayed message, the new message overwrites the existing message.

Updating and Clearing the Status Area

If you receive a message that exceeds one line, only the first line is displayed in the status area. To view the next line, issue the `UPDATE_STATUS` request. The next line is displayed in the status area, overwriting the previous line. Once the previous line is overwritten, you cannot retrieve it.

To clear a one-line message from the status area, issue the `CLEAR_STATUS` request. To clear a multiline message, issue one `CLEAR_STATUS` request for each line of the message.

To display a window-specific message that is behind a device-specific message, you can issue either the `UPDATE_STATUS` request or the `CLEAR_STATUS` request.

Appendix A:

Summary of Input Requests

The three tables in this appendix summarize the input requests. Table A-1 lists and describes the requests available on the command line. Table A-2 describes the requests available when using display forms. Table A-3 describes the requests available in window manager mode.

If the behavior of the request varies depending on whether the terminal is in Insert mode or Overlay mode, the differences are described.

Table A-1. Command-Line Requests (Page 1 of 3)

Request	Description
<code>BACK_SPACE</code>	Deletes the character immediately preceding the cursor and moves the cursor one position to the left. If the cursor is at the beginning of the current command line, no action occurs. In Insert mode, all characters to the right of the cursor move one position to the left to fill the gap. In Overlay mode, the deleted character is replaced with a space.
<code>BACK_TAB</code>	Moves the cursor to the beginning of the current command line.
<code>BLANKS</code> <code>←</code>	Moves the cursor to the left to the space immediately following the first nonblank character. If no blank characters immediately precede the cursor, the cursor does not move.
<code>BLANKS</code> <code>→</code>	Moves the cursor to the right to the first nonblank character. If no blank characters immediately follow the cursor, the cursor does not move.
<code>BREAK</code> (Typically, <code>BREAK</code> , <code>CTRL</code> <code>BREAK</code> , or <code>CTRL</code> <code>C</code>)	Places the terminal in window manager mode. If you issue this request a second time, it suspends activity in the current primary window.
<code>CANCEL</code> <code>CANCEL</code>	Cancels the current command line.
<code>CHANGE_CASE</code> <code>↓</code>	Changes to lowercase the character on which the cursor is positioned and moves the cursor one position to the right.
<code>CHANGE_CASE</code> <code>↑</code>	Changes to uppercase the character on which the cursor is positioned and moves the cursor one position to the right.
<code>CLEAR_STATUS</code>	Clears the status area. For a multiline message, issue this request once for each line of the message. If there is another message behind the currently displayed message, issue this request to display that message.
<code>DEL</code>	Deletes the character on which the cursor is positioned and moves all characters to the right of the cursor one position to the left. Deletes the character on which the cursor is positioned and moves all characters to the right of the cursor one position to the left.
<code>DELETE</code> <code>BLANKS</code>	Deletes the space on which the cursor is positioned and any spaces adjacent to the cursor. Any nonspace characters to the right of the cursor move to the left to fill the gap.
<code>DELETE</code> <code>←</code>	Deletes all characters to the left of the cursor. The deleted characters are copied into the insert-saved buffer. In Insert mode, all characters to the right of the cursor move to the left to fill the gap. In Overlay mode, the deleted characters are replaced with spaces. In either mode, the cursor is positioned at the beginning of the current command line.
<code>DELETE</code> <code>→</code>	Deletes all characters to the right of the cursor and positions the cursor at the end of the current command line. The deleted characters are copied into the insert-saved buffer.

Table A-1. Command-Line Requests (Page 2 of 3)

Request	Description
<code>DELETE</code> <code>WORD</code>	Deletes the word on which the cursor is positioned or, if the cursor is not positioned on a word, deletes the word that precedes the cursor. The deleted word is copied into the insert-saved buffer. In Insert mode, all characters to the right of the cursor move to the left to fill the gap. In Overlay mode, the deleted word is replaced with spaces.
<code>DISPLAY_FORM</code>	Produces the display form of a command
<code>↓</code>	When at a pause, displays one additional line of paused output in the output area.
<code>ENTER</code>	Issues the command on the command line.
<code>GOTO</code> <code>←</code>	Moves the cursor to the beginning of the current command line.
<code>GOTO</code> <code>→</code>	Moves the cursor to the end of the current command line (the position immediately following the last character in the command line).
<code>HELP</code>	Displays information about a command or argument.
<code>INSERT_DEFAULT</code>	Retrieves the contents of the insert-default buffer (the last command processed).Retrieves the contents of the insert-default buffer (the last command processed).
<code>INSERT_SAVED</code>	Retrieves the contents of the insert-saved buffer.
<code>INSERT/OVERLAY</code>	Toggles the terminal between Insert mode and Overlay mode.
<code>INTERRUPT</code>	Places the terminal in window manager mode.
<code>←</code>	Moves the cursor one position to the left. If the cursor is at the beginning of the current command line, no action occurs.
<code>NEXT_SCREEN</code>	Displays the next portion of paused output in the output area.
<code>NO_PAUSE</code>	Turns off the pause mechanism for the duration of the current command, allowing continuous scrolling.
<code>REDISPLAY</code>	Redisplays the contents of the screen.
<code>REPEAT_LAST</code>	Repeats the previous keystroke sequence.
<code>RETURN</code>	Issues the command on the command line.
<code>→</code>	Moves the cursor one position to the right. If the cursor is at the end of the current command line, no action occurs.
<code>TAB</code>	Moves the cursor to the next tab stop.
<code>UPDATE_STATUS</code>	Produces and updates a status display or displays a subsequent line of the current message or status display. If there is another message behind the currently displayed message, issue this request to display that message.

Table A-1. Command-Line Requests (Page 3 of 3)

Request	Description
<code>WORD</code> <code>CHANGE_CASE</code> <code>↓</code>	Changes to lowercase the word on which the cursor is positioned and moves the cursor to the beginning of the next word. If the cursor is not positioned on a word, this request changes to lowercase the word that precedes the cursor.
<code>WORD</code> <code>CHANGE_CASE</code> <code>←</code>	Changes to uppercase the first letter of the word on which the cursor is positioned, changes the remainder of the word to lowercase, and then moves the cursor to the beginning of the next word. If the cursor is not positioned on a word, this request changes to uppercase the first letter of the word that precedes the cursor.
<code>WORD</code> <code>CHANGE_CASE</code> <code>↑</code>	Changes to uppercase the word on which the cursor is positioned and moves the cursor to the beginning of the next word. If the cursor is not positioned on a word, this request changes to uppercase the word that precedes the cursor.
<code>WORD</code> <code>←</code>	Moves the cursor to the first character of the current word or, if the cursor is already positioned on the first character of the current word or on a space between words, moves the cursor to the first character of the word to the left of the current word. If the cursor is at the beginning of the current command line, no action occurs.
<code>WORD</code> <code>→</code>	Moves the cursor to the first character of the word to the right of the current word. If the cursor is positioned on the last word in the current command line, this request moves the cursor one position to the right of that word.

Table A-2. Display-Form Requests (Page 1 of 4)

Request	Description
<code>BACK_SPACE</code>	Deletes the character immediately preceding the cursor and moves the cursor one position to the left. If the cursor is at the beginning of the current field, no action occurs. In Insert mode, all characters to the right of the cursor move one position to the left to fill the gap. In Overlay mode, the deleted character is replaced with a space.
<code>BACK_TAB</code>	Moves the cursor to the previous field.
<code>BLANKS</code> <code>←</code>	Moves the cursor to the left to the space following the first nonblank character. If no blank characters immediately precede the cursor, the cursor does not move.
<code>BLANKS</code> <code>→</code>	Moves the cursor to the right to the first nonblank character. If no blank characters immediately follow the cursor, the cursor does not move.

Table A-2. Display-Form Requests (Page 2 of 4)

Request	Description
BREAK (Typically, <input type="button" value="BREAK"/> , <input type="button" value="CTRL"/> <input type="button" value="BREAK"/> , or <input type="button" value="CTRL"/> <input type="button" value="C"/>)	Places the terminal in window manager mode. If you issue this request a second time, it suspends activity in the current primary window.
<input type="button" value="CANCEL"/>	Cancels the current display form.
<input type="button" value="CHANGE_CASE"/> <input type="button" value="↓"/>	Changes to lowercase the character on which the cursor is positioned and moves the cursor one position to the right.
<input type="button" value="CHANGE_CASE"/> <input type="button" value="↑"/>	Changes to uppercase the character on which the cursor is positioned and moves the cursor one position to the right.
<input type="button" value="CLEAR_STATUS"/>	Clears the status area. For a multiline message, issue this request once for each line of the message. If there is another message behind the currently displayed message, issue this request to display that message.
<input type="button" value="CYCLE"/>	In a cycle field, displays the next value.
<input type="button" value="CYCLE_BACK"/>	In a cycle field, displays the previous value.
<input type="button" value="DEL"/>	Deletes the character on which the cursor is positioned and moves all characters to the right of the cursor one position to the left.
<input type="button" value="DELETE"/> <input type="button" value="BLANKS"/>	Deletes the space on which the cursor is positioned and any spaces adjacent to the cursor. Any nonspace characters to the right of the cursor move to the left to fill the gap.
<input type="button" value="DELETE"/> <input type="button" value="←"/>	Deletes all characters to the left of the cursor in the current field. The deleted characters are copied into the insert-saved buffer. In Insert mode, all characters to the right of the cursor move to the left to fill the gap. In Overlay mode, the deleted characters are replaced with spaces. In either mode, the cursor is positioned at the beginning of the current field.
<input type="button" value="DELETE"/> <input type="button" value="→"/>	Deletes all characters to the right of the cursor in the current field and positions the cursor at the end of the field. The deleted characters are copied into the insert-saved buffer.
<input type="button" value="DELETE"/> <input type="button" value="WORD"/>	Deletes the word on which the cursor is positioned or, if the cursor is not positioned on a word, deletes the word that precedes the cursor. The deleted word is copied into the insert-saved buffer. In Insert mode, all characters to the right of the cursor move to the left to fill the gap. In Overlay mode, the deleted word is replaced with spaces.
<input type="button" value="↓"/>	Moves the cursor to the field in the next line closest to the current cursor position.
<input type="button" value="ENTER"/>	Issues the command from the display form.
<input type="button" value="GOTO"/> <input type="button" value="←"/>	Moves the cursor to the beginning of the current field.
<input type="button" value="GOTO"/> <input type="button" value="↓"/>	Moves the cursor to the last field in the display form.

Table A-2. Display-Form Requests (Page 3 of 4)

Request	Description
<code>GOTO</code> <code>→</code>	Moves the cursor to the end of the current field.
<code>GOTO</code> <code>↑</code>	Moves the cursor to the first field in the display form.
<code>HELP</code>	Displays information about an argument.
<code>INSERT_DEFAULT</code>	Inserts the default value into a field.
<code>INSERT_SAVED</code>	Inserts text from the insert-saved buffer into a field.
<code>INSERT/OVERLAY</code>	Toggles the terminal between Insert mode and Overlay mode.
<code>INTERRUPT</code>	Places the terminal in window manager mode.
<code>←</code>	In a noncycle field, moves the cursor one position to the left. If the cursor is at the beginning of a field, no action occurs. If the cursor is in a cycle field, this request is equivalent to the <code>CYCLE_BACK</code> request.
<code>REDISPLAY</code>	Redisplays the contents of the screen.
<code>REPEAT_LAST</code>	Repeats the previous keystroke sequence.
<code>RETURN</code>	Moves the cursor to the first field on the next line.
<code>→</code>	In a noncycle field, moves the cursor one position to the right. If the cursor is at the end of a field, no action occurs. If the cursor is in a cycle field, this request is equivalent to the <code>CYCLE</code> request.
<code>TAB</code>	Moves the cursor to the next field.
<code>TAB_STOP</code> <code>←</code>	Moves the cursor to the previous field.
<code>TAB_STOP</code> <code>→</code>	Moves the cursor to the next field.
<code>↑</code>	Moves the cursor to the field in the previous line closest to the current cursor position.
<code>UPDATE_STATUS</code>	Produces and updates a status display or displays a subsequent line of the current message or status display. If there is another message behind the currently displayed message, issue this request to display that message.
<code>WORD</code> <code>CHANGE_CASE</code> <code>↓</code>	Changes to lowercase the word on which the cursor is positioned and moves the cursor to the beginning of the next word. If the cursor is not positioned on a word, this request changes to lowercase the word that precedes the cursor.
<code>WORD</code> <code>CHANGE_CASE</code> <code>←</code>	Changes to uppcase the first letter of the word on which the cursor is positioned and moves the cursor to the beginning of the next word. If the cursor is not positioned on a word, this request changes the case of the first letter of the word that precedes the cursor.

Table A-2. Display-Form Requests (Page 4 of 4)

Request	Description
<code>WORD</code> <code>CHANGE_CASE</code> <code>↑</code>	Changes to uppercase the word on which the cursor is positioned and moves the cursor to the beginning of the next word. If the cursor is not positioned on a word, this request changes to uppercase the word that precedes the cursor.
<code>WORD</code> <code>←</code>	Moves the cursor to the first character of the current word or, if the cursor is already positioned on the first character of the current word or on a space between words, moves the cursor to the first character of the word to the left of the current word. If the cursor is at the beginning of the field, no action occurs.
<code>WORD</code> <code>→</code>	Moves the cursor to the first character of the word to the right of the current word. If the cursor is positioned on the last word in the field, this request moves the cursor one position to the right of that word.

Table A-3. Window Manager Requests

Request	Description
<code>BREAK</code>	Suspends activity in the current primary window.
<code>BREAK_PROCESS</code> (typically, <code>F6</code>)	Suspends activity in the current primary window.
<code>CANCEL</code>	See the description of the <code>LEAVE_WINDOW_MANAGER</code> request in this table.
<code>CLEAR_STATUS</code>	Clears the status area. For a multiline message, issue this request once for each line of the message. If there is another message behind the currently displayed message, issue this request to display that message.
<code>CYCLE</code>	Cycles to the next primary window.
<code>CYCLE_BACK</code>	Cycles to the previous primary window.
<code>ENTER</code>	See the description of the <code>LEAVE_WINDOW_MANAGER</code> request in this table.
<code>LEAVE_WINDOW_MANAGER</code> (typically, <code>CANCEL</code> or <code>ENTER</code>)	Exits the terminal from window manager mode.
<code>LOGIN_PROCESS</code> (typically, <code>F1</code>)	Starts a new login process, opens a new primary window, and exits the terminal from window manager mode.
<code>REDISPLAY</code>	Redisplays the contents of the screen.
<code>STOP_PROCESS</code> (typically, <code>F7</code>)	Stops the login process associated with the current primary window.
<code>UPDATE_STATUS</code>	Produces and updates a status display or displays a subsequent line of the current message or status display. If there is another message behind the currently displayed message, issue this request to display that message.

Note: The `INTERRUPT` request, which you issue at operating system command level or in a display form, places the terminal in window manager mode. This request is not available when your terminal is in window manager mode.

Appendix B:

Function/Special Key Maps for Stratus Terminals

Key maps are tables that associate keystroke sequences with requests. This appendix provides key maps for the Stratus V101, V102, and V103 terminals. (The V103 terminal comes with either an ASCII keyboard or an EPC keyboard.) Tables B-1 and B-2 list the keystroke sequences for the V101 keyboard and V102 keyboard, respectively. Tables B-3 and B-4 list the keystroke sequences for the V103 ASCII keyboard and V103 EPC keyboard, respectively.

The first column of each table lists the request names. The request names are **not** shown in key outlines (unlike their presentation in the rest of this manual) so that you can distinguish them from keystroke sequences more easily.

The second column of each table lists the keystroke sequences that produce the requests. Sequences using function keys (such as **F11**) are shown with the function-key legend rather than the corresponding template word so that you can use these tables without a template.

Some requests can be produced using more than one keystroke sequence. For these requests, each keystroke sequence is listed on a separate line.

If a keystroke sequence contains any of the following keys, you must hold that key down while you press the next key in the sequence.

- **SHIFT** or **Shift**
- **CTRL** or **Ctrl**
- **Funct**

Table B-1. Keystroke Sequences for the V101 Keyboard (*Page 1 of 3*)

Request	Keystroke Sequence
BACK SPACE	BACK_SPACE
BACK TAB	BACK_TAB
BLANKS ←	SHIFT F0 ←
BLANKS →	SHIFT F0 →
BREAK	CTRL BREAK CTRL C
BREAK_PROCESS	F6
CANCEL	F17
CHANGE CASE ↓	F9 ↓

Table B-1. Keystroke Sequences for the V101 Keyboard (Page 2 of 3)

Request	Keystroke Sequence
CHANGE CASE ↑	[F9] [↑]
CLEAR STATUS	[SHIFT] [STATUS]
CYCLE	[F16]
CYCLE BACK	[SHIFT] [F16]
DEL	[DEL]
DELETE BLANKS	[F12] [SHIFT] [F0]
DELETE ←	[F12] [←]
DELETE →	[F12] [→]
DELETE WORD	[F12] [F0]
DISCARD	[SHIFT] [F12]
DISPLAY FORM	[F18]
↓	[↓]
ENTER	[ENTER]
GOTO ↓	[GOTO] [↓]
GOTO ←	[GOTO] [←]
GOTO →	[GOTO] [→]
GOTO ↑	[GOTO] [↑]
HELP	[SHIFT] [F8]
INSERT DEFAULT	[SHIFT] [F1]
INSERT SAVED	[F11]
INSERT/OVERLAY	[CTRL] [F16]
INTERRUPT	[CTRL] [F0]
LEAVE_WINDOW_MANAGER	[ENTER] [F17]
←	[←]
LOGIN_PROCESS	[F1]
NEXT_PROCESS	[SHIFT] [↓]
NO PAUSE	[SHIFT] [F17]
REDISPLAY	[SHIFT] [F18]
REPEAT LAST	[CTRL] [F7]
RETURN	[RETURN]

Table B-1. Keystroke Sequences for the V101 Keyboard (Page 3 of 3)

Request	Keystroke Sequence
→	[→]
STOP_PROCESS	[F7]
TAB	[TAB]
TAB STOP ←	[F1] [←]
TAB STOP →	[F1] [→]
↑	[↑]
UPDATE STATUS	[STATUS]
WORD CHANGE CASE ←	[F0] [F9] [←]
WORD CHANGE CASE ↑	[F0] [F9] [↑]
WORD ←	[F0] [←] [F0] [SHIFT] [←]
WORD →	[F0] [→] [SHIFT] [→]

Table B-2. Keystroke Sequences for the V102 Keyboard (Page 1 of 3)

Request	Keystroke Sequence
BACK SPACE	[BACK_SPACE]
BACK TAB	[BACK_TAB]
BLANKS ←	[SHIFT] [F0] [←]
BLANKS →	[SHIFT] [F0] [→]
BREAK	[CTRL] [BREAK] [CTRL] [C]
BREAK_PROCESS	[F6]
CANCEL	[F20] [SHIFT] [CE]
CHANGE CASE ↓	[F9] [↓]
CHANGE CASE ↑	[F9] [↑]
CLEAR STATUS	[SHIFT] [STATUS]
CYCLE	[F19]
CYCLE BACK	[F16]
DEL	[DEL]
DELETE BLANKS	[F12] [SHIFT] [F0]
DELETE ←	[F12] [←]
DELETE →	[F12] [→]

Table B-2. Keystroke Sequences for the V102 Keyboard (Page 2 of 3)

Request	Keystroke Sequence
DELETE WORD	[F12] [F0]
DISCARD	[SHIFT] [F12]
DISPLAY FORM	[F21]
↓	[↓]
ENTER	[ENTER]
GOTO ↓	[GOTO] [↓]
GOTO ←	[GOTO] [←]
GOTO →	[GOTO] [→]
GOTO ↑	[GOTO] [↑]
HELP	[SHIFT] [F8]
INSERT DEFAULT	[SHIFT] [F11]
INSERT SAVED	[F11]
INSERT/OVERLAY	[F22]
INTERRUPT	[SHIFT] [F22]
LEAVE_WINDOW_MANAGER	[ENTER] [F20]
←	[←]
LOGIN_PROCESS	[F1]
NEXT_PROCESS	[SHIFT] [↓]
NO PAUSE	[F17]
REDISPLAY	[F18]
REPEAT LAST	[CTRL] [F2]
RETURN	[RETURN]
→	[→]
STOP_PROCESS	[F7]
TAB	[TAB]
TAB STOP ←	[SHIFT] [F1] [←] [SHIFT] [BACK_TAB]
TAB STOP →	[SHIFT] [F1] [→] [SHIFT] [TAB]
↑	[↑]
UPDATE STATUS	[STATUS]
WORD CHANGE CASE ↓	[F0] [F9] [↓]

Table B-2. Keystroke Sequences for the V102 Keyboard (Page 3 of 3)

Request	Keystroke Sequence
WORD CHANGE CASE ←	[F0] [F9] [←]
WORD CHANGE CASE ↑	[F0] [F9] [↑]
WORD ←	[F0] [←] [F0] [SHIFT] [←]
WORD →	[F0] [→] [SHIFT] [→]

Table B-3. Keystroke Sequences for the V103 ASCII Keyboard (Page 1 of 3)

Request	Keystroke Sequence
BACK SPACE	[Back_Space]
BACK TAB	[Shift_Tab]
BLANKS ←	[Shift] [F16] [←]
BLANKS →	[Shift] [F16] [→]
BREAK	[Ctrl] [Break] [Ctrl] [C]
BREAK_PROCESS	[F6]
CANCEL	[F18]
CHANGE CASE ↓	[F9] [↓]
CHANGE CASE ↑	[F9] [↑]
CLEAR STATUS	[Shift] [Status]
CYCLE	[F17]
CYCLE BACK	[Shift] [F17]
DEL	[Del]
DELETE BLANKS	[F12] [Shift] [F16]
DELETE ←	[F12] [←]
DELETE →	[F12] [→]
DELETE WORD	[F12] [F12]
DISCARD	[Shift] [F12]
DISPLAY FORM	[F19]
↓	[↓]
ENTER	[Enter]
GOTO ↓	[Home] [↓]
GOTO ←	[Home] [←]

Table B-3. Keystroke Sequences for the V103 ASCII Keyboard (Page 2 of 3)

Request	Keystroke Sequence
GOTO →	[Home] [→]
GOTO ↑	[Home] [↑]
HELP	[Shift] [F8]
INSERT DEFAULT	[Shift] [F11]
INSERT SAVED	[F11]
INSERT/OVERLAY	[F20]
INTERRUPT	[Shift] [F20]
LEAVE_WINDOW_MANAGER	[Enter] [F18]
←	[←]
LOGIN_PROCESS	[F1]
NEXT_PROCESS	[Shift] [↓]
NO PAUSE	[Shift] [F18]
REDISPLAY	[Shift] [F19]
REPEAT LAST	[Shift] [F2]
RETURN	[Return]
→	[→]
SCROLL ↓	[F1] [↓]
SCROLL ←	[F1] [←]
SCROLL →	[F1] [→]
SCROLL ↑	[F1] [↑]
SCROLL SHIFT ↓	[F1] [Shift] [↓]
SCROLL SHIFT ←	[F1] [Shift] [←]
SCROLL SHIFT →	[F1] [Shift] [→]
SCROLL SHIFT ↑	[F1] [Shift] [↑]
STOP_PROCESS	[F7]
TAB	[Tab]
TAB STOP ←	[Shift] [F1] [←]
TAB STOP →	[Shift] [F1] [→]
↑	[↑]
UPDATE STATUS	[Status]

Table B-3. Keystroke Sequences for the V103 ASCII Keyboard (Page 3 of 3)

Request	Keystroke Sequence
WORD ←	F16 ← Shift ←
WORD →	F16 → Shift →
WORD CHANGE CASE ↓	F16 F9 ↓
WORD CHANGE CASE ←	F16 F9 ←
WORD CHANGE CASE ↑	F16 F9 ↑

Table B-4. Keystroke Sequences for the V103 EPC Keyboard (Page 1 of 3)

Request	Keystroke Sequence
BACK SPACE	← Back_Space
BACK TAB	↑Shift Tab
BLANKS ←	↑Shift F7 ←
BLANKS →	↑Shift F7 →
BREAK	Ctrl Pause/Break Ctrl C
BREAK_PROCESS	F6
CANCEL	* (on the keypad)
CHANGE CASE ↓	Ctrl F2 ↓
CHANGE CASE ↑	Ctrl F2 ↑
CLEAR STATUS	Ctrl F9
CYCLE	F12 Alt C
CYCLE BACK	↑Shift F12 Alt B
DEL	Delete Del (on the keypad)
DELETE BLANKS	F8 ↑Shift F7
DELETE ←	F8 ←
DELETE →	F8 →
DELETE WORD	F8 F7
DISCARD	Ctrl F8
DISPLAY FORM	- (on the keypad)
↓	↓
ENTER	Enter (on the keypad) + (on the keypad)

Table B-4. Keystroke Sequences for the V103 EPC Keyboard (Page 2 of 3)

Request	Keystroke Sequence
GOTO ↓	↑Shift End Alt G ↓
GOTO ←	Home Alt G ←
GOTO →	End Alt G →
GOTO ↑	↑Shift Home Alt G ↑
HELP	↑Shift F2
INSERT DEFAULT	↑Shift F10
INSERT SAVED	F10 ↑Shift Insert
INSERT/OVERLAY	Insert
INTERRUPT	↑Shift Delete ↑Shift Del (on the keypad) Alt I
LEAVE_WINDOW_MANAGER	Enter (on the keypad) + (on the keypad) + (on the keypad) F18
←	←
LOGIN_PROCESS	F1
NEXT_PROCESS	↑Shift ↓ Page_Down
NO PAUSE	↑Shift * (on the keypad) Alt P
REDISPLAY	↑Shift - (on the keypad) Alt R
REPEAT LAST	Shift F1
RETURN	Enter ←
→	→
SCROLL ↓	F5 ↓
SCROLL ←	F5 ←
SCROLL SHIFT ↓	F5 ↑Shift ↓ F5 Page_Down
SCROLL SHIFT ←	F5 ↑Shift ←
SCROLL SHIFT →	F5 ↑Shift →
SCROLL SHIFT ↑	F5 ↑Shift ↑ F5 Page_Up
SCROLL →	F5 →
SCROLL ↑	F5 ↓
STOP_PROCESS	F7

Table B-4. Keystroke Sequences for the V103 EPC Keyboard (Page 3 of 3)

Request	Keystroke Sequence
TAB	Tab
TAB STOP ←	Ctrl F7 ←
TAB STOP →	Ctrl F7 →
↑	↑
UPDATE STATUS	↑Status F9
WORD ←	F7 ← ↑Shift ←
WORD →	F7 → ↑Shift →
WORD CHANGE CASE ↓	F7 Ctrl F2 ↓
WORD CHANGE CASE ←	F7 Ctrl F2 ←
WORD CHANGE CASE ↑	F7 Ctrl F2 ↑

Glossary

alphanumeric character

The characters a through z (uppercase and lowercase) and the numbers 0 through 9.

American Standard Code for Information Interchange (ASCII)

In the Stratus internal character coding system, the half of the 8-bit code page with code values in the range 00-7F (hexadecimal), representing the American Standard Code for Information Interchange.

application

A program (or set of programs) that runs on a module.

argument

A character string that specifies how a command, request, subroutine, or function is to be executed.

ASCII

See **American Standard Code for Information Interchange (ASCII)**.

ASCII string

A character string that contains only ASCII data. The string does not contain any shift characters or any characters from the right-hand graphic sets; however, it may contain generic input or output sequences.

asynchronous communication

A type of communication that is characterized by a start/stop transmission mode. Each character transmitted in start/stop mode is preceded by a start bit and followed by one or more stop bits. Since there is no clocking mechanism, the interval between transmission of characters varies. This method of transmitting data is designed to accommodate transmissions where data is sent sporadically (for example, from terminals).

attribute

1. A characteristic of a field, especially the visual characteristics: high intensity, low intensity, blinking, blanked, inverse video, and underlined.
2. A video parameter of a terminal. Six attributes are defined: high intensity, low intensity, blinking, blanked, inverse video, and underlined.

background text

Text within a form, but not in a field. Background text includes titles and labels that identify fields.

block mode

A communications mode that sends text entered at the keyboard only to the screen until the terminal is signaled to send the information to the computer.

block-mode terminal

A terminal that sends text entered at the keyboard to the computer in blocks, rather than as soon as the keys are pressed. Typically, the characters are not sent until the ENTER key is pressed.

break

A signal (or to send a signal) that interrupts a program being executed and places the process executing the program at break level.

break level

The state that results when the user gives a break signal and it is handled by the operating system. When an interactive process is placed at break level, you are prompted to choose one of the following courses of action.

stop	Stops the program and returns the process to command level.
continue	Continues to execute the program from the point at which the break signal was sent.
debug	Invokes the VOS symbolic debugger.
keep	Saves the current state of the program in a keep module, which is a file named <i>program_name</i> .kp. This allows you to examine the state of the program at a later time with the debugger.
login	Creates a new interactive process while preserving the current process.
re-enter	If the program contains a handler for the re-enter condition, signals the condition and passes control to its condition handler.

When a batch process is placed at break level, the operating system automatically creates a keep module.

character

A symbol, such as a letter of the alphabet or a numeral, or a control signal, such as a carriage return or a backspace. Characters are represented in electronic media by character codes.

character string

An ordered set of characters from one or more character sets. The length of a character string depends on the size of each character (one or more bytes, depending on the

character set), the number of characters in the string, and the use of single-shift and locking-shift characters within the string. Character strings are evaluated from left to right.

command

A program invoked from command level, either interactively or as a statement in a command macro.

command area

The portion of the terminal screen that displays input to the terminal (for example, command lines). The command area comprises the line(s) immediately above the status area (which is the last line on the screen). The number of lines in the command area increases as necessary to accommodate multiple input lines and decreases when those input lines are processed. See also **output area** and **status area**.

command level

The state at which you can issue commands to the command processor or run application programs.

command line

A set of one or more commands, separated by semicolons.

command name

The name of a program or command macro that can be called/invoked as a command.

command processor

The part of the operating system that accepts and executes commands. The command processor replaces abbreviations and evaluates command functions in a command before loading the program specified in the command.

command string

A command name and any command arguments you specify.

configuration table

One of the table files that the operating system uses to identify the elements of a system or a network. For example, the file `devices.table` contains information about each device present in the system; the file `disks.table` contains information about each of the disks present in the system; and the file `modules.table` contains information about each module in the system.

current primary window

The primary window currently displayed on the terminal screen. If the terminal is not in window manager mode, all input goes to this primary window. If the terminal **is** in window manager mode, all input goes to the window manager and may affect this primary window.

cursor

A marker on the display (typically, a blinking rectangular block or an underline) that shows where the next character you type will appear.

cycle field

A field that has a specific list of possible values. The user can type the first letter of a value in a cycle field or can change the displayed value by issuing the `CYCLE` or `CYCLE_BACK` request or by pressing the `←` or `→` key.

cycle list

The list of possible values for a cycle field.

cycle values

Two or more predefined values that you display in the display form by issuing the `CYCLE` or `CYCLE_BACK` request or by pressing the `←` or `→` key.

default value

The value that the operating system uses if a specific value is not supplied.

device

Any hardware component that can be referenced and used by the system or users of the system and that is defined in the device configuration table. Terminals, printers, tape drives, and communications lines are devices. The term *logical device* indicates that a device can be a software entity; for example, a virtual terminal is a logical device.

device configuration table

The file `devices.table`, which contains information about each device present in a system. See also **configuration table**.

EBCDIC

See **Extended Binary Coded Decimal Interchange Code (EBCDIC)**.

editor

A program used to create and modify certain types of files.

For text files, Stratus provides two screen editors (designed for video display terminals) and one line editor (designed for printing terminals). You invoke the screen editors by typing `emacs` or `edit` on the command line; you invoke the Line Editor by typing `line_edit` on the command line.

For FMS forms, Stratus provides the Forms Editor.

error message

A character string that is associated with an error code.

execute

1. To process an executable statement at run time.
2. To run a program.

Extended Binary Coded Decimal Interchange Code (EBCDIC)

A coded character set consisting of 8-bit coded characters.

file

A set of records or bytes stored as a unit on disk or tape. A disk file has a path name that identifies it as a unique entity in the system's directory hierarchy. Attributes of a disk file, such as its size and when it was created, are maintained in the directory containing the file.

function key

One of the keys on the keyboard that generally produces a nonprinting character or a sequence of nonprinting and printing characters. Examples include the numbered function keys (`F1`), (`F2`), and so on), the arrow keys, and the `BACK_SPACE` key.

generic input request

See **input request**.

group

A set of users whose directories are contained in the same group directory.

group name

The name that identifies the group directory to which a person registered to use the system belongs. A group name is one of the two parts of a user name.

help message

Explanatory text displayed to the user.

input request

A request that the operating system either executes on the terminal screen or sends to a program when it recognizes the input characters that define the request in the terminal type definition file. The `TAB` and `BACK_SPACE` requests are examples of input requests.

insert-default buffer

A command-line storage area that stores the text that you enter by issuing the `ENTER` request or the `DISPLAY_FORM` request.

Insert mode

A terminal mode in which the characters you type are inserted at the current cursor location. Characters that follow the cursor move to the right (that is, they are not overwritten). See also **Overlay mode**.

insert-saved buffer

A command-line storage area that stores the text that you enter by issuing the **ENTER** request or the **DISPLAY_FORM** request, or that you delete by issuing the **DELETE** **←**, **DELETE** **→**, or **DELETE** **WORD** request.

inverse video

A terminal mode in which the color of the characters on the screen and the color of the screen background are reversed from the default.

mode

An operating state that controls how the terminal operates or how it reacts to commands.

module

A single Stratus computer. A module is the smallest hardware unit of a system capable of executing the user's process.

output area

The portion of the terminal screen that displays output (for example, command output). The output area extends from the top of the primary window down to the command area. The number of lines in the output area increases or decreases in response to activity in the command area. See also **command area** and **status area**.

Overlay mode

A terminal mode in which each character you type replaces the character at the current cursor position. See also **Insert mode**.

parameter

A value upon which a procedure operates. The actual value is supplied when the procedure is called.

person name

The name that identifies a person to the system. When logging in, you must supply either the person name or the alias by which the system can identify you.

A person name is one of the two parts of a user name.

primary window

An area comprising the entire terminal screen that displays the output of a single process or application program running the window terminal software. When the user logs in, the window terminal software creates a primary window. The user or the application program can create additional primary windows and can move among all existing primary windows.

Primary windows overlap. Conceptually, they are ordered from top to bottom, where the currently displayed primary window is on top of the other primary windows.

process

The sequence of states of the hardware and software during the execution of the user's application programs. When you log in, the operating system creates a process in which you can control the execution of your application programs. This process can create other processes at your request. A process is always in one of three states: running, waiting, or ready.

prompt

A message to the terminal user requesting further information before processing can continue.

screen

The display area of a video display terminal.

startup command macro

A command macro named `start_up.cm`, which is located in the user's home directory. When the user logs in, the operating system executes the command macro before placing the user's process at command level or placing the process in the subsystem specified in the user's registration entry or in the `-subsystem` argument of the `login` command.

status area

The line at the bottom of the terminal screen that displays information about the current status of the terminal and messages sent to the user. See also **command area** and **output area**.

string

A character string or bit string. See also **character string**.

subwindow

A portion of a primary window that is controlled by the application program running in the primary window. There are three types of subwindows: simple sequential I/O, formatted sequential I/O, and forms. All I/O goes to the subwindow that is designated as the current subwindow.

Subwindows may or may not overlap, depending on how the application program creates and arranges them within the primary window.

table

A collection of related information that is stored in a database. See also **table file**.

table file

A file that the operating system creates by means of the `create_table` command. All of the records in a table file have the same format and are subdivided into fixed-length fields. The name of a table file always ends in the suffix `.table`.

terminal

A device used for input and output. Most modern terminals have screens and keyboards; however, printing terminals (with keyboards) also fall into the class of terminal. If a terminal has its own hard-copy printer connected to it, that printer is considered part of the terminal.

terminal type definition file

In the operating system, a file that defines the relationship between generic input and output requests and terminal-specific input and output sequences. The terminal type definition file is the input that the `define_terminal_type` command uses to create the terminal type tables for a terminal type.

user

An individual who is registered to use a system. A user is specified by a user name, which is composed of a person name and a group name.

user name

An identifier that is composed of a person name and a group name.

value

A measurable, describable, storable quantity that is associated with a constant, variable, or expression.

Virtual Operating System

See **VOS operating system**.

VOS operating system

The virtual operating system of a Stratus computer.

window

A portion of a terminal screen or primary window. If the former, the window is a primary window. It comprises the entire screen and displays the output of a single process or application program running the window terminal software. If the latter, the window is a subwindow. It is controlled by the application program running in the primary window.

Primary windows overlap completely; subwindows may or may not overlap, depending on the application program.

See also **primary window** and **subwindow**.

window manager

A mechanism that monitors all primary windows and provides a means for manipulating them. Among the actions the user can request are opening additional primary windows, cycling among a group of primary windows, and interrupting activities in primary windows in various ways.

window manager mode

A mode of terminal operation in which the terminal recognizes only window manager requests. See also **window manager requests**.

window manager requests

A set of input requests that manipulate primary windows. These requests perform activities such as opening additional primary windows, cycling among a group of primary windows, and interrupting activities in primary windows in various ways (for example, breaking or stopping a process, and deleting a primary window). The requests can only be used when the terminal is in window manager mode.

Window manager requests must be defined as such in the appropriate terminal type definition file.

window terminal software

A loadable operating system device driver that supports character-mode terminals (for example, video display terminals) in a window environment. Its layered architecture separates the application interface and terminal code from the code associated with the communications medium.

Index

Misc.

BACK_SPACE input request, 2-22
BACK_TAB input request, 2-8, 2-18
CANCEL input request, 2-4, 2-10, 4-2
CLEAR_STATUS input request, 5-5
DEL input request, 2-21
DELETE **WORD** input request, 2-21
DISCARD request, 3-6
DISPLAY FORM input request, 2-4
ENTER input request, 2-10, 4-2
HELP input request, 5-1, 5-2
INSERT/OVERLAY input request, 2-20
INTERRUPT input request, 2-26, 4-2
NO_PAUSE input request, 3-8
REDISPLAY input request, 2-25, 4-8
RETURN input request, 2-9
TAB_STOP **<** input request, 2-8
TAB input request, 2-8, 2-18
UPDATE_STATUS input request, 5-3, 5-5
input request, 3-4
↓ input request, 2-9
Continuing the display of paused output
input request, 3-4
Input requests , 3-4
↓, 2-9
Moving the cursor among display-form fields
↓, 2-9
< input request, 2-9, 2-16
Moving the cursor within text
< input request, 2-16
> input request, 2-9, 2-16
Input requests
>, 2-9, 2-16
Moving the cursor within text
> input request, 2-16
↑ input request, 2-9
Input requests
↑, 2-9
Moving the cursor among display-form fields
↑, 2-9

A

Adding field values in a display form, 2-9
Application-level input requests, 1-3

B

BLANKS **<** input request, 2-18
Moving the cursor within text
BLANKS **<** input request, 2-18
BLANKS **>** input request, 2-18
Input requests
BLANKS **>**, 2-18
Moving the cursor within text
BLANKS **>** input request, 2-18
BREAK_PROCESS input request, 4-7
continue response, 4-7
debug response, 4-7
keep response, 4-7
login response, 4-7
re-enter response, 4-7
stop response, 4-7
Breaking a process in a primary window, 4-7

C

Canceling activity in the current primary
window, 4-7
Capitalizing words, 2-24
CHANGE_CASE **↓** input request, 2-22
Editing text
CHANGE_CASE **↓** input request, 2-22
Input requests
CHANGE_CASE **↓**, 2-22
CHANGE_CASE **↑** input request, 2-22
Editing text
CHANGE_CASE **↑** input request, 2-22
Input requests
CHANGE_CASE **↑**, 2-22
Changing
between Insert mode and Overlay
mode, 2-20
field values in a display form, 2-9
the case of single characters, 2-22
the case of words, 2-23

- Clearing a status display, 5-5
- Command area, 2-2
 - command-line behavior, 2-13
 - entering multiple commands, 2-13
- Command-level input requests, 1-3
- Commands
 - interrupting execution of, 2-15
 - issuing on the command line, 2-4
 - issuing using the display form, 2-10
 - retrieving, 2-10
- Continuing the display of paused output, 3-4
 - NEXT_SCREEN input request, 3-5
 - one line at a time, 3-4
 - one screen at a time, 3-5
- Creating new primary windows, 4-3
- Current primary window, 1-6, 4-1
- `[CYCLE]` input request, 2-9, 4-6
- `[CYCLE_BACK]` input request, 2-9, 4-6
- Cycling
 - among field values in a display form, 2-9
 - among primary windows, 4-5

D

- Defining input requests, 1-2
- `[DELETE]` `[BLANKS]` input request, 2-22
- Editing text
 - `[DELETE]` `[BLANKS]` input request, 2-22
- Input requests
 - `[DELETE]` `[BLANKS]`, 2-22
- `[DELETE]` `[←]` input request, 2-21
- Editing text
 - `[DELETE]` `[←]` input request, 2-21
- `[DELETE]` `[→]` input request, 2-21
- Editing text
 - `[DELETE]` `[→]` input request, 2-21
- Input requests
 - `[DELETE]` `[→]`, 2-21
- Editing text
 - `[DELETE]` `[WORD]` input request, 2-21
- Deleting
 - a portion of a line, 2-21
 - a single character, 2-21, 2-22
 - a status display, 5-5
 - a word, 2-21
 - blanks, 2-22
- Deleting characters
 - in command lines, 2-21
 - in display-form fields, 2-21
- Disabling the pause mechanism, 3-7
- Discarding paused output, 3-6
- Display forms, 1-7
 - adding field values, 2-9
 - changing field values, 2-9

- cycling among field values, 2-9
- entering arguments on the command line, 2-5
- invoking, 2-4, 2-7
- moving the cursor from field to field, 2-8
- `display_terminal_parameters`
 - operating system command, 3-8
- Displaying
 - paused output, 3-4

E

- Editing text, 2-20
 - `[BACK_SPACE]` input request, 2-22
 - `[DEL]` input request, 2-21
 - `[INSERT/OVERLAY]` input request, 2-20
 - capitalizing words, 2-24
 - changing between Insert mode and Overlay mode, 2-20
 - changing the case of single characters, 2-22
 - changing the case of words, 2-23
 - deleting a portion of a line, 2-21
 - deleting a single character, 2-21, 2-22
 - deleting blanks, 2-22
 - deleting words, 2-21
 - Insert mode, 2-20
 - Overlay mode, 2-20
- Entering window manager mode, 2-26, 4-2
- Exiting window manager mode, 4-2

F

- Field values (display form)
 - adding, 2-9
 - changing, 2-9
 - cycling among, 2-9
 - retrieving, 2-10

G

- Getting help system information, 5-1
 - operating system command arguments, 5-2
 - operating system commands, 5-1
- Getting status information, 5-3
- `[GOTO]` `[↓]` input request, 2-9
- Input requests
 - `[GOTO]` `[↓]`, 2-9
- Moving the cursor among display-form fields
 - `[GOTO]` `[↓]`, 2-9
- `[GOTO]` `[←]` input request, 2-16
- Moving the cursor within text
 - `[GOTO]` `[←]` input request, 2-16
- `[GOTO]` `[→]` input request, 2-16
- Input requests
 - `[GOTO]` `[→]`, 2-16

Moving the cursor within text

`GOTO` `→` input request, 2-16

`GOTO` `↑` input request, 2-9

Input requests

`GOTO` `↑`, 2-9

Moving the cursor among display-form fields

`GOTO` `↑`, 2-9

H

help operating system command, 5-1

Help system, 5-1

I

Input requests

`←`, 2-9, 2-16

`BACK_SPACE`, 2-22

`BACK_TAB`, 2-8, 2-18

`BLANKS` `←`, 2-18

`CANCEL`, 2-4, 2-10, 4-2

`CLEAR_STATUS`, 5-5

`CYCLE_BACK`, 2-9, 4-6

`CYCLE`, 2-9, 4-6

`DEL`, 2-21

`DELETE` `←`, 2-21

`DELETE` `WORD`, 2-21

`DISCARD`, 3-6

`DISPLAY_FORM`, 2-4

`ENTER`, 2-10, 4-2

`GOTO` `←`, 2-16

`HELP`, 5-1, 5-2

`INSERT_DEFAULT`, 2-10

`INSERT_SAVED`, 2-10

`INSERT/OVERLAY`, 2-20

`INTERRUPT`, 2-26, 4-2

`NO_PAUSE`, 3-8

`REDISPLAY`, 2-25, 4-8

`RETURN`, 2-9

`TAB_STOP` `←`, 2-8

`TAB`, 2-8, 2-18

`UPDATE_STATUS`, 5-3, 5-5

`WORD` `←`, 2-16

application-level, 1-3

`BREAK_PROCESS`, 4-7

command-level, 1-3

defining, 1-2

for the command line, A-1

for the display form, A-1

issuing, 1-3

`LOGIN_PROCESS`, 4-3

`NEXT_SCREEN`, 3-5

request-name format, 1-4

special, 1-3

`STOP_PROCESS`, 4-8

summary of, A-1

Insert-default buffer, 2-10

`INSERT_DEFAULT` input request, 2-10

Insert mode, 2-20

Insert-saved buffer, 2-10

`INSERT_SAVED` input request, 2-10

Interrupting command execution, 2-15

Issuing commands, 2-4

using the display form, 2-10

Issuing input requests, 1-3

K

Key maps, B-1

L

Leaving window manager mode, 4-2

Line errors, 2-25

Logging in, 4-1

Logging out of a primary window, 4-8

`LOGIN_PROCESS` input request, 4-3

Login processes, 4-1

logout operating system command, 4-8

M

Memory used by primary windows, 4-4, 5-4

Messages

generated by applications, 5-4

multiline, 5-5

Moving among primary windows, 4-5

Moving the cursor among display-form fields

`BACK_TAB`, 2-8

`RETURN`, 2-9

`TAB_STOP` `←`, 2-8

`TAB`, 2-8

Moving the cursor within text, 2-16

`BACK_TAB` input request, 2-18

`TAB` input request, 2-18

Multiple primary windows, 4-1

N

`NEXT_SCREEN` input request, 3-5

O

Output

continuing the display of paused output, 3-4, 3-5

discarding paused output, 3-6

paused, 3-4, 3-6

Output area, 2-3

Overlay mode, 2-20

P

- Pause messages, 3-4
 - `NO_PAUSE` input request, 3-8
 - disabling the pause mechanism, 3-7
- Paused output, 3-4
 - continuing the display of, 3-4
 - discarding, 3-6
- Primary windows, 1-5
 - breaking processes in, 4-7
 - canceling activity in, 4-7
 - creating, 4-3
 - current, 1-6, 4-1
 - cycling among, 4-5
 - logging out of, 4-8
 - memory usage, 4-4, 5-4
 - moving among, 4-5
 - multiple, 1-6, 4-1
 - stopping processes in, 4-8
 - suspending activity in, 4-7
 - window manager activities, 4-1
- Producing a status display, 5-3

R

- Redisplaying
 - the current primary window, 4-8
 - the screen, 2-25
- Refreshing the contents of the screen, 2-25, 4-8
- Removing line errors, 2-25, 4-8
- Requesting a status display, 5-3
- Retrieving
 - display-form field values, 2-10
 - input after issuing a command, 2-10

S

- Screen layout, 2-1
 - command area, 2-2
 - output area, 2-3
 - status area, 2-3
- `set_terminal_parameters` operating
 - system command, 3-4, 3-8
- Setting pause messages, 3-4
- Special input requests, 1-3
- Status area, 2-3, 5-3, 5-5
- Status display, 5-3
 - clearing, 5-5
 - requesting, 5-3
 - updating, 5-5
- `STOP_PROCESS` input request, 4-8
- Stopping a process in a primary window, 4-8
- Subwindows, 1-7
- Suspending activity in primary windows, 4-7

System requirements

- VOS operating system release number, 1-1
- input-request definitions, 1-2
- terminal configuration, 1-2
- terminal type, 1-2

T

- `TAB_STOP` `→` input request, 2-8
- Input requests
 - `TAB_STOP` `→`, 2-8
- Moving the cursor among display-form fields
 - `TAB_STOP` `→`, 2-8
- Terminal type definition file, 1-2

U

- Updating a status display, 5-5
- Using window manager requests, 1-5

W

- Window manager, 4-1
- Window manager mode, 4-1
 - `CANCEL` input request, 4-2
 - `ENTER` input request, 4-2
 - `INTERRUPT` input request, 4-2
 - entering, 2-26, 4-2
 - exiting, 4-2
 - leaving, 4-2
- Window manager requests
 - using, 1-5
- Window terminal software, 1-1
 - features of, 1-1
- Windows
 - current primary, 1-6
 - primary, 1-5, 4-1, 4-3, 4-4, 4-5, 4-7, 4-8, 5-4
 - subwindows, 1-7
- `WORD` `←` input request, 2-16
- Moving the cursor within text
 - `WORD` `←` input request, 2-16
- `WORD` `→` input request, 2-17
- Input requests
 - `WORD` `→`, 2-17
- Moving the cursor within text
 - `WORD` `→` input request, 2-17
- `WORD` `CHANGE_CASE` `↓` input request, 2-23
- Editing text
 - `WORD` `CHANGE_CASE` `↓` input request, 2-23
- Input requests
 - `WORD` `CHANGE_CASE` `↓`, 2-23
 - `WORD` `CHANGE_CASE` `←` input request, 2-24

Editing text

`WORD` `CHANGE_CASE` `<-` input
request, 2-24

Input requests

`WORD` `CHANGE_CASE` `<-`, 2-24

`WORD` `CHANGE_CASE` `↑` input request, 2-23

Editing text

`WORD` `CHANGE_CASE` `↑` input request, 2-23

Input requests

`WORD` `CHANGE_CASE` `↑`, 2-23

