

**Rational Environment**  
**Basic Operations**

**Facit Terminal**

Copyright © 1985, 1986, 1987 by Rational

Document Control Number: 8001A-51 (803-002318)

Rev. 4.0, November 1985

Rev. 4.1, December 1985

Rev. 4.2, March 1986

Rev. 4.3, July 1986

Rev. 5.0, July 1987 (Delta)

This document subject to change without notice.

Note the Reader's Comments form on the last page of this book, which requests the user's evaluation to assist Rational in preparing future documentation.

Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).

Rational and R1000 are registered trademarks and Rational Environment and Rational Subsystems are trademarks of Rational.

Rational  
1501 Salado Drive  
Mountain View, California 94043

## Contents

<b>Chapter 1. Logging In and Out</b>	<b>1</b>
Logging In	1
Logging Out	1
Saving Changes	2
<b>Chapter 2. Getting Help</b>	<b>3</b>
Getting Help on Help	3
Getting Help on a Specific Item	3
Getting Help on Keys	4
Displaying Ada Specifications	4
Displaying the Help Window	4
Getting Help on Errors	5
<b>Chapter 3. Executing Commands</b>	<b>7</b>
Creating and Executing a Command Window Program	7
Expanding a Command Window	7
Shrinking a Command Window	7
Getting Command Completion	8
Moving to the Next Prompt or Underline	8
Moving to the Previous Prompt or Underline	8
Turning Off a Prompt	8
Reexecuting the Same Command	8
Changing and Reexecuting a Command	9
Entering a New Command in the Same Command Window	9
Clearing a Command Window of Unneeded Text	9
Going Back to Previous Commands	9
Getting the Parameters of a Command Bound to a Key	9

<b>Chapter 4. Managing Windows</b>	11
Finding a Window Using the Window Directory	11
Deleting Windows from the Window Directory	11
Moving between Windows	11
Expanding a Window	12
Shrinking a Window	12
Expanding Current Window to Include Next Frame	12
Expanding Current Window to Include Previous Frame	12
Transposing Windows	12
Realigning the Windows on the Screen	13
Removing a Window	13
Locking a Window on the Screen	13
Unlocking a Window on the Screen	14
Scrolling the Image	14
<b>Chapter 5. Traversing the Environment</b>	15
Viewing a Library	15
Viewing an Object in a Library	15
Viewing a Library's Parent	15
Viewing Your Home Library	15
Viewing the Specification of an Environment Package	16
<b>Chapter 6. Using General Editing Operations</b>	17
Selecting an Arbitrary Region of Text	17
Moving Selected Text	17
Copying Selected Text	17
Searching for a String	18
Searching and Replacing a String	18
Searching and Replacing All Occurrences of a String	19
Deleting Text	19
Joining Lines	19
Transposing Text	20
Changing the Case of Text	21

<b>Chapter 7. Writing Text Files</b>	<b>23</b>
Creating a File	23
Viewing a File	23
Editing an Existing File	23
Saving a File	24
Setting Tabs	24
Setting Overwrite Mode On	24
Setting Insert Mode On	25
Setting Wordwrap for Text	25
Changing the Wordwrap Column	25
Turning Wordwrap Off	25
<b>Chapter 8. Writing Ada Programs</b>	<b>27</b>
Creating an Ada Package Specification	27
Creating an Ada Package Body	28
Creating an Ada Subprogram	29
Creating a Subunit	29
Importing Units	29
Adding a Statement, Declaration, or Comment	30
Changing a Statement, Declaration, or Comment	31
Deleting a Statement, Declaration, or Comment	32
Changing the Name or Kind of an Ada Unit	33
Adding a Subprogram to a Package	34
Making a Package Body or Subprogram Body into a Subunit	36
Making a Subunit In-line in the Parent	36
Demoting a Unit and Its Dependents	36
Making a Library Program Executable	36
Executing a Library Program	37
Saving the Changes of Incomplete Units	37
Setting Overwrite Mode On	37
Setting Insert Mode On	37
<b>Chapter 9. Browsing Ada Programs</b>	<b>39</b>
Getting the Definition or Use of an Identifier	39
Viewing the Specification of an Environment Package	39
Viewing a Unit's Specification from Its Body	39
Viewing a Unit's Body from Its Specification	40
Viewing a Unit's Parent	40
Showing the Using Occurrences of a Defined Ada Name	40

<b>Chapter 10. Debugging</b>	41
Starting the Debugger	41
Stopping the Debugger	41
Displaying the Program Being Debugged	41
Displaying the Value of a Program Variable	41
Displaying the Call Stack	42
Displaying Source for a Call Stack Frame	42
Displaying Parameters for a Call Stack Frame	42
Stepping Through the Program	42
Executing the Program	43
Setting Up Exception Handling	43
Setting Breakpoints	43
Showing Breakpoints	43
Removing Breakpoints	44
Modifying a Program Variable	44
Returning to the Point of Program Suspension	44
Displaying the Debugger Window	44
<b>Chapter 11. Managing Libraries</b>	45
Controlling the Library Display	45
Creating Libraries	46
Deleting Objects in a Library	46
Undeleting Objects or Previous Versions in a Library	47
Copying Objects in a Library	47
Moving Objects in a Library	48
Renaming Objects in a Library	48
Printing Objects Contained in a Library	49
<b>Chapter 12. Managing Links</b>	51
Listing Links—Simple Method	51
Adding Links—Simple Method	51
Getting the Pathname for an Environment Package	51
Editing Links for a World	52
Controlling the Link Display	52
Inserting a New Link	52
Deleting a Link	53
Viewing the Source of a Link	53
Exiting from the Link Display	53
Adding a Set of Links	53

Replacing a Link . . . . .	53
<b>Chapter 13. Managing Session Switches . . . . .</b>	<b>55</b>
Editing Session Switches . . . . .	55
Controlling the Session Switch Display . . . . .	55
Modifying Session Switch Values . . . . .	56
Getting Help on Session Switches . . . . .	57
Saving Session Switches . . . . .	57
Exiting from the Session Switch Display . . . . .	57
<b>Chapter 14. Managing Searchlists . . . . .</b>	<b>59</b>
Editing the Searchlist for a Session . . . . .	59
Adding a Component to a Searchlist . . . . .	59
Deleting a Component from a Searchlist . . . . .	59
Replacing One Component with Another . . . . .	60
Viewing the Library Named by a Searchlist Entry . . . . .	60
Exiting from the Searchlist Display . . . . .	60
<b>Chapter 15. Managing Jobs . . . . .</b>	<b>61</b>
Disconnecting from a Job . . . . .	61
Reconnecting to a Job . . . . .	61
Killing the Current Job or the Last Job Created . . . . .	61
Killing Any Job . . . . .	62
<b>Chapter 16. Customizing Your Workspace . . . . .</b>	<b>63</b>
Building Macros . . . . .	63
Defining Your Own Login Procedure . . . . .	64
Rebinding Keys . . . . .	64
<b>Chapter 17. Using CMVC . . . . .</b>	<b>65</b>
Creating a Subsystem . . . . .	65
Adding, Changing, or Deleting Ada Units in a View . . . . .	65
Making Ada Units Controlled . . . . .	65
Making a Subpath . . . . .	66
Checking Out a Unit for Changes . . . . .	66
Checking In a Unit after Changes . . . . .	66
Making a Frozen Release . . . . .	67
Accepting Changes . . . . .	67
Getting Information . . . . .	68

<b>Chapter 18. Networking</b>	69
Logging Into Another System with Telnet	69
Interrupting a Telnet Session	69
Resuming a Telnet Session	70
Terminating a Telnet Session	70
Copying a Single Object or Library onto Another R1000	71
Copying Objects or Libraries from Another R1000	72
Copying Objects onto a Non-R1000 System	73
Copying Objects from a Non-R1000 System	73



## Preface

This *Rational Environment Basic Operations* manual describes, with simple step-by-step procedures, how to perform various common operations in the Rational Environment™ using the Facit Terminal.

Not intended as a self-study guide, this manual assumes some familiarity with the Environment. No conceptual discussions are included. Familiarity typically is acquired through the Rational Environment Training: Fundamentals course or the *Rational Environment User's Guide*.

This manual focuses on fundamental areas of the Environment necessary to begin work on small Ada® programs in single libraries. Some of the areas are: executing commands, managing windows, writing and debugging programs, and editing text files. Areas not included are multilibrary development, sophisticated use Rational Subsystems™, and optional products such as the Rational Design Facility, Rational Mail Utility, host-target development products, and so on.

RATIONAL

## Chapter 1. Logging In and Out

### Logging In

Begin with the terminal turned on.

1. Start the login sequence: `Return`
2. At the Enter user name: prompt, enter your username and press `Return`.
3. At the Enter password: prompt, enter your password (it will not be echoed) and press `Return`.
4. At the Enter session name: prompt, enter a session name and press `Return`, (just press `Return` for the default session named S\_1).

The Environment momentarily displays a message indicating the last time you were logged in, the screen goes blank, and the Environment session appears on the screen. A Login procedure in your home library is executed if it exists and is in the coded state.

### Logging Out

Begin in any window.

1. Create a Command window: `Create Command`
2. Enter quit and press `Promote`

If no uncommitted (unsaved) images exist and if no programs requesting interactive input are running, the command is displayed in reverse video; the screen goes blank and you are logged out.

If any images were left without saving or promoting, or if a program requesting interactive input is running, an error message is displayed in the Message window indicating that images were left with unsaved changes. You can save all changed images (see below) and terminate any such running programs. Otherwise, enter quit (true) and press `Promote`. This logs you off the Environment without saving any uncommitted images.

## Saving Changes

Begin in any window.

### *Saving changes one image at a time*

1. Go to the Window Directory:  -
2. Place the cursor on a line containing an asterisk (\*) in the Mod column.
3. Select the Window Directory entry:  -
4. Save the selected image:

The Mod column is now blank.

Note that running programs requesting input still have a \* in the Mod column. These programs must be terminated by killing their jobs (see "Killing Any Job" in Chapter 15).

5. Continue saving the changes desired by repeating the steps above.

### *Saving changes in all images in a single operation*

1. Go to the Window Directory:  -
2. Place the cursor on the top line of the image:  -
3. Save all changes:

All images that have been changed now have a blank in the Mod column.

Note that running programs requesting input still have a \* in the Mod column. These programs must be terminated by killing their jobs (see "Killing Any Job" in Chapter 15).

## Chapter 2. Getting Help

### Getting Help on Help

To determine the available help for the Environment:

1. Ask for help: `Help on Help`

The Environment displays the available help options in the Help window.

### Getting Help on a Specific Item

*To get help on an Ada item (for example, a command) in an Ada or a Command window*

Begin in the window containing the Ada item.

1. Place the cursor on the item for which you want help.
2. Press `Help`. The Environment creates a Command window and displays the command `What.Does ( "");`
3. Execute the command by pressing `Promote`

If help is available for the command, it is displayed in the Help window.

*To get help on a named topic, command name or name fragment, and so on*

1. Ask for help: `Help`  
The Environment creates a Command window and displays the command `What.Does(Name => "");`
2. At the prompt, enter the topic, command name, or command name fragment for the area of interest and press `Promote`

If more than one command related to that topic exists, all the related commands are listed in the Help window. If you want to see the help for one of these items, place the cursor on the line on which the item is located and press `Object` - `T`. The help for that item is displayed in the Help window.

If only one command about that topic exists, information about that command, including a brief command description and a list of any keys bound to the command, is displayed in the Help window.

If no commands can be found about that topic, a message appears indicating that no help is available for that topic.

### Getting Help on Keys

To determine what commands are bound to a key or key combination:

1. Ask for help on a key: `Help on Key`

The Environment displays the following prompt in the Message window:

Press key to be described:

2. Press the key or key combination of interest.

The command name bound to the key or key combination is displayed in the Message window. Additional help about the command, if any exists, is also displayed in the Help window.

### Displaying Ada Specifications

To go to the Ada specification for an item described in the Help window:

Begin in the Help window in the entry for the message of interest.

1. Place the cursor on the line in the Help window containing the text for the Ada code for the item.
2. Ask for the definition of the designated item: `Definition`

If there is an Ada spec for the item, it is displayed and highlighted in an Ada window.

### Displaying the Help Window

Begin in any window.

1. Ask to go to the Help window: `Help Window`

The Help window is brought onto the screen and the cursor is placed in it. You can now scroll through the contents of the window to view the help messages that have been requested since you logged in.

## Getting Help on Errors

To get additional information about an error in your program or command:

1. Move the cursor onto the underlined error.
2. Ask for help on the error: `Object - ?`

Additional messages about the error appear in the **Message** window if the **Environment** has any more information to give you.

RATIONAL



## Chapter 3. Executing Commands

### Creating and Executing a Command Window Program

A Command window program can contain any arbitrarily sized Ada code—for example, one-line Environment commands, multiple-line test programs, or Ada main programs.

Begin in any window.

1. Create a Command window: `Create Command`
2. Enter the program, formatting frequently for multiple-line programs: `Format`
3. Semanticize for multiple-line programs: `Semanticize`  
The Environment marks the errors that exist. Press `Object` - `?` for further information about any errors.
4. Correct any errors and semanticize again.
5. Execute the command program: `Promote`

### Expanding a Command Window

Begin in the Command window you want to expand.

1. Enlarge the window: `Window` - `!`

The window expands by four lines.

### Shrinking a Command Window

Begin in the Command window you want to shrink.

1. Shrink the window: `Window` - `.`

The window shrinks by four lines.

## Getting Command Completion

Begin in a Command window.

1. Enter some fragment of the command.
  - You may supply only a command name or name fragment. Completion will fail if you enter any part of the argument list, including the parenthesis that begins the list.
  - Completion ignores final semicolons if any exist (for example, if you have pressed the `Format` key and it has added a semicolon after the name or name fragment).
2. Complete the command and provide prompting for any parameters: `Complete`  
If the command fragment is ambiguous, the complete operation fails and the Environment displays the possibilities in another window. Enter the necessary characters to make the command unique and press `Complete` again.

## Moving to the Next Prompt or Underline

Begin in the Command window.

1. Move to the next item (highlighted or underlined): `Esc` - `N`

The cursor is now placed at the next item (to the right or below).

## Moving to the Previous Prompt or Underline

Begin in the Command window.

1. Move to the previous item (highlighted or underlined): `Esc` - `U`

The cursor is now placed at the next item (to the left or above).

## Turning Off a Prompt

Begin with the cursor on the prompt that is to be turned into text.

1. Turn off the prompt: `Control` `X`

## Reexecuting the Same Command

Begin in the Command window containing the command to be reexecuted.

1. Execute the command: `Promote`

## Changing and Reexecuting a Command

Begin with the cursor on the command to be changed.

1. Turn the command from a prompt into text: `Control X`  
The command text can now be edited.
2. Execute the changed command: `Promote`

## Entering a New Command in the Same Command Window

Begin with the cursor on the old command prompt.

1. Type the new command over the old command.

The old command prompt disappears.

## Clearing a Command Window of Unneeded Text

Begin in the Command window to be cleared.

1. Clear the Command window: `Edit`

Note that the unneeded text in the Command window has been replaced with a statement prompt allowing entry of new commands.

## Going Back to Previous Commands

A history of commands and Ada programs entered into a Command window is maintained. You can access and execute any of the commands in this sequential history.

Begin in the Command window.

*Redisplaying a previous command in the historical sequence (undoing)*

1. Redisplay the previous command: `Object - U`

*Redisplaying a later command in the historical sequence (redoing)*

1. Redisplay the next command: `Object - R`

## Getting the Parameters of a Command Bound to a Key

Begin in any window.

1. Create a Command window with the parameters for a command bound to a key: `Esc - Q - command key`



## Chapter 4. Managing Windows

### Finding a Window Using the Window Directory

Begin in any window.

1. Display the Window Directory: **Window** - **Definition**  
The Window Directory is displayed in a new window.
2. Place the cursor on the line of the Window Directory entry that names the window at which you want to look.
3. Ask to view the object: **Definition**

The indicated object appears in the same frame as the Window Directory window (or in an empty frame if one exists).

### Deleting Windows from the Window Directory

Begin in the Window Directory window.

1. Place the cursor on the line of the window to be deleted.
2. Select the line: **Object** - **—**
3. Delete the window: **Object** - **D**

The window is removed from the Window Directory. This releases the image.

### Moving between Windows

*Moving to the window above (with vertical wraparound)*

1. Move to the window above: **Window** - **↑**

*Moving to the window below (with vertical wraparound)*

1. Move to the window below: **Window** - **↓**

### Expanding a Window

Begin in the window you want to expand.

1. Enlarge the window: `Window` - `I`

The window expands by four lines.

### Shrinking a Window

Begin in the window you want to shrink.

1. Shrink the window: `Window` - `O`

The window shrinks by four lines.

### Expanding Current Window to Include Next Frame

Begin in the window you want to expand.

1. Join the windows: `Window` - `J`

The current window expands to the size of the current window plus the window below, replacing any window that might have been on the screen. The window returns to its normal size automatically when the next object is viewed.

### Expanding Current Window to Include Previous Frame

Begin in the window you want to expand.

1. Join the windows: `Window` - `Delete`

The current window expands to the size of the current window plus the window above, replacing any window that might have been on the screen. The window returns to its normal size automatically when the next object is viewed.

### Transposing Windows

You can switch the location of a window with that of the window above it (with vertical wraparound).

Begin in the lower window.

1. Transpose the windows: `Window` - `T`

The cursor appears in the new lower window. It is in the same position that it was in when that window was last viewed.

## Realigning the Windows on the Screen

Begin in any window.

1. Return windows to their default configuration: **Window** - **Format**

## Removing a Window

You can remove a window from your screen in one of three ways.

### *Removing a window temporarily*

This command removes the window from the screen and leaves it available in the Window Directory.

1. Place the cursor in the window you want to remove.
2. Delete the window: **Window** - **D**

### *Releasing an image permanently and saving the changes*

This command releases the image and removes the window after saving the image. The window is no longer available in the Window Directory.

1. Place the cursor in the window you want to release.
2. Release the image: **Object** - **X**

### *Releasing an image permanently without saving the changes*

This command abandons the image and removes the window. The window is no longer available in the Window Directory. Unsaved changes are discarded.

1. Place the cursor in the window you want to release.
2. Abandon the image: **Object** - **G**

## Locking a Window on the Screen

Begin in the window you want to lock.

1. Lock the window: **Window** - **Promote**

An *at* sign (@) appears in the window banner. The window is not removed unless you explicitly remove it or unlock it.

## Unlocking a Window on the Screen

Begin in the window you want to unlock.

1. Unlock the window: `Window` - `Demote`

The *at* sign (@) disappears from the window banner.

## Scrolling the Image

Begin in the window containing the image to be scrolled.

### *Scrolling the image up*

1. Scroll the image up: `Image` - `↑`

### *Scrolling the image down*

1. Scroll the image down: `Image` - `↓`

### *Scrolling to the beginning of the image*

1. Scroll to the beginning of the image: `Image` - `Begin Of`

### *Scrolling to the end of the image*

1. Scroll to the end of the image: `Image` - `End Of`

### *Scrolling the current line to the top*

1. Scroll the current line to the top: `Window` - `Begin Of`

### *Scrolling the current line to the bottom*

1. Scroll the current line to the bottom: `Window` - `End Of`



## Chapter 5. Traversing the Environment

### Viewing a Library

Begin in the world or directory that contains the library.

1. Place the cursor on the line containing the library.
2. View the library: `Definition`

A window appears, displaying the full pathname of the library underlined and listing additional library objects, such as Ada units or files, if they exist.

### Viewing an Object in a Library

Begin in the library containing the object.

1. Place the cursor on the line of the library object you want to view.
2. View the object: `Definition`

A window displaying the object appears.

### Viewing a Library's Parent

Begin in the library.

1. View the parent: `Enclosing`

A window containing the parent library appears.

### Viewing Your Home Library

Begin in any library.

1. View your home library: `Esc` - `1`

A window containing your home library appears.

## Viewing the Specification of an Environment Package

Here is a convenient shortcut for displaying the specifications for Ada units provided as part of the Environment (for example, for viewing the specification for package Compilation, which contains the compilation commands).

Begin in any window.

1. Get a prompt for the Definition command: `Esc - Q - Definition`
2. Enter the simple name of the Ada unit at the prompt for the Name parameter preceded by the \ character (for example, "\Compilation").
3. Execute the command: `Promote`

Note that this shortcut for viewing Environment package specifications works for most Environment packages. If the shortcut fails, an error message appears, and you will have to traverse to the specification instead.

## Chapter 6. Using General Editing Operations

### Selecting an Arbitrary Region of Text

Begin in the window containing the text to be selected.

1. Move the cursor to the start of the region of text to be selected.
2. Define the start of the region: **Region** - **[**
3. Move the cursor to the end of the region of text.
4. Define the end of the region: **Region** - **]**

The selected region is highlighted.

### Moving Selected Text

Begin in the window containing the text to be moved.

1. Select the region of text.
2. Move the cursor to the location in which the text will be moved. You can move text within the same image or to some other image.
3. Move the region of text: **Region** - **M**

The highlighted region of text is deleted from its original location and appears in the new location.

### Copying Selected Text

Begin in the window containing the text to be copied.

1. Select the region of text.
2. Move the cursor to the location in which the text will be copied. You can copy text within the same image or into some other image.
3. Copy the region of text: **Region** - **C**

The region of text appears in its original location and in the new location.

## Searching for a String

Begin in the text in which you want to search for the string.

1. Move to the beginning of the image: **Image** - **Begin Of**
2. Start the search command (enter composing mode): **Control****F**
3. Enter the target string, without quotes. Note that the characters you type in composing mode appear at the SEARCH prompt in the Message window.
4. Start the actual search (enter search mode): **Control****F**  
If the target string is found, the cursor is positioned one character after the target string.
5. To get to each additional occurrence of the string: **Control****F**
6. To return to a previous occurrence of the string: **Control****R**
7. To cancel the search, press any key—for example, **I**.  
The SEARCH prompt is removed from the Message window.

## Searching and Replacing a String

Begin in the text with the string to be changed.

1. Move to the beginning of the image: **Image** - **Begin Of**
2. Start the search/replace command: **Esc** - **F**
3. At the SEARCH prompt in the Message window, enter the target string, without quotes.
4. Press **Next Item** to move to the REPLACE prompt.
5. At the REPLACE prompt in the Message window, enter the replacement string, without quotes.
6. Start the actual search/replace: **Esc** - **F**  
The Environment places the cursor one character after the target string.
7. To replace the target string: **Esc** - **F**  
The Environment replaces the string and places the cursor one character after the next occurrence of the target string.
8. To get to each additional occurrence of the string without changing the string: **Control****F**
9. To replace a previous occurrence of the string: **Esc** - **R**
10. To abort searching and replacing, press any key—for example, **I**.  
The SEARCH and REPLACE prompts are removed from the Message window.

## Searching and Replacing All Occurrences of a String

Begin in the text with the string to be changed.

1. Move to the beginning of the image: **Image** - **Begin Of**
2. Start the search/replace command: **Esc** - **F**
3. At the SEARCH prompt in the Message window, enter the existing string, without quotes.
4. At the REPLACE prompt in the Message window, enter the new string, without quotes.
5. Start the actual search and global replace: **numeric -** - **numeric 1** - **Esc** - **F**  
(Use the numeric keypad to enter the -1.)

The Environment replaces all occurrences of the target string and displays the number of occurrences in the Message window.

## Deleting Text

Text such as characters, words, lines, and regions can be deleted. Text can be deleted from varying cursor positions.

- Delete the character at the cursor: **Control** - **D**
- Delete the character before the cursor position (backspacing): **Delete**
- Delete the entire word: **Word** - **D**
- Delete from the cursor to the end of the word: **Word** - **K**
- Delete from the cursor to the beginning of the word: **Word** - **Delete**
- Delete the entire line: **Line** - **D**
- Delete from the cursor to the end of the line: **Line** - **K**
- Delete from the cursor to the beginning of the line: **Line** - **Delete**
- Delete the selected text: **Region** - **D**

## Joining Lines

This command joins the line on which the cursor is located with the following line.

1. Move the cursor to any position on the first line of the two lines to be joined.
2. Join the second line to the end of the first line: **Line** - **J**

## Transposing Text

### *Transposing characters*

This command switches the character that the cursor is on with the previous character. Assume, for example, that character 2 follows character 1, and you want character 1 to follow character 2.

1. Move the cursor to character 2.
2. Transpose the character that the cursor is on and the previous character: `Control-T`

### *Transposing words*

This command switches the word that the cursor is on with the previous word. Assume, for example, that word 2 follows word 1, and you want word 1 to follow word 2. Word terminators are blanks, underscores, semicolons, or periods.

1. Move the cursor to any place on word 2.
2. Transpose the word that the cursor is on and the previous word: `Word - T`

### *Transposing lines*

This command switches the line that the cursor is on with the previous line. Assume, for example, that line 2 follows line 1, and you want line 1 to follow line 2.

1. Move the cursor to any place on line 2.
2. Transpose the line that the cursor is on and the previous line: `Line - T`

## Changing the Case of Text

The case of text such as characters, words, lines, and regions can be changed to lowercase, uppercase, or initial capitals. Begin with the cursor anywhere in the text to be changed.

- Capitalize a character: **Control** - **>**
- Lowercase a character: **Control** - **<**
  
- Uppercase a word: **Word** - **>**
- Lowercase a word: **Word** - **<**
- Capitalize a word: **Word** - **^**
  
- Uppercase a line: **Line** - **>**
- Lowercase a line: **Line** - **<**
- Capitalize a line: **Line** - **^**
  
- Uppercase a selected region: **Region** - **>**
- Lowercase a selected region: **Region** - **<**
- Capitalize a selected region: **Region** - **^**

RATIONAL



## Chapter 7. Writing Text Files

### Creating a File

Begin in the library in which you want the file.

1. Create a file: `Create Text`

A Command window with the Text.Create command and its parameter is created.

2. At the Image\_Name prompt, enter the name of the file to be created and press `Promote`

A new window is created for the image of your file, and an entry for the file appears in the library.

### Viewing a File

Begin in the library containing the file.

1. Move the cursor to the line containing the file declaration.
2. Go to the definition: `Definition`

A window with a read-only image of the file appears.

### Editing an Existing File

Begin in the library containing the file.

1. Move the cursor to the line containing the file declaration.
2. Select the file to be edited: `Object` - `[-]`
3. Edit the selected file: `Edit`

The Environment displays the image of the object in a window. You are now ready to edit the file.

4. Save the image periodically by pressing `Enter`
5. When you have finished editing, promote the file to a read-only image by pressing `Promote`

### Saving a File

A file can be saved in one of two ways.

#### *Saving a file (close for editing)*

When you have made some changes and you want to save them and terminate editing:

1. Place the cursor in the window that has the image of the file.
2. Promote the image to a read-only image: `Promote`

This command saves the image of the file and allows others to access it.

#### *Saving a file (leave open for editing)*

When you have made some changes and you want to save them but continue editing:

1. Place the cursor in the window that has the image of the file.
2. Commit the image: `Enter`

This command saves the image of the file, and you retain update access.

### Setting Tabs

Begin in the text.

1. Create a Command window.
2. To set tab stops at every *n*th column, enter `set.tab_width(n)` and press `Promote`

As you edit the text file, pressing `Control,I` indents *n* spaces.

### Setting Overwrite Mode On

Begin in the text.

1. Set overwrite mode on: `Image - O`

The banner is updated to indicate that overwrite mode is in effect in this window.

## Setting Insert Mode On

Begin in the text.

1. Set insert mode on: `Image` - `I`

## Setting Wordwrap for Text

Begin in the text.

1. Turn fill mode on: `Image` - `F`

The banner shows that fill mode is in effect and indicates the column number. The column number default is 72.

## Changing the Wordwrap Column

Begin in the text.

1. Create a Command window.
2. To set a different wordwrap column, enter `set.fill_column` and press `Complete`.
3. At the prompt, enter `n`, where `n` is the desired column number, and press `Promote`.

## Turning Wordwrap Off

Begin in the text.

1. Turn fill mode off: `Image` - `X`

The banner is updated to remove the fill mode indicator and fill column number.

RATIONAL

## Chapter 8. Writing Ada Programs

Libraries are of two kinds: directories and worlds. Programs can be written in either kind of library.

### Creating an Ada Package Specification

Begin in the library that will contain the Ada unit.

1. Create a workspace: **Object** - **I**  
A new window is created with a `comp_unit` prompt for you to begin editing.
2. Enter the contents of the specification in the new window at the `comp_unit` prompt.  
Use **Create Private** for building the private part of the specification, if appropriate.
3. Format frequently by pressing **Format**  
The Environment marks any errors that exist. Use **Object** - **?** for information about any errors.
4. Semanticize frequently by pressing **Semanticize**  
The Environment marks any errors that exist. Use **Object** - **?** for information about any errors.  
The first time you semanticize, a temporary name appears in the banner of the Ada unit you are editing and in the library that contains the Ada unit. A temporary name is of the form `_Ada_#_`, where # is some number.
5. Correct any errors.
6. Promote the specification to the installed state: **Promote**

The Environment replaces the temporary name in the library with the Ada name for the unit specification.

## Creating an Ada Package Body

Begin in the package specification.

1. Use `Create Body` to build the skeletal package body.

A new window appears with the skeletal package body for you to edit.

2. Enter the contents of the body.
3. Format and semanticize frequently.

The Environment marks any errors that exist. Use `Object` - `?` for information about any errors.

The first time you semanticize, a temporary name appears in the banner of the Ada unit you are editing and in the library that contains the Ada unit. A temporary name is of the form `_Ada_#_`, where # is some number.

4. Correct any errors.
5. Promote the body to the installed state: `Promote`

The Environment replaces the temporary name in the library with the Ada name for the unit specification.

## Creating an Ada Subprogram

Begin in the library that is to contain the Ada unit.

1. Create a workspace: **Object** - **1**

The Environment creates a new window with a `comp_unit` prompt.

2. Enter the body of the subprogram.
3. Format and semanticize the unit.

The Environment marks any errors that exist. Use **Object** - **7** for information about any errors.

The first time you semanticize, a temporary name appears in the banner of the Ada unit you are editing and in the library that contains the Ada unit. A temporary name is of the form `_Ada_#_`, where # is some number.

4. Correct any errors.
5. Promote the subprogram to the installed state: **Promote**

The Environment replaces the temporary name in the library with the Ada name for the unit. It also creates a separate specification for the unit in the library.

## Creating a Subunit

Begin in the Ada unit that will contain the subunit.

1. Enter the Ada subunit stub notation. You might enter, for example, `procedure foo is separate;`
2. Format.
3. Place the cursor on the stub.
4. Select the stub: **Object** - **--**
5. Edit the selected stub: **Edit**

A new window containing the skeletal subunit appears. The name of the subunit appears in the library under the parent unit.

## Importing Units

To import units, see “Adding Links—Simple Method” in Chapter 12.

## Adding a Statement, Declaration, or Comment

### *Adding to an Ada unit in the source state*

Begin in the Ada unit in which you want to make the addition.

1. Edit the Ada unit, if it is still in read-only mode: **Edit**
2. Go the position where the new statement, declaration, or comment is to be added.
3. Enter the changes.
4. Format and semanticize.
5. Correct any errors.

### *Adding to an Ada unit in the installed or coded state*

Begin in the Ada unit in which you want to make the addition.

1. If the Ada unit is a package specification or if the addition you want to make contains only Ada comments, skip to the next step.

If it is already coded, demote the Ada unit to the installed state: **Install Unit**

2. Go to the position where the new statement, declaration, or comment is to be added.
3. Open an insertion point: **Object** - **I**

A new window appears with the banner labeled either statement or declaration, depending on the location of the insertion point.

The library now contains a temporary name of the form `_Ada_#_`, where # is some number, under the library unit you are editing.

4. Enter the new statement, declaration, or comment.  
Note that multiple statements, declarations, or comments can be entered per insertion point.
5. Format and semanticize.
6. Correct any errors.
7. Promote the statement, declaration, or comment: **Promote**

The new window disappears, and the prompt in the unit is replaced by the actual statement, declaration, or comment. The temporary name in the library is removed.



## Changing a Statement, Declaration, or Comment

### *Making changes in an Ada unit in the source state*

Begin in the Ada unit in which you want to make the change.

1. Edit the Ada unit, if it is still in read-only mode:
2. Go to the position where the statement, declaration, or comment is to be changed.
3. Enter the changes.
4. Format and semanticize.
5. Correct any errors.

### *Making changes in an Ada unit in the installed or coded state*

Begin in the Ada unit in which you want to make the change.

1. If the Ada unit is a package specification or if the change you want to make consists only of Ada comments, skip to the next step.

If it is already coded, demote the unit to the installed state:

2. Go to the end of the statement, declaration, or comment to be changed.
3. Select the entire statement, declaration, or comment:  -
4. Edit the selected statement, declaration, or comment:

The selected statement, declaration, or comment becomes a prompt, and a window with the statement, declaration, or comment appears on the screen.

The library now contains a temporary name of the form `_Ada_#_`, where `#` is some number, under the library unit you are editing.

Note that if the selected declaration has dependents, the edit operation will not succeed until all dependents are demoted to source.

5. Enter the changes.

Note that multiple declarations, statements, or comments can be entered.

6. Format and semanticize.
7. Correct any errors.
8. Promote the statement, declaration, or comment:

The new window disappears, and the prompt in the unit is replaced by the actual statement, declaration, or comment. The temporary name in the library is removed.

## Deleting a Statement, Declaration, or Comment

### *Deleting in an Ada unit in the source state*

Begin in the Ada unit in which you want to make the change.

1. Edit the Ada unit, if it is still in read-only mode:
2. Go the position where the statement, declaration, or comment is to be deleted.
3. Use line delete or region delete to remove the statement, declaration, comment.

The unit remains in the source state for further editing.

### *Deleting in an Ada unit in the installed or coded state*

Begin in the Ada unit in which you want to make the change.

1. If the Ada unit is a package specification or if the deletion you want to make contains only Ada comments, skip to the next step.

If it is already coded, demote the unit to the installed state:

2. Go to the end of the statement, declaration, or comment to be deleted.
3. Select the entire statement, declaration, or comment:  -
4. Delete the selected statement, declaration, or comment:  -

The selected statement, declaration, or comment is removed.

Note that if the selected declaration has dependents, the delete operation will not succeed until all dependents are demoted to source.

## Changing the Name or Kind of an Ada Unit

### *Changing the name or kind of an Ada unit in the source state*

Begin in the library containing the Ada unit to be changed.

1. Move the cursor to the line containing the Ada unit.
2. Select the Ada unit:  -
3. Edit and withdraw the selection: 

The selected Ada unit is replaced by a temporary name, and a window with the Ada unit appears on the screen. The unit can be edited.
4. Change the unit name, parameter profile, or unit kind.

The temporary name in the library is replaced by the new actual name for the Ada unit when you promote the unit. The unit is still in the source state to allow continued editing.

### *Changing the name or kind of an Ada unit in the installed or coded state*

Begin in the library containing the Ada unit to be changed.

1. Move the cursor to the line containing the Ada unit.
2. Select the Ada unit:  -
3. Edit and withdraw the selection: 

The selected Ada unit is replaced by a temporary name, and a window with the Ada unit appears on the screen. The unit is in the source state.

Note that if the selected unit has dependents, the withdraw operation will not succeed until all dependents are demoted to source.
4. Enter the changes.
5. Format and semanticize.
6. Correct any errors.
7. Promote the unit:

The temporary name in the library is replaced by the new actual name for the Ada unit.

## Adding a Subprogram to a Package

These steps assume that the subprogram is to be added to both the specification and the body of the package.

### *Adding to an Ada unit in the source state*

Begin in the package specification in which you want to add the subprogram specification.

1. Edit the Ada unit, if it is still in read-only mode:
2. Go to the position in the package where the new subprogram specification is to be added.
3. Enter the new subprogram specification.
4. Format and semanticize.
5. Correct any errors.
6. Select the subprogram specification:  - - 7. Create the body:   
The skeletal subprogram body is placed at the end of the existing package body.
- 8. Enter the subprogram body.
- 9. Format and semanticize frequently.
- 10. Correct any errors.

*Adding to an Ada unit in the installed or coded state*

Begin in the package specification in which you want to add the subprogram specification.

1. Go to the position in the package where the new subprogram specification is to be added.
2. Open an insertion point:  -   
 A new window with a declaration prompt is created for editing. A temporary name appears in the library under the package specification to which you are adding the subprogram.
3. Enter the new subprogram specification at the prompt.  
 Note that multiple subprogram specifications can be entered per insertion point.
4. Format and semanticize.
5. Correct any errors.
6. Promote the declaration:   
 The new window disappears and the prompt in the package specification is replaced with the added subprogram specification. The temporary name in the library disappears.
7. Select the subprogram specification:  -
8. Create the body:   
 A new window appears on the screen with the skeletal subprogram body.
9. Enter the subprogram.
10. Format and semanticize frequently.
11. Promote the subprogram body:

The window is replaced by a window displaying the existing package body with the new subprogram installed.

## Making a Package Body or Subprogram Body into a Subunit

Begin in the parent unit containing the declaration stub in either the source or the installed state.

1. Select the unit that you want to make into a subunit:  -
2. Create a Command window.
3. Enter `make_separate` and press

A new window with the subunit appears and the parent unit has an appropriate subunit stub. Note that the subunit is now in the source state.

## Making a Subunit In-line in the Parent

Begin in the parent Ada unit in either the source or the installed state.

1. Select the subunit stub.
2. Create a Command window.
3. Enter `make_inline` and press

The subunit stub is replaced by the actual subunit code. Note that the in-line unit is in the same state as the parent.

## Demoting a Unit and Its Dependents

Begin in the library that contains the program unit.

1. Place the cursor on the line containing the program unit to be demoted.
2. Select the unit to be demoted:  -
3. Demote the program unit:

The progress of the command is displayed in the Environment I/O window. The unit, plus any units that depend on it, is demoted to source.

## Making a Library Program Executable


Begin in the library that contains the program.

1. Make the program executable:

All units in the library are promoted to the coded state. The progress of the command is displayed in the Environment I/O window.

## Executing a Library Program


Begin in the library containing the program.

1. Create a Command window.
2. Enter the Ada name for the program.
3. Execute the program: 

The Environment then executes the program just as it executes any Environment command.

## Saving the Changes of Incomplete Units



Begin in the Ada unit that is incomplete—that is, the unit still may have errors or you want to do further development on the unit before promoting it.

1. Save the image: 

A message appears in the Message window indicating that the unit has been saved (*committed*). The banner of the Ada unit now has a blank in the first character position.

## Setting Overwrite Mode On



Begin in the Ada unit you are editing.

1. Set overwrite mode on:  - 

The banner is updated to indicate that overwrite mode is in effect in this window. Overwrite mode is set on a window-by-window basis.

## Setting Insert Mode On

Insert mode is the default. Begin in the window that is currently in overwrite mode.

1. Set insert mode on:  - 

The banner is updated to remove the overwrite mode indicator.

RATIONAL



## Chapter 9. Browsing Ada Programs

### Getting the Definition or Use of an Identifier

Begin with the cursor on the identifier.

1. Select the identifier: `Object` - `—`
2. Go to the definition: `Definition`

A window containing the definition of the declaration appears.

### Viewing the Specification of an Environment Package

Here is a convenient shortcut for displaying the specifications for Ada units provided as part of the Environment (for example, for viewing the specification for package Compilation, which contains the compilation commands).

Begin in any window.

1. Get a prompt for the Definition command: `Esc` - `Q` - `Definition`
2. Enter the simple name of the Ada unit at the prompt for the Name parameter preceded by the `\` character (for example, `"\Compilation"`).
3. Execute the command: `Promote`

Note that this shortcut for viewing Environment package specifications works for most Environment packages. If the shortcut fails, an error message appears, and you have to traverse to the specification instead.

### Viewing a Unit's Specification from Its Body

Begin in the body.

1. Go to the specification: `Other Part`

A window containing the specification appears.

## Viewing a Unit's Body from Its Specification

Begin in the specification.

1. Go to the body: **Other Part**

A window containing the body appears.

## Viewing a Unit's Parent

Begin in the unit.

1. Go to the parent: **Enclosing**

A window containing the parent object appears.

## Showing the Using Occurrences of a Defined Ada Name

Begin in the window containing the Ada name of interest.

1. Place the cursor on an occurrence of the Ada name.
2. Select the Ada name of interest: **Object** - **—**
3. View the using occurrences: **Show Usage**
4. The using occurrences of the Ada name within the current unit are underlined. Use **Esc** - **N** or **Esc** - **U** to step through.

For using occurrences of the Ada name in other units, a window containing the names of these units appears.

5. Place the cursor on a unit.
6. Select the unit: **Object** - **—**
7. View the unit with the using occurrence: **Definition**

A window appears displaying the selected unit with all occurrences of the Ada name of interest underlined.

8. Use **Esc** - **N** or **Esc** - **U** to step through.

## Chapter 10. Debugging

### Starting the Debugger

Begin in the Command window containing the name of the program to be debugged.

1. Invoke the Debugger: `Esc` - `Promote`

The Debugger window appears, and a debugging session begins.

Program execution does not begin until further debugging commands are entered.

### Stopping the Debugger

A debugging session is terminated automatically when you begin to debug a new job or when you log off.

### Displaying the Program Being Debugged

A window automatically displays a section of the program around the point where execution was suspended. The statement or declaration to be executed next is highlighted (*selected*).

### Displaying the Value of a Program Variable

Begin in any window.

1. Place the cursor on an occurrence of the program variable.
2. Select the program variable: `Object` - `~`
3. Display the value: `Put`

The value of the variable is displayed in the Debugger window.

## Displaying the Call Stack

Begin in any window.

1. Display the stack:

The call stack is displayed in the Debugger window with the most current call on the top of the stack (it is frame number one: “\_1”).

## Displaying Source for a Call Stack Frame

Begin in the Debugger window.

1. Display the stack:
2. Place the cursor on the frame you want to display.
3. Select the frame:  -
4. Display the source for the frame:

The Ada unit corresponding to the frame is displayed with the program counter location (either current or saved) highlighted.

## Displaying Parameters for a Call Stack Frame

Begin in the Debugger window.

1. Display the stack:
2. Place the cursor on the frame for which you want to display the parameters.
3. Select the frame:  -
4. Display the parameters for the frame:

## Stepping Through the Program

You can step in one of two ways. Note that in either case you can step multiple times with a single command by pressing a numeric prefix key () before you press the key to step.

Begin in any window.

### *Stepping by every statement*

1. Press

### *Stepping by statements without stopping in called subprograms*

1. Press

## Executing the Program

Begin in any window.

1. Execute the program: **Execute**

The program runs to completion or until an exception or breakpoint is encountered.

## Setting Up Exception Handling

The Debugger stops when any exception is encountered, unless that exception has been propagated.

Begin in an Ada window containing the unit that declares the exception or a unit that handles the exception.

### *Propagating a particular exception*

1. Place the cursor on an occurrence of the exception name.
2. Select the exception: **Object** - **—**
3. Press **Propagate**

### *Catching a previously propagated exception*

1. Place the cursor on an occurrence of the exception name.
2. Select the exception: **Object** - **—**
3. Press **Catch**

## Setting Breakpoints

Begin in the window displaying the Ada unit in which you want to set a breakpoint.

1. Place the cursor on the statement or declaration in the Ada unit.
2. Select the entire statement or declaration by pressing **Object** - **—** repeatedly.
3. Set the breakpoint: **Break**

A breakpoint number is assigned. This breakpoint is in effect until the Debugger session terminates or until it is explicitly deactivated.

## Showing Breakpoints

Begin in any window.

1. Show breakpoints: **Show Breaks**

The display shows all active and inactive breakpoints.

## Removing Breakpoints

Begin in any window.

### *Removing all breakpoints*

1. Remove all breakpoints:

### *Removing a specific breakpoint*

1. Prompt for the remove command:  -  -
2. At the Breakpoint prompt, enter the number of the breakpoint you want deactivated and press

## Modifying a Program Variable

Begin in the window displaying the program variable.

1. Place the cursor on an occurrence of the program variable you want to change.
2. Select the program variable:
3. Prompt for the modify command:
4. At the New\_Value prompt, enter the desired new variable value (in double quotes) and press

## Returning to the Point of Program Suspension

Begin in any window.

1. Go to the program suspension point:

A window containing the definition of the program being debugged appears with the statement or declaration to be executed next highlighted.

## Displaying the Debugger Window

Begin in any window.

1. Create a Command window:
2. Enter `debug.current_debugger` and press

The Debugger window appears on the screen and the cursor is in it.

## Chapter 11. Managing Libraries

### Controlling the Library Display

Begin with the cursor in the library.

#### *Toggling information on library objects*

1. Move to the beginning of the library: `Image` - `Begin Of`
2. Change the display: `Object` - `T`

Repeating this command toggles the library display so that you view one of the following: only the names of the library objects; the name and the type of library objects; and the name, type, Ada unit state plus update information.

#### *Showing more detail on the objects in the library*

1. Show more detail: `Object`- `!`

This causes deleted units, versions, and so on to be added to the library display.

This step can be repeated if necessary until the desired detail level is reached.

#### *Showing less detail on the objects in the library*

1. Show less detail: `Object`- `.`

This causes deleted units, versions, and so on to be removed from the library display.

This step can be repeated if necessary until the desired detail level is reached.

## Creating Libraries

### *Creating a directory*

Begin in the directory or world that is to contain the new directory.

1. Create the directory:

The Environment creates a Command window containing the Library.Create-Directory command and prompts for its parameters.

2. At the Name prompt, enter the name for the new directory and press

The Environment creates a directory. In the containing library, you see the new directory name inserted in alphabetical order.

### *Creating a world*

Begin in the directory or world that is to contain the new world.

1. Create the world:

The Environment creates a Command window containing the Library.Create-World command and prompts for its parameters.

2. At the Name prompt, enter the name for the new world and press

The Environment creates a world. In the containing library, you see the new world name inserted in alphabetical order.

By default, this world has links to commonly used Ada and Environment packages such as Text\_Io, Calendar, and String-Tools. These links are from the model world !Model.R1000.

## Deleting Objects in a Library

### *Deleting a library*

Begin in the library containing the library to be deleted.

1. Place the cursor on the line containing the library to be deleted.
2. Select the library to be deleted:  -
3. Create a Command window:
4. Enter `compilation.delete` and press

The I/O window displays the progress and results of the Delete command. When the command is complete, the library to be deleted disappears from the library.

### *Deleting an Ada unit or file*

Begin in the library containing the object to be deleted.



1. Place the cursor on the line containing the object to be deleted.
2. Select the object: `Object` - `--`
3. Delete the object: `Object` - `D`

If an Ada unit has no dependents, the declaration is removed from the library.

## Undeleting Objects or Previous Versions in a Library

Begin in the library containing the deleted object or version.

1. Expand detail in the library (if necessary) so you can see the object or version to be undeleted: `Object` - `|`

Repeat as necessary until you can see the deleted object or version you want to undelete. A deleted object is enclosed in braces `{ }` to indicate that it is deleted. A previous version has its name prefixed with a minus `(-)`, indicating that it is not the default version.

2. Select the object or version to undelete: `Object` - `--`
3. Undelete it: `Object` - `U`

The object or version is now undeleted and is displayed without the braces around it or without the minus in front of it.

## Copying Objects in a Library

### *Copying into a different library*

Begin in the library containing the object (library, Ada unit, file) to be copied.

1. Place the cursor on the object to be copied.
2. Select the object to be copied: `Object` - `--`
3. Place the cursor in the new library to which the existing object is to be copied.
4. Copy the selected object: `Object` - `C`

A Command window appears with the `Library.Copy` command and prompts for its parameters. The parameter names are supplied automatically by the Environment.

5. Press `Promote`

The progress of the command is displayed in the Environment I/O window.

### *Copying into the same library*

Begin in the library containing the object (library, Ada unit, file) to be copied.

1. Select the object to be copied: **Object** - **—**
2. Copy the selected object: **Object** - **C**  
A Command window appears with the Library.Copy command and prompts for its parameters.
3. At the To prompt, enter the name of the object into which you want to copy.
4. Press **Promote**

The progress of the command is displayed in the Environment I/O window.

### Moving Objects in a Library

#### *Moving to a different library*

Begin in the library containing the object (library, Ada unit, file) to be moved.

1. Place the cursor on the object to be moved.
2. Select the object to be moved: **Object** - **—**
3. Place the cursor in the new library to which the existing object is to be moved.
4. Move the selected object: **Object** - **M**

A Command window appears with the Library.Move command and prompts for its parameters. The parameter names are supplied automatically by the Environment.

5. Press **Promote**

The progress of the command is displayed in the Environment I/O window.

#### *Moving to the same library*

This is equivalent to renaming a library object. See “Renaming Objects in a Library,” below.

### Renaming Objects in a Library

Begin in the library structure containing the object (library, Ada unit, file) to be renamed.

1. Select the object to be renamed: **Object** - **—**
2. Create a Command window: **Create Command**
3. Enter library.rename and press **Complete**
4. At the To prompt, enter the new name and press **Promote**

The progress of the command is displayed in the Environment I/O window. Ada units are demoted to source.

## Printing Objects Contained in a Library

### *Printing a file or an Ada unit*

Begin in the library containing the object to be printed.

1. Move the cursor to the line containing the object to be printed.
2. Select the object: `Object` - `-`
3. Print the object: `Print`

The progress and status are displayed in the Message window. A listing appears on the printer.

### *Printing a library, its units, and its subunits*

Begin in the library containing the objects to be printed.

1. Print: `Esc` - `Q` - `Print`

A Command window appears with the `Queue.Print` command and prompts for its parameters.

2. At the Name prompt, enter the wildcard symbol `?` and press `Promote`

The progress and status are displayed in the Message window. A listing appears on the printer.

RATIONAL

## Chapter 12. Managing Links

### Listing Links—Simple Method

Begin in the world for which you want to see the links.

1. Create a Command window:
2. Enter `links.display` and press

A list of the links appears in the standard I/O window.

### Adding Links—Simple Method

Begin with the cursor in the world to which you want to add the link.

1. Create a Command window:
2. Enter `links.add` and press
3. At the `Source` prompt, enter the full pathname of the Ada unit to which you want the link to refer and press

The new link is added to the world. The link name is the simple Ada name derived from the full pathname.

### Getting the Pathname for an Environment Package

Begin in any window.

1. Create a command window:
2. Enter `library.resolve` and press
3. At the `Name_Of` prompt, enter the simple name of the Ada unit for which you want the pathname prefixed with the `\` character (for example, `\Text_io`).
4. Execute the command:

The full pathname is displayed in the I/O window. If you want to use this pathname as a parameter to another command, you can select the text of the pathname in the I/O window and then copy this region into a Command window.

Note that this shortcut for getting pathnames works for most Environment packages. If the shortcut fails, an error message appears, and you have to look for the pathname in the World ! section of the Reference Summary (in Volume 1 of the *Rational Environment Reference Manual*) or in the reference manual for the product area in question.

## Editing Links for a World

Begin in the world for which you want to edit the links.

1. Create a Command window:
2. Enter `links.edit` and press

A window displaying the links appears. You can now edit the links. See the individual editing operations that follow.

## Controlling the Link Display

Begin with the cursor in the link display.

*Toggling the order of the link display*

1. Change display order:  -

Repeating this command toggles the display so that it appears alphabetically either by source name or by link name.

*Toggling the contents of the link display*

1. Change display contents:  -

Repeating this command toggles the display so that you view one of the following: only internal links, only external links, or all links.

## Inserting a New Link

Begin with the cursor in the link display.

1. Open an insertion point:  -   
A Command window appears attached to the link display window with the Insert command and its parameter.
2. At the prompt, enter the full pathname of the Ada unit to which you want the link to refer and press

The link display is updated to show the new link. The link name is the simple Ada name derived from the full pathname.

## Deleting a Link

Begin with the cursor in the link display.

1. Move to the link you want to delete.
2. Select that link: `Object` - `—`
3. Delete the link: `Object` - `D`

The link is deleted and the link display is updated.

## Viewing the Source of a Link

Begin with the cursor in the link display.

1. Move to the link whose source you want to view.
2. Select that link: `Object` - `—`
3. Go to the definition: `Definition`

A window appears containing the definition of the Ada unit to which the link refers.

## Exiting from the Link Display

Begin with the cursor in the link display.

1. Release the link image: `Object` - `X`

The window containing the link display disappears.

## Adding a Set of Links

Begin in the world to which you want to add a set of links.

1. Create a Command window: `Create Command`
2. Enter `links.add` and press `Complete`
3. At the Source prompt, enter a name (using substitution characters and wild-cards, if desired) that specifies the complete set of links and press `Promote`

All links are added.

## Replacing a Link

Begin in the world containing the link you want to replace.

1. Create a Command window: `Create Command`
2. Enter `links.replace` and press `Complete`
3. At the Source prompt, enter the new source name you want to have associated with an existing link and press `Promote`

The source for the link is replaced.

RATIONAL



## Chapter 13. Managing Session Switches

### Editing Session Switches

Begin in any window.

1. Create a Command window: `Create Command`
2. Enter `switches.edit_session_attributes` and press `Promote`

A window displaying the session switches appears. You can now edit the switches. A session switch file called *Current\_Session\_Name\_Switches* appears in your home library, if it does not already exist.

### Controlling the Session Switch Display

Begin with the cursor in the session switch display. Two commands toggle the session switches display so that you see one of the following views: all switches or nondefault switches (switches that you have modified).

1. Change the display to all switches: `Object` - `!`
2. Change the display to nondefault switches: `Object` - `.`

## Modifying Session Switch Values

Begin with the cursor in the session switch display.

### *Modifying a Boolean switch*

1. Place the cursor on the session switch whose value you want to modify.
2. Edit the selected session switch:   
The value toggles between true and false. The session switch display is updated to show the new value.
3. Save the session switch image:

Session switches take effect at varying times: immediately, at login, or when next displaying the object image.

### *Modifying a non-Boolean switch*

1. Place the cursor on the session switch whose value you want to modify.
2. Edit the selected session switch:   
A Command window appears with the Change command and a prompt for its parameter.
3. At the prompt, enter the new parameter value and press   
The session switch display is updated to show the new value.
4. Save the session switch image:

Session switches take effect at varying times: immediately, at login, or when next displaying the object image.

## Getting Help on Session Switches

Begin with the cursor in the session switch display.

### *Getting an explanation*

1. Place the cursor on the session switch for which you want to have further information.
2. Ask for help: `Object` - `?`

An explanation of the session switch, if it exists, appears in the switch display below the selected session switch.

### *Removing an explanation*

1. Place the cursor on the explanation that you want to remove.
2. Remove the explanation: `Object` - `?`

The explanation disappears from the session switch display.

## Saving Session Switches

Begin with the cursor in the session switch display.

1. Save the image: `Enter`

A message appears in the Message window indicating that the session switches have been saved (*committed*).

## Exiting from the Session Switch Display

Begin with the cursor in the session switch display.

1. Release the switch image: `Object` - `X`

The window containing the session switch display disappears.

RATIONAL

## Chapter 14. Managing Searchlists

### Editing the Searchlist for a Session

Begin in any Command window.

1. Enter `search_list.show_list` and press `Promote`.

A window displaying the session searchlist appears. You can now edit your searchlist.

### Adding a Component to a Searchlist

Begin with the cursor in the searchlist display.

1. Move to the line where the new entry is to be added.
2. Open an insertion point: `Object` - `I`

A Command window appears with the Add command and prompts for its parameters.

3. At the Component prompt, enter the new searchlist entry and press `Promote`.

The searchlist display is updated to show the new entry.

### Deleting a Component from a Searchlist

Begin with the cursor in the searchlist display.

1. Put the cursor on the searchlist component you want to delete.
2. Select the searchlist component: `Object` - `--`
3. Delete the searchlist component: `Object` - `D`

The entry is deleted and the display is updated.

## Replacing One Component with Another

Begin with the cursor in the searchlist display.

1. Select the entry to be replaced: **Object** - **--**
2. Create a Command window: **Create Command**
3. Enter replace and press **Complete**
4. At the `New_Component` prompt, enter the new entry and press **Promote**

The old entry is replaced with the new one.

## Viewing the Library Named by a Searchlist Entry

Begin with the cursor in the searchlist display.

1. Move to the searchlist entry you want to view.
2. Go to the definition: **Definition**

A window appears containing the library.

## Exiting from the Searchlist Display

Begin with the cursor in the searchlist display.

1. Release the searchlist image: **Object** - **X**

The window containing the searchlist disappears.

## Chapter 15. Managing Jobs

### Disconnecting from a Job

1. Disconnect the job: `Control G`

A user-interrupt message is displayed in the Message window. You can now move the cursor and perform other tasks. The job continues to execute.

Note that logging out does not terminate disconnected jobs that are still executing unless these jobs attempt to perform input or output to Editor windows.

### Reconnecting to a Job

Begin in any window.

1. Determine the number of the job to be reconnected. The job number is displayed on the banner of the I/O window for the job (if used). Otherwise, to display all the jobs currently running on the system, press `What Users`
2. Get a prompt for the connect command: `Esc - Q - Job Connect`
3. At the The\_Job parameter, enter the number of the job and press `Promote`

### Killing the Current Job or the Last Job Created

Begin in any window.

1. Kill the last job: `Job Kill`

A job-abort message is displayed in the Message window.

## Killing Any Job

Begin in any window.

1. Disconnect from the current job if necessary: **Control** - **G**
2. Determine the number of the job to be killed. The job number is displayed on the banner of the I/O window for the job (if used). Otherwise, to display all the jobs currently running on the system, press **What Users**
3. Prompt for the command to kill the job: **Esc** - **Q** - **Job Kill**
4. Enter the job number at the The\_Job prompt and press **Promote**

Note that the default job number is that of the job from which you just disconnected.

A message is displayed in the Message window indicating that the job has been killed.



## Chapter 16. Customizing Your Workspace

### Building Macros

You can bind a series of keystrokes to a single key by building a macro.

#### *Defining the macro*

1. Start the definition: `Mark - Begin Of`
2. Press the keystrokes that are to make up the macro.
3. End the definition: `Mark - End Of`

#### *Executing the macro*

1. Execute the last macro you entered: `Mark - Promote`

#### *Binding the macro to a function key*

1. Press `Mark - Definition`

You are prompted in the Message window for a key to bind to the last macro entered.

2. Press the key to be bound.

The key remains bound only until you log out, unless you explicitly save it.

#### *Saving the macro*

1. Create a Command window.
2. Enter `macro.save` and press `Promote`

All macros currently bound to keys are permanently saved.

## Defining Your Own Login Procedure

Begin in your home library.

1. Create a procedure named `Login` with the commands you want to have executed each time you log into the Environment.

See “Creating an Ada Subprogram” in Chapter 8 for details.

2. Promote the procedure to the coded state: `Code Unit`

The `Login` procedure is now executed automatically as part of the login process.

## Rebinding Keys

Before starting, you may want to press `Help on Key` to see if the key is already bound.

You can rebind commands to keys in one of two ways.

Begin in any window.

### *Rebinding temporarily*

1. Create a Command window: `Create Command`
2. Enter `key.define` and press `Complete`
3. At the `Key_Name` prompt, enter the key you want to rebind to the new command.  
If you do not know the name of the key, press `Help on Key` and then press the key for which you want to know the name. The key name for that key is displayed in the Message window.
4. At the `Command_Name` prompt, enter the name of the command you want bound to this key and press `Promote`

The new key binding is in effect until you log out.

### *Rebinding permanently*

Begin in your home world.

1. Create a procedure named `Facit_Commands` by copying the text from the template in `!Machine.Editor_Data.Facit_User_Commands` into an Ada window.  
See “Creating an Ada Subprogram” in Chapter 8 for details.
2. Edit the body of `Facit_Commands` so that the case statement contains alternatives for those keys you want to rebind.
3. Promote the procedure to the installed state: `Promote`

The changes will be in effect when you next log in.

## Chapter 17. Using CMVC

CMVC is an abbreviation for Configuration Management and Version Control.

### Creating a Subsystem

Begin in the library that is to contain the subsystem.

1. Create a Command window:
2. Enter `cmvc.initial` and press
3. At the Subsystem prompt, enter the name of the subsystem and press

The command generates logging messages to the I/O window. When the command completes, the subsystem appears in the library. It contains an initial view called `Rev1_Working`.

### Adding, Changing, or Deleting Ada Units in a View

Begin in the view's world (for example, `Rev1_Working`).

1. Go to the directory called Units.
2. Add, change, or delete Ada units as necessary.

Note: You cannot change controlled objects unless they are checked out.

### Making Ada Units Controlled

Begin in the units directory for the view containing the units to be controlled.

1. Create a Command window:
2. Enter `cmvc.make_controlled` and press
3. At the `What_Object` prompt, enter the wildcard `?` and press

The command generates messages to the I/O window. All units in the view are now controlled.

Note: If units are later added to the view, they will not be controlled unless you perform the above operations again.

## Making a Subpath

Begin in the subsystem.

1. Place the cursor on the working view for the path from which the subpath is to be created (typically, `Revn_Working`).
2. Create a Command window:
3. Enter the command `cmvc.make_subpath` and press
4. At the `New_Subpath_Extension` prompt, enter the name of the subpath (for example, the name of the developer who will be working in the subpath) and press

The command displays messages in the I/O window. When it completes, a new view appears in the subsystem that is the working view for the subpath. This view has a name of the form *Pathname\_Subpathname\_Working*.

## Checking Out a Unit for Changes

Begin in the unit to be changed.

1. Create a Command window:
2. Enter `cmvc.check_out` and press
3. At the `Comments` prompt, enter the reason for the change and press

The command displays its output. When it completes, the unit can be modified.

## Checking In a Unit after Changes

Begin in the unit to be checked in after changes.

1. Create a Command window:
2. Enter `cmvc.check_out` and press
3. At the `Comments` prompt, enter a summary of the changes made and press

The command displays its output. When it completes, the unit can no longer be changed and a new generation will have been created for the unit.

## Making a Frozen Release

Begin in any library in the working view to be released or in the world for the working view. All controlled units in the view must be checked in.

1. Create a Command window:
2. Enter `cmvc.release` and press

The command generates messages to the I/O window. When it completes, a new view, which is a frozen copy of the working view, appears in the subsystem world. The Environment automatically generates a release number. The form of the name of the released view is *Pathname/Subpathname\_n\_m*.

Note: Since the released view is frozen (units cannot be edited, promoted, and so on), be sure that the units in the working view are at the proper state (typically coded) before releasing.

## Accepting Changes

Begin in the world of the view you want to make current.

*If you do not want to accept any changes that will cause units in your view to be demoted*

1. Place the cursor on the first line of the library display.  
Note: If you want to accept changes only for a specific unit, you can place the cursor on the library entry for the unit you want updated instead.
2. Create a Command window:
3. Enter `cmvc.accept_changes` and press

The command displays its output. All objects in the view are updated to the most current generation unless updating them causes demotions in your view.

*If you want to accept all changes even if they cause units in your view to be demoted*

1. Place the cursor on the first line of the library display.  
Note: If you want to accept changes only for a specific unit, you can place the cursor on the library entry for the unit you want updated instead.
2. Create a Command window:
3. Enter `cmvc.accept_changes` and press
4. At the `Allow_Demotion` prompt, enter `true` and press

The command displays its output. All objects in the view are updated to the most current generation.

## Getting Information

### *Determining out-of-date units in a view*

Begin anywhere in the view.

1. Create a Command window:
2. Enter `cmvc.show_out`, press , and then press

The list of units that are not the most recent generations available are displayed in the I/O window.

### *Determining units that are checked out in a view*

Begin anywhere in the view.

1. Create a Command window:
2. Enter `cmvc.show_checked_out_in_view` and press

The units that are checked out in the view are displayed in the I/O window.

### *Determining units you have checked out (any view)*

Begin anywhere in a view that defines the set of units that you may have checked out in that view or other views sharing its reservation tokens.

1. Create a Command window:
2. Enter `cmvc.show_checked_out_by_user` and press

A list of units you have checked out and the views to which they are checked out is displayed in the I/O window.

### *Change history for a unit*

Begin in the unit of interest.

1. Create a Command window:
2. Enter `cmvc.show_history_by`, press , and then press

The history for the unit is displayed in the I/O window.

### *General information on a unit*

Begin in the unit of interest.

1. Create a Command window:
2. Enter `cmvc.show` and press

The command generates output to the I/O window. This output tells you what views share reservation tokens (this is, are subject to check-in/check-out synchronization of changes). It also tells you what generation of the unit you have and how many generations exist, who has the unit checked out, and so on.

## Chapter 18. Networking

### Logging Into Another System with Telnet

Begin in any window.

1. Create a Command window:
2. Enter `telnet.connect` and press
3. At the `Remote_Machine` prompt, enter the name of the remote machine (enclosed in double quotes) and press

Messages appear in the I/O window, the screen clears, and you are now connected to the remote machine and can begin logging in.

### Interrupting a Telnet Session

Interrupting a Telnet session leaves the connection intact and takes you back to your original machine. You can later resume the interrupted session to continue work on the remote machine.

Begin in a Telnet session connected to a remote machine.

1. Interrupt the session:

The connection to the remote is interrupted and your original R1000 session reappears on the screen.

Note: If the above steps do not work, the key that interrupts Telnet sessions may have been changed from  to another key. Check with your system manager.

## Resuming a Telnet Session

Begin in any window.

1. Create a Command window:
2. Enter `telnet.connect` and press
3. At the `Remote_Machine` prompt, enter the name of the remote machine with which the connection was interrupted (enclosed in double quotes) and press

The screen clears and the interrupted connection with the remote system is resumed.

You have to press the key that redraws the screen on the remote system (if the remote machine is another R1000, press   to redraw the screen).

## Terminating a Telnet Session

*If you are still connected to the remote machine*

1. Log off the remote machine.

For most remote Telnet servers, this terminates the Telnet session and returns you to your original session.

If you are not returned to your original session, interrupt from the session as described above and then follow the steps below.

*If you are in your original R1000 session*

Begin in any window.

1. Create a Command window:
2. Enter `telnet.disconnect` and press
3. At the `Remote_Machine` prompt, enter the name of the remote system to which the session you want to terminate is connected.
4. Execute the command:

The Telnet session is disconnected.



## Copying a Single Object or Library onto Another R1000

*Copying into an identical library structure keeping the same simple names for the items copied*

Begin in the object or the library to be copied onto the other machine. Make sure that there are no selections in this window.

1. Create a Command window:
2. Enter `archive.copy` and press
3. At the `Use_Prefix` prompt, enter the name of the machine onto which to copy prefixed with the string `“!!”`—for example, `“!!m1”`.
4. Execute the command:

The object and its children, or the library and its contents, are copied onto the designated machine in the same library structure and with the same names as on the source machine. Note that if the library structure does not already exist on the target machine, it is created automatically.

*Copying into another library structure keeping the same simple names for the items copied*

Begin in the object or the library to be copied onto the other machine. Make sure that there are no selections in this window.

1. Create a Command window:
2. Enter `archive.copy` and press
3. At the `Use_Prefix` prompt, enter the name of the machine and the pathname of the target library to contain the object or library—for example, `“!!m1!users-.sjl.example”`.
4. Execute the command:

The object and its children, or the library and its contents, are copied onto the designated machine in the specified library structure and with the same names as on the source machine. Note that if the library structure does not already exist on the target machine, it is created automatically.

## Copying Objects or Libraries from Another R1000

*Copying into an identical library structure keeping the same simple names for the items copied*

Begin in any window.

1. Create a Command window:
2. Enter `archive.copy` and press
3. At the Objects prompt, enter the name of the machine and the pathname of the object from which to copy—for example, “`!!m1!users.sjl.some_object`”.
4. Execute the command:

The object and its children, or the library and its contents, are copied from the designated machine into the same library structure and with the same names as on the source machine. Note that if the library structure does not already exist on your machine, it is created automatically.

*Copying into another library structure keeping the same simple names for the items copied*

Begin in the library to contain the copied item.

1. Create a Command window:
2. Enter `archive.copy` and press
3. At the Objects prompt, enter the name of the machine and the pathname of the target library to contain the object or library—for example, “`!!m1!users.sjl.example`”.
4. At the Use\_Prefix prompt, enter `$`
5. At the For\_Prefix prompt, enter the name of the library in which the object is located on the source machine without the machine name—for example, “`!users.sjl`”.
6. Execute the command:

The object and its children, or the library and its contents, are copied from the designated machine into the specified library structure and with the same names as on the source machine.

## Copying Objects onto a Non-R1000 System

Begin in the object to be moved.

1. Create a Command window:
2. Enter ftp.put and press
3. At the To\_Remote\_File prompt, enter the simple name (without a directory name prefix) of the object on the target system.
4. At the Remote\_Machine prompt, enter the name of the remote machine (enclosed in double quotes).
5. At the Username prompt, enter your username on the remote machine (enclosed in double quotes).
6. At the Password prompt, enter your password on the remote machine (enclosed in double quotes).
7. If you want the object to go to a directory on the remote machine other than your home directory, at the Remote\_Directory prompt, enter the full pathname of the directory to contain the object on the target (enclosed in double quotes).
8. Execute the Command:

## Copying Objects from a Non-R1000 System

Begin in the library to contain the object to be moved.

1. Create a command window:
2. Enter ftp.get and press
3. At the From\_Remote\_File prompt, enter the simple name (without a directory name prefix) of the object on the remote system.
4. At the To\_Local\_File prompt, enter the name you want the object to have on your system.
5. At the Remote\_Machine prompt, enter the name of the remote machine (enclosed in double quotes).
6. At the Username prompt, enter your username on the remote machine (enclosed in double quotes).
7. At the Password prompt, enter your password on the remote machine (enclosed in double quotes).
8. If the object on the remote machine is not in your home directory, at the Remote\_Directory prompt, enter the full pathname of the directory on the remote machine containing the object.
9. Execute the command:

RATIONAL

# **Rational Environment**

## **Basic Operations**

**Basic Keymap: Facit Terminal**

Copyright © 1985, 1986, 1987 by Rational

Document Control Number: 8001A-51 (803-002325)

Rev. 6.0, November 1985  
Rev. 6.1, March 1986  
Rev. 6.2, July 1986  
Rev. 7.0, July 1987 (Delta)

This document subject to change without notice.

Note the Reader's Comments form on the last page of this book, which requests the user's evaluation to assist Rational in preparing future documentation.

Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).

Rational and R1000 are registered trademarks and Rational Environment and Rational Subsystems are trademarks of Rational.

VT100 is a trademark of Digital Equipment Corporation.

Rational  
1501 Salado Drive  
Mountain View, California 94043

## Contents

<b>How to Use the Basic Keymap</b> . . . . .	1
Keymap Overview . . . . .	1
Quick Reference to Key Bindings . . . . .	1
Detailed Reference to Key Bindings . . . . .	1
Master Reference to Key Bindings by Command . . . . .	1
Environment Key Combinations . . . . .	2
Item-Operation Key Combinations . . . . .	2
Patterns among Item-Operation Combinations . . . . .	2
Modified Key Combinations . . . . .	3
Basic and Accelerated Keystrokes . . . . .	3
Keymap Notation . . . . .	3
Symbols . . . . .	3
Numeric Arguments . . . . .	3
Case Sensitivity of Key Bindings . . . . .	4
<b>Quick Reference to Key Bindings</b> . . . . .	5
Getting Help . . . . .	6
Traversing the Environment . . . . .	6
Logging Off . . . . .	6
Selecting Items . . . . .	6
Executing Commands . . . . .	6
Managing Windows . . . . .	7
Moving within an Image . . . . .	7
Writing Text Files . . . . .	7
General Editing Operations . . . . .	8
Writing Ada Programs . . . . .	8
Debugging Ada Programs . . . . .	9
Managing Libraries . . . . .	9
Using CMVC . . . . .	9

Managing Links . . . . .	9
Using Environment I/O Resources . . . . .	10
Managing Jobs . . . . .	10
<b>Detailed Reference to Key Bindings . . . . .</b>	<b>11</b>
Getting Help and Other Information . . . . .	12
Traversing the Environment . . . . .	12
Logging Off . . . . .	12
Selecting Items . . . . .	13
Executing Commands . . . . .	13
Managing Windows . . . . .	13
Moving between Windows . . . . .	13
Resizing and Repositioning Windows . . . . .	14
Redrawing the Screen . . . . .	14
Retaining Windows . . . . .	14
Removing Windows . . . . .	14
Finding Windows . . . . .	14
Moving within an Image . . . . .	15
By Character . . . . .	15
By Word . . . . .	15
By Underline or Prompt . . . . .	15
By Line . . . . .	15
In a Region . . . . .	16
By Tabs . . . . .	16
By Scrolling . . . . .	16
By Marking Your Place . . . . .	16
General Editing Operations . . . . .	17
Selecting an Arbitrary Region . . . . .	17
Moving and Copying Text . . . . .	17
Deleting Text . . . . .	17
Searching and Replacing Text . . . . .	17
Entering Text . . . . .	18
Transposing Text . . . . .	18
Controlling Case . . . . .	18
Holding and Retrieving Text . . . . .	19
Formatting Text . . . . .	19
Writing Text Files . . . . .	20
Accessing Text Files . . . . .	20



Saving Changes . . . . .	20
Terminating Edit . . . . .	20
Selecting Substructures within Text . . . . .	20
Writing Ada Programs . . . . .	21
Creating Ada Programs . . . . .	21
Accessing Ada Programs . . . . .	21
Saving Changes and Terminating Edit . . . . .	21
Checking for Errors . . . . .	21
Changing the Compilation State . . . . .	22
Changing to a Higher Compilation State . . . . .	22
Changing to a Lower Compilation State . . . . .	22
Selecting Structures within Ada Programs . . . . .	22
Modifying Ada Programs . . . . .	23
Entering Comments and Special Strings . . . . .	23
Browsing Ada Programs . . . . .	23
Checking Using Occurrences . . . . .	23
Debugging Ada Programs . . . . .	24
Stepping and Executing . . . . .	24
Setting and Removing Breakpoints . . . . .	24
Viewing Stacks . . . . .	24
Displaying and Modifying Variables . . . . .	24
Handling Exceptions . . . . .	24
Managing Libraries . . . . .	25
Creating Libraries . . . . .	25
Manipulating Objects in Libraries . . . . .	25
Controlling Library Display . . . . .	25
Using CMVC . . . . .	25
Managing Links . . . . .	26
Accessing Links . . . . .	26
Removing the Link Editor . . . . .	26
Selecting Links . . . . .	26
Modifying Links . . . . .	26
Traversing Linked Ada Units . . . . .	26
Controlling the Display . . . . .	26
Managing Searchlists . . . . .	27
Accessing the Searchlist . . . . .	27
Removing the Searchlist Editor . . . . .	27
Selecting Entries . . . . .	27

Modifying the Searchlist . . . . .	27
Using Keyboard Macros . . . . .	28
Using Environment I/O Resources . . . . .	28
Managing Jobs . . . . .	28
<b>Master Reference to Key Bindings by Command . . . . .</b>	<b>29</b>

## How to Use the Basic Keymap

The Rational Environment Basic Keymap is designed to acquaint new users with the keys that have been bound to Environment commands. Users have the option of modifying these key bindings for their own use, following procedures described in Rational Environment Basic Operations, also in this manual.

Note that there is a more complete reference to Environment key bindings in the Rational Environment Keymap, in Volume 1 of the *Rational Environment Reference Manual*. It is intended as the primary key reference for Environment users.

### Keymap Overview

The Keymap has been divided into the following three sections. The first two sections apply to the Facit terminal only. The last section includes key bindings for both the Facit terminal and the Rational Terminal.

#### Quick Reference to Key Bindings

The Quick Reference is a guide to the most commonly used key combinations, organized by topic. The Quick Reference entry for each key combination includes:

- A brief description of what the combination does
- The full name of the command that is bound to it

#### Detailed Reference to Key Bindings

The Detailed Reference provides a nearly complete list of key combinations, organized by topic and subtopic. The Detailed Reference entry for each key combination includes:

- A brief description of what the combination does
- The full name of the command that is bound to it
- Alternative key bindings, including accelerated key combinations (see "Basic and Accelerated Keystrokes," below)

#### Master Reference to Key Bindings by Command

This section provides a complete, alphabetic list of the commands that are bound to keys on both the Facit terminal and the Rational Terminal. Each entry includes:

- The full name of an Environment command
- The key combination(s) to which the command is bound on the Facit terminal
- The key combination(s) to which the command is bound on the Rational Terminal

### Environment Key Combinations

Environment commands are bound to two types of key combinations:

- Item-operation combinations
- Modified key combinations

These two types of key combinations differ in how they are executed.

### Item-Operation Key Combinations

Each item-operation key combination contains an item key (`Esc`, `Object`, `Region`, `Window`, `Image`, `Line`, `Word`, or `Mark`) followed by an operation key (either alphabetic or nonalphabetic). The item key identifies the item affected by the operation; the operation key identifies the action that applies to the indicated item.

The keystrokes must be sequential in an item-operation key combination. To execute an item-operation key:

1. Press and release the item key.
2. Press and release the operation key.

The notation indicates sequential keystrokes by separating them with a hyphen:

`item key` - `operation key`.

### Patterns among Item-Operation Combinations

In general, commands that execute similar operations are bound to combinations that contain a common operation key. Some examples include:

`Item` - `C`

Commands that copy items are bound to combinations such as `Line` - `C`, `Region` - `C`, and `Object` - `C`, which share the operation key `C`.

`Item` - `D`

Commands that delete items are bound to combinations such as `Line` - `D`, `Word` - `D`, and `Window` - `D`, which share the operation key `D`.

`Item` - `T`

Commands that transpose items are bound to combinations such as `Word` - `T`, `Line` - `T`, and `Window` - `T`, which share the operation key `T`.

## Modified Key Combinations

Each modified key combination contains one or more modifier keys (`[Shift]`, `[Control]`), along with another key (either alphabetic or nonalphabetic). Modifier keys are never used with item keys.

The keystrokes must overlap in a modified key combination. To execute a modified combination:

1. Press and hold the modifier key(s).
2. While holding down the modifier key(s), press the key to be modified.

The notation indicates overlapping keystrokes by naming the keys adjacently:

`[modifier key][other key]`.

## Basic and Accelerated Keystrokes

Certain key combinations (namely, item-operation combinations and modified function keys) are considered *basic* combinations because they involve explicitly labeled keys, such as `[Word]` or `[Definition]`. Basic key bindings are recommended if you are new to the Environment, because they are easy to remember.

However, experienced users may find *accelerated* key bindings more convenient. Accelerated bindings generally involve the modifier keys in combination with keys on the main keyboard so that you can use them without moving your hands away from normal typing position.

Many commands are bound to both basic and accelerated key combinations. As an example, you can delete a word using either `[Word] - [D]` or the corresponding accelerated key combination, `[Esc] - [D]`.

## Keymap Notation

The following notations apply to all sections of the Keymap except the “Master Reference to Key Bindings by Command.”

### Symbols

`[key1] - [key2]` Press and release `[key1]`; then press `[key2]`.

`[key1][key2]` Press and hold `[key1]` while pressing `[key2]`.

`[numeric 1]` Press `[1]` on the numeric keypad.

### Numeric Arguments

You can give a numeric argument to many of the commands that are bound to keys. Indicate the desired number using the numeric keypad, and then press the key combination bound to the command. For example, `[Word] - [D]` deletes one word; the following combination deletes four words: `[numeric 4] - [Word] - [D]`.

## How to Use the Basic Keymap

Indicate negative numbers by pressing `numeric -` first. For example, the following combination shrinks a window by seven lines ("expands" it by `-7` lines):

`numeric -` - `numeric 7` - `Window` - `!`

### Case Sensitivity of Key Bindings

Although keys are shown as uppercase, the unshifted equivalent also works. This is true for the nonalphabetic characters as well. For example, `Object` - `d` is equivalent to `Object` - `D` and `Object` - `!` is equivalent to `Object` - `I`.

**Quick Reference to Key Bindings**

### Getting Help

<i>Description</i>	<i>Basic Keys</i>	<i>Command</i>
Determine what help is available	<b>Help on Help</b>	<b>What.Does</b>
Get help on item	<b>Help</b>	<b>What.Does</b>
Get help on key	<b>Help on Key</b>	<b>Editor.Key.Name</b>
Display Help window	<b>Help Window</b>	<b>Editor.Image.Find</b>

### Traversing the Environment

<i>Description</i>	<i>Basic Keys</i>	<i>Command</i>
View object cursor is on	<b>Definition</b>	<b>Common.Definition</b>
Get to parent object	<b>Enclosing</b>	<b>Common.Enclosing</b>
Get to your home library	<b>Esc - ?</b>	<b>What.Home_Library</b>

### Logging Off

<i>Description</i>	<i>Basic Keys</i>	<i>Command</i>
Log off, unless changes aren't saved	-	<b>Editor.Quit</b>
Log off, ignoring unsaved changes	-	<b>Editor.Quit(True)</b>

### Selecting Items

<i>Description</i>	<i>Basic Keys</i>	<i>Command</i>
Select successively larger structures	<b>Object - ←</b>	<b>Common.Object.Parent</b>
Select successively smaller structures	<b>Object - →</b>	<b>Common.Object.Child</b>
Select previous structure, same level	<b>Object - ↑</b>	<b>Common.Object.Previous</b>
Select next structure, same level	<b>Object - ↓</b>	<b>Common.Object.Next</b>
Turn off selection cursor is in	<b>Control X</b>	<b>Editor.Set.Designation_Off</b>

### Executing Commands

<i>Description</i>	<i>Basic Keys</i>	<i>Command</i>
Create a Command window	<b>Create Command</b>	<b>Common.Create_Command</b>
Complete command name and parameters	<b>Complete</b>	<b>Common.Complete</b>
Execute a command	<b>Promote</b>	<b>Common.Promote</b>
Move to the next parameter	<b>Esc - N</b>	<b>Editor.Cursor.Next</b>
Move to the previous parameter	<b>Esc - U</b>	<b>Editor.Cursor.Previous</b>
Turn a prompt into text	<b>Control X</b>	<b>Editor.Set.Designation_Off</b>
Redisplay the previous command (undo)	<b>Object - U</b>	<b>Common.Undo</b>
Redisplay the next command (redo)	<b>Object - R</b>	<b>Common.Redo</b>



## Managing Windows

<i>Description</i>	<i>Basic Keys</i>	<i>Command</i>
Move to the next window	<b>Window</b> - <b>I</b>	<b>Editor.Window.Next</b>
Move to the previous window	<b>Window</b> - <b>P</b>	<b>Editor.Window.Previous</b>
Join with the next window	<b>Window</b> - <b>J</b>	<b>Editor.Window.Join (1)</b>
Transpose current window with previous	<b>Window</b> - <b>T</b>	<b>Editor.Window.Transpose</b>
Realign windows	<b>Window</b> - <b>Format</b>	<b>Editor.Window.Focus</b>
Redraw the screen	<b>Control</b> - <b>L</b>	<b>Editor.Screen.Redraw</b>
Lock a window on the screen	<b>Window</b> - <b>Promote</b>	<b>Editor.Window.Promote</b>
Release a locked window	<b>Window</b> - <b>Demote</b>	<b>Editor.Window.Demote</b>
Remove a window temporarily	<b>Window</b> - <b>D</b>	<b>Editor.Window.Delete</b>
Release image permanently, saving changes	<b>Object</b> - <b>X</b>	<b>Common.Release</b>
Display the Window Directory	<b>Window</b> - <b>Definition</b>	<b>Editor.Window.Directory</b>
View Window Directory entry cursor is on	<b>Definition</b>	<b>Common.Definition</b>
Delete selected Window Directory entry	<b>Object</b> - <b>D</b>	<b>Common.Object.Delete</b>

## Moving within an Image

<i>Description</i>	<i>Basic Keys</i>	<i>Command</i>
Move to beginning of line	<b>Line</b> - <b>Begin Of</b>	<b>Editor.Line.Beginning-Of</b>
Move to end of line	<b>Line</b> - <b>End Of</b>	<b>Editor.Line.End-Of</b>
Scroll up	<b>Image</b> - <b>↑</b>	<b>Editor.Image.Up</b>
Scroll down	<b>Image</b> - <b>↓</b>	<b>Editor.Image.Down</b>
Scroll to top of image	<b>Image</b> - <b>Begin Of</b>	<b>Editor.Image.Beginning-Of</b>
Scroll to end of image	<b>Image</b> - <b>End Of</b>	<b>Editor.Image.End-Of</b>

## Writing Text Files

<i>Description</i>	<i>Basic Keys</i>	<i>Command</i>
Create a new text file	<b>Create Text</b>	<b>Text.Create</b>
View existing text file	<b>Definition</b>	<b>Common.Definition</b>
Edit existing text file	<b>Edit</b>	<b>Common.Edit</b>
Revert to last saved version	<b>Object</b> - <b>L</b>	<b>Common.Revert</b>
Save, leaving open for editing	<b>Enter</b>	<b>Common.Commit</b>
Save, making read only	<b>Promote</b>	<b>Common.Promote</b>

## General Editing Operations

<i>Description</i>	<i>Basic Keys</i>	<i>Command</i>
Select start of region	<b>Region</b> . <b>[</b>	<b>Editor.Region.Start</b>
Select end of region	<b>Region</b> . <b>]</b>	<b>Editor.Region.Finish</b>
Copy a selected item	<b>Region</b> . <b>C</b>	<b>Editor.Region.Copy</b>
Move a selected item	<b>Region</b> . <b>M</b>	<b>Editor.Region.Move</b>
Delete character — forward	<b>Control</b> <b>D</b>	<b>Editor.Char.Delete_Forward</b>
Delete character — backward	<b>Delete</b>	<b>Editor.Char.Delete_Backward</b>
Delete word	<b>Word</b> . <b>D</b>	<b>Editor.Word.Delete</b>
Delete line	<b>Line</b> . <b>D</b>	<b>Editor.Line.Delete</b>
Delete selected item	<b>Region</b> . <b>D</b>	<b>Editor.Region.Delete</b>
Search for next occurrence	<b>Control</b> <b>F</b>	<b>Editor.Search.Next</b>
Replace next occurrence	<b>Esc</b> . <b>F</b>	<b>Editor.Search.Replace_Next</b>

## Writing Ada Programs

<i>Description</i>	<i>Basic Keys</i>	<i>Command</i>
Create an Ada unit in library	<b>Object</b> . <b>I</b>	<b>Common.Object.Insert</b>
Build a body	<b>Create Body</b>	<b>Ada.Create_Body</b>
Build a private part	<b>Create Private</b>	<b>Ada.Create_Private</b>
Demote to source, open for editing	<b>Edit</b>	<b>Common.Edit</b>
Revert to last saved version	<b>Object</b> . <b>L</b>	<b>Common.Revert</b>
Save, leaving open for editing	<b>Enter</b>	<b>Common.Commit</b>
Save, regardless of errors	<b>Enter</b>	<b>Common.Commit</b>
Complete and check syntax	<b>Format</b>	<b>Common.Format</b>
Check for semantic errors	<b>Semanticize</b>	<b>Common.Semanticize</b>
Explain underlined error	<b>Object</b> . <b>?</b>	<b>Common.Explain</b>
Move to next underlined error	<b>Esc</b> . <b>N</b>	<b>Editor.Cursor.Next</b>
Move to previous underlined error	<b>Esc</b> . <b>U</b>	<b>Editor.Cursor.Previous</b>
Promote to next higher state	<b>Promote</b>	<b>Common.Promote</b>
Change to source state	<b>Source Unit</b>	<b>Ada.Source_Unit</b>
Change to installed state	<b>Install Unit</b>	<b>Ada.Install_Unit</b>
Change to coded state	<b>Code Unit</b>	<b>Ada.Code_Unit</b>
Demote to next lower state	<b>Demote</b>	<b>Common.Demote</b>
Compile unit and those it depends on	<b>Code (This World)</b>	<b>Compilation.Make</b>
Demote units and its dependents	<b>Source (This World)</b>	<b>Compilation.Demote</b>
Get to other part of Ada unit	<b>Other Part</b>	<b>Ada.Other_Part</b>

### Debugging Ada Programs

<i>Description</i>	<i>Basic Keys</i>	<i>Command</i>
Execute program with Debugger on	<b>Esc</b> - <b>Promote</b>	<b>Command.Debug</b>
Continue program execution	<b>Execute</b>	<b>Debug.Execute</b>
Step one statement	<b>Run</b>	<b>Debug.Run</b>
Step one statement at same level	<b>Run Local</b>	<b>Debug.Run (Local)</b>
Display values of variables	<b>Put</b>	<b>Debug.Put</b>
Set breakpoints	<b>Break</b>	<b>Debug.Break</b>
Display breakpoints	<b>Show Breaks</b>	<b>Debug.Show</b>

### Managing Libraries

<i>Description</i>	<i>Basic Keys</i>	<i>Command</i>
Create a world	<b>Create World</b>	<b>Library.Create_World</b>
Create a directory	<b>Create Directory</b>	<b>Library.Create_Directory</b>
Delete selected object from library	<b>Object</b> - <b>D</b>	<b>Common.Object.Delete</b>
Print image or selected object	<b>Print</b>	<b>Queue.Print</b>
Toggle information in library display	<b>Object</b> - <b>?</b>	<b>Common.Explain</b>
Show access list for designated object	<b>Show Access List</b>	<b>Access_List.Display</b>

### Using CMVC

<i>Description</i>	<i>Basic Keys</i>	<i>Command</i>
Check out designated object	-	<b>Cmvc.Check_Out</b>
Check in designated object	-	<b>Cmvc.Check_In</b>
Accept changes for designated object	-	<b>Cmvc.Accept_Changes</b>
Show objects that are checked out		
In this view	-	<b>Cmvc.Show_Checked_Out_In_View</b>
By you, any view	-	<b>Cmvc.Show_Checked_Out_By_User</b>
Show info about designated object	-	<b>Cmvc.Show</b>
Show out-of-date objects in this view	-	<b>Cmvc.Show_Out_Of_Date_Objects</b>

### Managing Links

<i>Description</i>	<i>Basic Keys</i>	<i>Command</i>
List links	-	<b>Links.Display</b>
Add a new link	-	<b>Links.Add</b>

### Using Environment I/O Resources

<i>Description</i>	<i>Basic Keys</i>	<i>Command</i>
Indicate end of input to program	<code>numeric .</code>	<code>Text.End.Of.Input</code>
Commit interactive input	<code>Promote</code>	<code>Common.Promote</code>

### Managing Jobs

<i>Description</i>	<i>Basic Keys</i>	<i>Command</i>
Disconnect job from terminal	<code>Control G</code>	<code>Job.Interrupt</code>
Kill job	<code>Job Kill</code>	<code>Job.Kill(0)</code>

## **Detailed Reference to Key Bindings**

### Getting Help and Other Information

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
Determine what help is available	Help on Help		What.Does
Get help on item	Help		What.Does
Get help on key	Help on Key	Esc - Q	Editor.Key.Name
Display Help window	Help Window		Editor.Image.Find
Explain underlined error	Object - ?		Common.Explain
Show time and date	What Time		What.Time
Show system load	What Load		What.Load (True)
Show current users	What Users		What.Users (True)
Show lock information for object in window	What Locks		What.Locks
Show full name of object in window	What Object		What.Object
Show access list for designated object	Show Access List		Access-List.Display

### Traversing the Environment

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
Display the Window Directory	Window - Definition	Window - ?	Editor.Window.Directory
Display object cursor is on	Definition		Common.Definition
Display object, same window	Definition In Place		Common.Definition
Display parent object	Enclosing		Common.Enclosing
Display parent object, same window	Enclosing In Place		Common.Enclosing
Display parent library, same window	Enclosing Library		Common.Enclosing
Display your home library	Esc - !		What.Home-Library
Set mark at current location	Mark - j		Editor.Mark.Push
Cycle through marks in stack	Mark - --	Esc - M	Editor.Mark.Next
Cycle back through marks in stack	Mark - --		Editor.Mark.Previous
Return to most recent mark	Mark - !		Editor.Mark.Top

### Logging Off

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
Log off, unless changes aren't saved	-	-	Editor.Quit
Log off, ignoring unsaved changes	-	-	Editor.Quit(True)

### Selecting Items

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
Select successively larger structures	<b>Object</b> - <b>--</b>		<b>Common.Object.Parent</b>
Select successively smaller structures	<b>Object</b> - <b>--</b>		<b>Common.Object.Child</b>
Select previous structure, same level	<b>Object</b> - <b>f</b>		<b>Common.Object.Previous</b>
Select next structure, same level	<b>Object</b> - <b>l</b>		<b>Common.Object.Next</b>
Select first structure	<b>Object</b> - <b>Begin Of</b>	<b>Object</b> - <b>B</b>	<b>Common.Object.First_Child</b>
Select last structure	<b>Object</b> - <b>End Of</b>	<b>Object</b> - <b>E</b>	<b>Common.Object.Last_Child</b>
Turn off selection cursor is in	<b>Control</b> <b>X</b>		<b>Editor.Set.Designation_Off</b>

### Executing Commands

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
Create a Command window	<b>Create Command</b>		<b>Common.Create_Command</b>
Complete command name and parameters	<b>Complete</b>		<b>Common.Complete</b>
Execute a command	<b>Promote</b>		<b>Common.Promote</b>
Execute command in background	<b>Shift</b> <b>Promote</b>		<b>Command.Spawn</b>
Move to the next parameter prompt	<b>Esc</b> - <b>N</b>		<b>Editor.Cursor.Next</b>
Move to the previous parameter prompt	<b>Esc</b> - <b>U</b>		<b>Editor.Cursor.Previous</b>
Turn a prompt into text	<b>Control</b> <b>X</b>		<b>Editor.Set.Designation_Off</b>
Redisplay the previous command (undo)	<b>Object</b> - <b>U</b>		<b>Common.Undo</b>
Redisplay the next command (redo)	<b>Object</b> - <b>R</b>		<b>Common.Redo</b>
Provide prompts for the next key pressed	<b>Esc</b> - <b>Q</b>		<b>Editor.Key.Prompt</b>

### Managing Windows

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
<b>Moving between Windows</b>			
Move to the next window	<b>Window</b> - <b>d</b>	<b>Esc</b> - <b>V</b> , <b>Shift</b> <b>i</b>	<b>Editor.Window.Next</b>
Move to the previous window	<b>Window</b> - <b>f</b>	<b>Esc</b> - <b>Z</b> , <b>Shift</b> <b>f</b>	<b>Editor.Window.Previous</b>
Move to next attached window	<b>Window</b> - <b>--</b>		<b>Editor.Window.Child</b>
Move to previous attached window	<b>Window</b> - <b>--</b>		<b>Editor.Window.Parent</b>

**Managing Windows (Continued)**

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
<b>Resizing and Repositioning Windows</b>			
Join with the next window	<b>Window</b> - <b>J</b>		Editor.Window.Join (1)
Join with the previous window	<b>Window</b> - <b>Delete</b>		Editor.Window.Join (-1)
Expand a window 4 lines	<b>Window</b> - <b>↑</b>		Editor.Window.Expand
Shrink a window 4 lines	<b>Window</b> - <b>↓</b>		Editor.Window.Expand (-4)
Transpose current window with previous	<b>Window</b> - <b>T</b>		Editor.Window.Transpose
Realign windows	<b>Window</b> - <b>Format</b>		Editor.Window.Focus
Copy a window	<b>Window</b> - <b>C</b>		Editor.Window.Copy
<b>Redrawing the Screen</b>			
Redraw the screen	<b>Control</b> - <b>L</b>		Editor.Screen.Redraw
Erase the screen, resetting the terminal	<b>Esc</b> - <b>L</b>		Editor.Screen.Clear
<b>Retaining Windows</b>			
Lock a window on the screen	<b>Window</b> - <b>Promote</b>		Editor.Window.Promote
Release a locked window	<b>Window</b> - <b>Demote</b>	<b>Window</b> - <b>Edit</b>	Editor.Window.Demote
<b>Removing Windows</b>			
Remove a window temporarily	<b>Window</b> - <b>D</b> , <b>Window</b> - <b>K</b> , <b>Window</b> - <b>X</b>		Editor.Window.Delete
Release image, discarding changes	<b>Object</b> - <b>G</b>		Common.Abandon
Release image, saving changes	<b>Object</b> - <b>X</b>		Common.Release
Delete selected Window Directory entry	<b>Object</b> - <b>D</b>		Common.Object.Delete
<b>Finding Windows</b>			
Display Window Directory	<b>Window</b> - <b>Definition</b>	<b>Window</b> - <b>?</b>	Editor.Window.Directory
Display Window Directory entry	<b>Definition</b>		Common.Definition



## Moving within an Image

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
<b>By Character</b>			
Move right 1 character	<code>--</code>	<code>Control J</code>	<code>Editor.Cursor.Right</code>
Move right 8 characters	<code>numeric 8 - --</code>	<code>Esc - Control J</code>	<code>Editor.Cursor.Right(8)</code>
Move left 1 character	<code>--</code>	<code>Control H</code>	<code>Editor.Cursor.Left</code>
Move left 8 characters	<code>numeric 8 - --</code>	<code>Esc - Control H</code>	<code>Editor.Cursor.Left(8)</code>
<b>By Word</b>			
Move to next word	<code>Word - --</code>	<code>Esc - J</code>	<code>Editor.Word.Next</code>
Move to previous word	<code>Word - --</code>	<code>Esc - H</code>	<code>Editor.Word.Previous</code>
Move to beginning of word	<code>Word - Begin Of</code>	<code>Esc - A, Esc - B</code>	<code>Editor.Word.Beginning_Of</code>
Move to end of word	<code>Word - End Of</code>	<code>Esc - E</code>	<code>Editor.Word.End_Of</code>
<b>By Underline or Prompt</b>			
Move to next underline or prompt	<code>Esc - N</code>		<code>Editor.Cursor.Next</code>
Move to previous underline or prompt	<code>Esc - U</code>		<code>Editor.Cursor.Previous</code>
<b>By Line</b>			
Move up 1 line	<code>↑</code>	<code>Control U</code>	<code>Editor.Cursor.Up</code>
Move up 8 lines	<code>numeric 8 - ↑</code>	<code>Esc - Control U</code>	<code>Editor.Cursor.Up(8)</code>
Move down 1 line	<code>↓</code>	<code>Control N</code>	<code>Editor.Cursor.Down</code>
Move down 8 lines	<code>numeric 8 - ↓</code>	<code>Esc - Control N</code>	<code>Editor.Cursor.Down(8)</code>
Move to beginning of line	<code>Line - Begin Of</code>	<code>Control B</code>	<code>Editor.Line.Beginning_Of</code>
Move to end of line	<code>Line - End Of</code>	<code>Control E</code>	<code>Editor.Line.End_Of</code>

**Moving within an Image (Continued)**

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
<b>In a Region</b>			
Move to beginning of region	<b>Region</b> - <b>Begin Of</b>	<b>Region</b> - <b>B</b>	Editor.Region.Beginning-Of
Move to end of region	<b>Region</b> - <b>End Of</b>	<b>Region</b> - <b>E</b>	Editor.Region.End-Of
<b>By Tabs</b>			
Tab forward	<b>Control</b> <b>I</b>		Editor.Char.Tab-Forward
Tab backward	<b>Esc</b> - <b>Control</b> <b>I</b>		Editor.Char.Tab-Backward
<b>By Scrolling</b>			
Scroll up	<b>Image</b> - <b>↑</b>	<b>Control</b> <b>Z</b>	Editor.Image.Up
Scroll down	<b>Image</b> - <b>↓</b>	<b>Control</b> <b>V</b>	Editor.Image.Down
Scroll right	<b>Image</b> - <b>→</b>		Editor.Image.Right
Scroll left	<b>Image</b> - <b>←</b>		Editor.Image.Left
Scroll to top of image	<b>Image</b> - <b>Begin Of</b>	<b>Image</b> - <b>B</b>	Editor.Image.Beginning-Of
Scroll to end of image	<b>Image</b> - <b>End Of</b>	<b>Image</b> - <b>E</b>	Editor.Image.End-Of
Scroll current line to top	<b>Window</b> - <b>Begin Of</b>	<b>Window</b> - <b>B</b>	Editor.Window.Beginning-Of
Scroll current line to bottom	<b>Window</b> - <b>End Of</b>	<b>Window</b> - <b>E</b>	Editor.Window.End-Of
<b>By Marking Your Place</b>			
Set mark at cursor position	<b>Mark</b> - <b>i</b>	<b>Control</b> <b>O</b>	Editor.Mark.Push
Cycle through marks in stack	<b>Mark</b> - <b>→</b>	<b>Esc</b> - <b>M</b>	Editor.Mark.Next
Cycle back through marks in stack	<b>Mark</b> - <b>←</b>		Editor.Mark.Previous
Return to most recent mark	<b>Mark</b> - <b>↑</b>		Editor.Mark.Top

## General Editing Operations

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
<b>Selecting an Arbitrary Region</b>			
Select start of region	<b>Region</b> - <b>[</b>	<b>Esc</b> - <b>[</b>	Editor.Region.Start
Select end of region	<b>Region</b> - <b>]</b>	<b>Esc</b> - <b>]</b>	Editor.Region.Finish
Unselect a region	<b>Region</b> - <b>X</b>		Editor.Region.Off
<b>Moving and Copying Text</b>			
Copy a selected item	<b>Region</b> - <b>C</b>		Editor.Region.Copy
Move a selected item	<b>Region</b> - <b>M</b>		Editor.Region.Move
Duplicate a single line	<b>Line</b> - <b>C</b>	<b>Esc</b> - <b>Control</b> <b>C</b>	Editor.Line.Copy
<b>Deleting Text</b>			
Delete character — forward	<b>Control</b> <b>D</b>		Editor.Char.Delete_Forward
Delete character — backward	<b>Delete</b>		Editor.Char.Delete_Backward
Reduce multiple blanks to one	<b>Control</b> <b>Delete</b>		Editor.Char.Delete_Spaces
Delete word	<b>Word</b> - <b>D</b>	<b>Esc</b> - <b>D</b>	Editor.Word.Delete
Delete to end of word	<b>Word</b> - <b>K</b>	<b>Esc</b> - <b>K</b>	Editor.Word.Delete_Forward
Delete to beginning of word	<b>Word</b> - <b>Delete</b>	<b>Esc</b> - <b>Delete</b>	Editor.Word.Delete_Backward
Delete line	<b>Line</b> - <b>D</b>	<b>Esc</b> - <b>Control</b> <b>D</b>	Editor.Line.Delete
Delete to end of line	<b>Line</b> - <b>K</b>	<b>Control</b> <b>K</b>	Editor.Line.Delete_Forward
Delete to beginning of line	<b>Line</b> - <b>Delete</b>	<b>Esc</b> - <b>Control</b> <b>F</b>	Editor.Line.Delete_Backward
Delete selected item	<b>Region</b> - <b>D</b> , <b>Region</b> - <b>K</b>		Editor.Region.Delete
<b>Searching and Replacing Text</b>			
Search for next occurrence	<b>Control</b> <b>F</b>		Editor.Search.Next
Search for previous occurrence	<b>Control</b> <b>R</b>		Editor.Search.Previous
Replace next occurrence	<b>Esc</b> - <b>F</b>		Editor.Search.Replace_Next
Replace previous occurrence	<b>Esc</b> - <b>R</b>		Editor.Search.Replace_Previous

**General Editing Operations (Continued)**

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
<b>Entering Text</b>			
Quote a special character	<b>Esc</b> - <b>'</b>		<b>Editor.Char.Quote</b>
Split line, cursor on new line	<b>Line</b> - <b>I</b>		<b>Editor.Line.Insert</b>
Split line, cursor on old line	<b>Line</b> - <b>O</b>	<b>Control</b> <b>O</b>	<b>Editor.Line.Open</b>
Join 2 lines	<b>Line</b> - <b>J</b>	<b>Esc</b> - <b>O</b> , <b>Esc</b> - <b>Control</b> <b>O</b>	<b>Editor.Line.Join</b>
Enter text in insert mode	<b>Image</b> - <b>I</b>		<b>Editor.Set.Insert_Mode(True)</b>
Enter text in overwrite mode	<b>Image</b> - <b>O</b>		<b>Editor.Set.Insert_Mode(False)</b>
Show current line number	<b>Line</b> - <b>?</b>		<b>What.Line</b>
<b>Transposing Text</b>			
Transpose with previous character	<b>Control</b> <b>T</b>		<b>Editor.Char.Transpose</b>
Transpose with previous word	<b>Word</b> - <b>T</b>	<b>Esc</b> - <b>T</b>	<b>Editor.Word.Transpose</b>
Transpose with previous line	<b>Line</b> - <b>T</b>	<b>Esc</b> - <b>Control</b> <b>T</b>	<b>Editor.Line.Transpose</b>
<b>Controlling Case</b>			
Capitalize to end of word	<b>Word</b> - <b>^</b>	<b>Esc</b> - <b>^</b>	<b>Editor.Word.Capitalize</b>
Capitalize words to end of line	<b>Line</b> - <b>^</b>		<b>Editor.Line.Capitalize</b>
Capitalize every word in region	<b>Region</b> - <b>^</b>		<b>Editor.Region.Capitalize</b>
Make lowercase to end of word	<b>Word</b> - <b>&lt;</b>	<b>Esc</b> - <b>&lt;</b>	<b>Editor.Word.Lower_Case</b>
Make lowercase to end of line	<b>Line</b> - <b>&lt;</b>		<b>Editor.Line.Lower_Case</b>
Convert entire region to lowercase	<b>Region</b> - <b>&lt;</b>		<b>Editor.Region.Lower_Case</b>
Make uppercase to end of word	<b>Word</b> - <b>&gt;</b>	<b>Esc</b> - <b>&gt;</b>	<b>Editor.Word.Upper_Case</b>
Make uppercase to end of line	<b>Line</b> - <b>&gt;</b>		<b>Editor.Line.Upper_Case</b>
Convert entire region to uppercase	<b>Region</b> - <b>&gt;</b>		<b>Editor.Region.Upper_Case</b>

**General Editing Operations (Continued)**

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
<b>Holding and Retrieving Text</b>			
Hold selected text	<b>Region</b> - <b>I</b>	<b>Control</b> - <b>C</b>	<b>Editor.Hold_Stack.Push</b>
Retrieve most recently held text	<b>Region</b> - <b>f</b>	<b>Control</b> - <b>Y</b>	<b>Editor.Hold_Stack.Top</b>
Retrieve previous held text	<b>Region</b> - <b>-</b>		<b>Editor.Hold_Stack.Previous</b>
Retrieve next held text	<b>Region</b> - <b>-</b>	<b>Esc</b> - <b>C</b> , <b>Esc</b> - <b>Y</b>	<b>Editor.Hold_Stack.Next</b>
<b>Formatting Text</b>			
Center the line cursor is on	<b>Line</b> - <b>\$</b>		<b>Editor.Line.Center</b>
Fill text in selected region	<b>Region</b> - <b>Format</b>		<b>Editor.Region.Fill</b>
Justify text in selected region	<b>Region</b> - <b>Complete</b>		<b>Editor.Region.Justify</b>
Automatically wrap lines	<b>Image</b> - <b>F</b>		<b>Editor.Set.Fill_Mode(True)</b>
Do not wrap lines	<b>Image</b> - <b>X</b>		<b>Editor.Set.Fill_Mode(False)</b>

### Writing Text Files

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
<b>Accessing Text Files</b>			
Create a new text file Display existing text file Open text file for editing Revert to last saved version	<div>Create Text</div> <div>Definition</div> <div>Edit</div> <div>Object . L</div>		Text.Create Common.Definition Common.Edit Common.Revert
<b>Saving Changes</b>			
Save, leaving open for editing Save, making read only	<div>Enter</div> <div>Promote</div>		Common.Commit Common.Promote
<b>Terminating Edit</b>			
Remove image, discarding changes Remove image, saving changes	<div>Object . G</div> <div>Object . X</div>		Common.Abandon Common.Release
<b>Selecting Substructures within Text</b>			
Select current word Select current sentence Select current paragraph Select smaller structure Select previous structure, same level Select next structure, same level Turn off selection	<div>Object . --</div> <div>numeric 2 . Object . --</div> <div>numeric 3 . Object . --</div> <div>Object . --</div> <div>Object . *</div> <div>Object . i</div> <div>Control X</div>		Common.Object.Parent Common.Object.Parent Common.Object.Parent Common.Object.Child Common.Object.Previous Common.Object.Next Editor.Set.Designation_Off

## Writing Ada Programs

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
<b>Creating Ada Programs</b>			
Create an Ada unit in library Build a body Build a private part Put temporary name in library	Object - I Create Body Create Private -		Common.Object.Insert Ada.Create_Body Ada.Create_Private Ada.Install_Stub
<b>Accessing Ada Programs</b>			
Display Ada unit, read only Demote to source, open for editing	Definition Edit		Common.Definition Common.Edit
<b>Saving Changes and Terminating Edit</b>			
Save, leaving open for editing Release image, discarding changes Release image, saving changes Revert to last version	Enter Object - G Object - X Object - L		Common.Commit Common.Abandon Common.Release Common.Revert
<b>Checking for Errors</b>			
Complete and check syntax Check for semantic errors Explain underlined error Move to next underlined error Move to previous underlined error Remove underline from error Clear all underlined errors Redisplay cleared errors	Format Semanticise Object - ? Esc - N Esc - U Control X Underlines Off Show Errors		Common.Format Common.Semanticize Common.Explain Editor.Cursor.Next Editor.Cursor.Previous Editor.Set.Designation_Off Common.Clear_Underlining Ada.Get_Errors

**Writing Ada Programs (Continued)**

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
<b>Changing the Compilation State</b>			
Change unit to source state from any state Change unit to installed state from any state Change unit to coded state from any state	Source Unit Install Unit Code Unit		Ada.Source_Unit Ada.Install_Unit Ada.Code_Unit
<b>Changing to a Higher Compilation State</b>			
Promote unit to next higher state Code unit and those it depends on In this world only Across worlds Install unit and those it depends on In this world only	Promote  Code (This World) Code (All Worlds)  Install (This World)		Common.Promote  Compilation.Make Compilation.Make  Compilation.Promote
<b>Changing to a Lower Compilation State</b>			
Demote unit to next lower state Demote unit and dependents to source In this world only	Demote  Source (This World)		Common.Demote  Compilation.Demote
<b>Selecting Structures within Ada Programs</b>			
Select successively larger structures Select successively smaller structures Select previous structure, same level Select next structure, same level Select first structure Select last structure Turn off selection cursor is in	Object . -- Object . -- Object . ↑ Object . ↓ Object . Begin Of Object . End Of Control X	Object . B Object . E	Common.Object.Parent Common.Object.Child Common.Object.Previous Common.Object.Next Common.Object.First_Child Common.Object.Last_Child Editor.Set.Designation_Off



**Writing Ada Programs (Continued)**

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
<b>Modifying Ada Programs</b>			
Edit selected Ada structure Insert Ada structures(s) in program Delete selected Ada structure  Copy selected Ada structure Move selected Ada structure Withdraw Ada unit stub	<div>Edit</div> <div>Object - I</div> <div>Object - D,</div> <div>Object - K</div> <div>Object - C</div> <div>Object - M</div> <div>Withdraw Unit</div>		Common.Edit Common.Object.Insert Common.Object.Delete  Common.Object.Copy Common.Object.Move Ada.Withdraw
<b>Entering Comments and Special Strings</b>			
Comment selected item or region Uncomment selected item or region Tab forward to comment	- - -		Region.Comment Region.Uncomment Editor.Char.Tab-To-Comment
<b>Browsing Ada Programs</b>			
Display other part of Ada unit Display other part, same window Display Ada unit cursor is on Display parent object Set mark at current location Cycle through marks in stack Cycle back through marks in stack Return to most recent mark	<div>Other Part</div> <div>Other Part In Place</div> <div>Definition</div> <div>Enclosing</div> <div>Mark - .</div> <div>Mark - -</div> <div>Mark - -</div> <div>Mark - ↑</div>	<div>Esc - M</div>	Ada.Other_Part Ada.Other_Part Common.Definition Common.Enclosing Editor.Mark.Push Editor.Mark.Next Editor.Mark.Previous Editor.Mark.Top
<b>Checking Using Occurrences</b>			
Show uses of selected identifier In this unit only In any unit Show unused declarations In this unit only Check other units	<div>Show Usage (Unit)</div> <div>Show Usage</div> <div>Show Unused (Unit)</div> <div>Show Unused</div>		Ada.Show_Usage Ada.Show_Usage  Ada.Show_Unused Ada.Show_Unused

### Debugging Ada Programs

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
Execute program with Debugger on Display Debugger window Show current statement in source	<div>Esc - Promote</div> <div>Debugger Window</div> <div>Show Source</div>		Command.Debug Debug.Current-Debugger Debug.Source
<b>Stepping and Executing</b>			
Continue program execution Step one statement Step one statement at same level Stop task execution Display information about tasks Display task rendezvous info	<div>Execute</div> <div>Run</div> <div>Run Local</div> <div>Stop</div> <div>Task Display</div> <div>-</div>		Debug.Execute Debug.Run Debug.Run (Local) Debug.Stop Debug.Task-Display Debug.Information
<b>Setting and Removing Breakpoints</b>			
Set breakpoints with default lifetime Display breakpoints Reactivate existing breakpoints Remove breakpoints	<div>Break</div> <div>Show Breaks</div> <div>Activate</div> <div>Remove Breaks</div>		Debug.Break Debug.Show Debug.Activate Debug.Remove
<b>Viewing Stacks</b>			
Display calling stack	<div>Stack</div>		Debug.Stack
<b>Displaying and Modifying Variables</b>			
Display values of selected variables Modify value of selected variable	<div>Put</div> <div>Modify</div>		Debug.Put Debug.Modify
<b>Handling Exceptions</b>			
Stop execution when exception raised Do not stop when exception raised Remove handling for this exception	<div>Catch</div> <div>Propagate</div> <div>-</div>		Debug.Catch Debug.Propagate Debug.Forget

## Managing Libraries

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
<b>Creating Libraries</b>			
Create a directory	Create Directory		Library.Create_Directory
Create a world	Create World		Library.Create_World
<b>Manipulating Objects in Libraries</b>			
Create an Ada unit in library	Object - I		Common.Object.Insert
Create a text file in library	Create Text		Text.Create
Delete selected object from library	Object - D		Common.Object.Delete
Undelete selected object from library	Object - K		Common.Object.Undo
Print selected object	Object - U		Queue.Print
Show access list for designated object	Print		Access_List.Display
	Show Access List		
<b>Controlling Library Display</b>			
Toggle information on library objects	Object - ?		Common.Explain
Show more detail	Object - F		Common.Expand
Show less detail	Object - .		Common.Elide

## Using CMVC

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
Check out designated object	-		Cmvc.Check-Out
Check in designated object	-		Cmvc.Check-In
Accept changes for designated object	-		Cmvc.Accept_Changes
Show objects that are checked out			
In this view	-		Cmvc.Show_Checked-Out_In_View
By you, any view	-		Cmvc.Show_Checked-Out_By_User
Show info about designated object	-		Cmvc.Show
Show out-of-date objects in this view	-		Cmvc.Show_Out-Of-Date_Objects

## Managing Links

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
<b>Accessing Links</b>			
<b>List links</b> <b>Edit links display</b> <b>Refresh link image</b>	- - Object - L		<b>Links.Display</b> <b>Links.Edit</b> <b>Common.Revert</b>
<b>Removing the Link Editor</b>			
<b>Remove window temporarily</b> <b>Release image permanently</b>	Window - D Object - X		<b>Editor.Window.Delete</b> <b>Common.Release</b>
<b>Selecting Links</b>			
<b>Select link cursor is on</b> <b>Select all links</b> <b>Select previous link</b> <b>Select next link</b> <b>Select first link in image</b> <b>Select last link in image</b>	Object - - Object - - Object - ↑ Object - ↓ Object - Begin Of Object - End Of	Object - B Object - E	<b>Common.Object.Child</b> <b>Common.Object.Parent</b> <b>Common.Object.Previous</b> <b>Common.Object.Next</b> <b>Common.Object.First_Child</b> <b>Common.Object.Last_Child</b>
<b>Modifying Links</b>			
<b>Add a new link—simple method</b> <b>Add a new link</b> <b>Give selected link another source</b> <b>Delete selected link</b>	- Object - I Edit Object - D, Object - K		<b>Links.Add</b> <b>Common.Object.Insert</b> <b>Common.Edit</b> <b>Common.Object.Delete</b>
<b>Traversing Linked Ada Units</b>			
<b>Go to source unit of current link</b> <b>Go to world associated with links</b> <b>List Ada units that use current link</b>	Definition Enclosing Object - ?		<b>Common.Definition</b> <b>Common.Enclosing</b> <b>Common.Explain</b>
<b>Controlling the Display</b>			
<b>Toggle order of kind of link</b> <b>Toggle classes of source of link</b>	Object - ! Object - -		<b>Common.Expand</b> <b>Common.Elide</b>

## Managing Searchlists

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
<b>Accessing the Searchlist</b>			
Edit or view searchlist Refresh searchlist image	- Object . L		Search_List.Edit Common.Revert
<b>Removing the Searchlist Editor</b>			
Remove window temporarily Release image permanently	Window . D Object . X		Editor.Window.Delete Common.Release
<b>Selecting Entries</b>			
Select entry cursor is on Select all entries Select next entry Select previous entry Select first entry on list Select last entry on list Go to world named by current entry	Object . -- Object . -- Object . I Object . ? Object . Begin Of Object . End Of Definition	Object . B Object . E	Common.Object.Child Common.Object.Parent Common.Object.Next Common.Object.Previous Common.Object.First_Child Common.Object.Last_Child Common.Definition
<b>Modifying the Searchlist</b>			
Add a new entry Delete selected entry Move selected entry	Object . I Object . D Object . K Object . M		Common.Object.Insert Common.Object.Delete Common.Object.Move

## Using Keyboard Macros

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
Start macro definition	Mark - Begin Of	Mark - [	Editor.Macro.Start
End macro definition	Mark - End Of	Mark - ]	Editor.Macro.Finish
Execute macro	Mark - Promote	Esc - X	Editor.Macro.Execute
Bind macro to key	Mark - Definition		Editor.Macro.Bind

## Using Environment I/O Resources

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
Indicate end of input to program	numeric .		Text.End.Of.Input
Commit interactive input	Promote	Enter	Common.Commit

## Managing Jobs

<i>Description</i>	<i>Basic Keys</i>	<i>Accelerated Keys</i>	<i>Command</i>
Disconnect job from terminal	Control G		Job.Interrupt
Kill job	Job Kill	Esc - G	Job.Kill(0)
Stop running jobs	Job Disable		Job.Disable(0)
Resume stopped jobs	Job Enable		Job.Enable(0)
Reconnect job	Job Connect		Job.Connect(0)

## Master Reference to Key Bindings by Command

Legend		
=====		
C = <b>Control</b>	X1 = <b>Object</b>	
ESC = <b>Esc</b>	X2 = <b>Region</b>	
ESC_C = <b>Esc</b> - <b>Control</b>	X3 = <b>Window</b>	
S = <b>Shift</b>	PF1 = <b>Image</b>	
X4 = <b>Promote</b>	PF2 = <b>Line</b>	
X5 = <b>Complete</b>	PF3 = <b>Word</b>	
X6 = <b>Format</b>	PF4 = <b>Mark</b>	
Command		
=====		
Access_List.Display	ESC_C_F5	
Access_List.Display ( For_Object...		
Ada.Code_Unit	S_F6	OMS_F20
Ada.Create_Body	S_F8	C_F13
Ada.Create_Private	ESC_C_F9	S_F15
Ada.Delete_Blank_Line		OMS_F15
		OMS_K
Ada.Get_Errors	S_F10	CM_K
Ada.Insert_Blank_Line		S_F16
		OMS_I
		CM_I
Ada.Install_Stub		S_F13
Ada.Install_Unit	F6	F13
Ada.Make_Inline		OMS_F17
Ada.Make_Separate		OMS_F18
Ada.Other_Part ( Name => "<Image...	F9	C_F10
Ada.Other_Part ( Name => "<Image...	S_F9	CS_F10
Ada.Show_Used ( In_Unit => "<I...	C_F10	M_F17
Ada.Show_Used ( In_Unit => "<I...	ESC_C_F10	CM_F17
Ada.Show_Usage ( Name => "<Curso...	ESC_F9	M_F16
Ada.Show_Usage ( Name => "<Curso...	C_F9	CM_F16
Ada.Show_Usage ( Name => "<Curso...		OMS_F16
Ada.Source_Unit	C_F7	C_F14
Ada.Withdraw	ESC_C_F7	M_F14
Cmvc.Accept_Changes ( Destination...		CM_F12
Cmvc.Check_In ( What_Object => "...		M_F12
Cmvc.Check_Out ( What_Object => "...		C_F12
Cmvc.Show ( Objects => "<Cursor>...		CS_F12
Cmvc.Show_Checked_Out_By_User ( ...		MS_F12
Cmvc.Show_Checked_Out_In_View ( ...		OMS_F12
Cmvc.Show_Out_Of_Date_Objects ( ...		S_F12
Command.Debug	ESC_C_M	M_PROMOT
		MS_CARRIAGE_RETURN
		M_CARRIAGE_RETURN
		MS_ENTER
		M_ENTER
Command.Spawn	S_X4	C_PROMOT
Common.Abandon	X1.'g'	OBJECT.'g'
	X1.'G'	OBJECT.'g'

Common.Clear_Underlining	ESC_F10	C_F16
Common.Commit	ENTER	S_ENTER
		C_CARRIAGE_RETURN
		ENTER
		CS_CARRIAGE_RETURN
Common.Complete	X5	COMPLT
Common.Create_Command	F8	F15
Common.Definition	X1.F4	OBJECT.F10
		CM_RIGHT
		S_RIGHT
		F10
Common.Definition ( In_Place => ...		
Common.Definition ( Name => "<Cu...	F4	
Common.Definition ( Name => "<Cu...	S_F4	
Common.Definition ( Name => "<Cu...		S_F10
Common.Demote	S_F7	S_F14
Common.Edit	F7	F14
Common.Elide	X1.'.'	OBJECT.'.'
		OBJECT.'>'
Common.Enclosing	C_F4	M_F10
		CM_LEFT
Common.Enclosing ( In_Place => F...	ESC_F4	
Common.Enclosing ( In_Place => T...		S_LEFT
Common.Enclosing ( In_Place => T...	ESC_C_F4	MS_F10
Common.Enclosing ( In_Place => T...		OMS_F10
Common.Expand	X1.'1'	C_EXCLAM
	X1.'I'	OBJECT.'I'
		C_I
		OBJECT.'1'
		OBJECT.'?'
Common.Explain	ESC_SLASH	F17
	ESC_PLUS	C_QUERY
	X1.'/'	OBJECT.'/'
	X1.'+'	C_SLASH
	ESC_QUERY	
	X1.'?'	
Common.Format	X6	FORMAT
Common.Object.Child	X1.RIGHT	OBJECT.RIGHT
		C_RIGHT
Common.Object.Copy	X1.'c'	OBJECT.'c'
	X1.'C'	OBJECT.'C'
	X1.'k'	OBJECT.'d'
Common.Object.Delete	X1.'D'	OBJECT.'D'
	X1.'k'	OBJECT.'k'
	X1.'d'	OBJECT.'K'
Common.Object.First_Child	X1.'B'	OBJECT.BEGIN_OF
	X1.'b'	C_BEGIN_OF
	X1.BEGIN_OF	
Common.Object.Insert	X1.'I'	OBJECT.'I'
	X1.'I'	C_F15
		OBJECT.'I'
		C_END_OF
Common.Object.Last_Child	X1.'e'	OBJECT.END_OF
	X1.END_OF	
	X1.'E'	
Common.Object.Move	X1.'M'	OBJECT.'M'
	X1.'m'	OBJECT.'m'
Common.Object.Next	X1.DOWN	C_DOWN
		OBJECT.DOWN
Common.Object.Parent	X1.LEFT	OBJECT.LEFT
		C_LEFT
Common.Object.Previous	X1.UP	OBJECT.UP
		C_UP
Common.Promote	X4	PROMOT

Common.Redo	X1.'R'	OBJECT.'r'
	X1.'r'	OBJECT.'R'
Common.Release	X1.'X'	OBJECT.'x'
	X1.'x'	OBJECT.'X'
Common.Revert	X1.'L'	OBJECT.'l'
	X1.'l'	OBJECT.'L'
Common.Semanticize	F10	F16
Common.Sort_Image		OBJECT.'S'
		OBJECT.'s'
Common.Undo	X1.'u'	OBJECT.'U'
	X1.'U'	OBJECT.'u'
Compilation.Demote ( Unit => "<S...		MS_F14
Compilation.Demote ( Unit => "<S...		CS_F14
Compilation.Demote ( Unit => "<S...		CMS_F14
Compilation.Demote ( Unit => "<S... ESC_F7		CM_F14
Compilation.Make ( Unit => "<Ima... C_F6		CM_F13
Compilation.Make ( Unit => "<Ima... ESC_C_F6		CMS_F13
Compilation.Promote ( Unit => "<...		MS_F13
Compilation.Promote ( Unit => "<... ESC_F6		M_F13
Debug.Activate ( Breakpoint => 0)	ESC_F2	M_F7
Debug.Break	S_F2	C_F7
Debug.Break ( Default_Lifetime =...		S_F7
Debug.Catch	S_F3	C_F8
Debug.Current_Debugger ( "")		CMS_F9
Debug.Execute	S_F1	S_F6
Debug.Forget		S_F8
Debug.Information ( Debug.Rendez...		CMS_F8
Debug.Modify ( New_Value => "", ... ESC_C_F3		CM_F9
Debug.Propagate	ESC_F3	M_F8
Debug.Put	F3	F9
Debug.Remove ( Breakpoint => 0)	C_F2	CM_F7
Debug.Run	F1	F6
Debug.Run ( Debug.Returned)		M_F6
Debug.Run ( Stop_At => Debug.Loc... ESC_F1		C_F6
Debug.Set_Value ( Variable => De...		C_F9
Debug.Set_Value ( Variable => De...		S_F9
Debug.Set_Value ( Variable => De...		M_F9
Debug.Show	ESC_C_F2	CMS_F7
Debug.Show ( Debug.Exceptions)		CM_F8
Debug.Source	F2	
Debug.Source ( Location => "", S...		F7
Debug.Stack	C_F3	F8
Debug.Stop	C_F1	
Debug.Stop ( Name => "")		CM_F6
Debug.Task_Display	ESC_C_F1	CMS_F6
Editor.Char.Capitalize		C_6
		C_CIRCUMFLEX
Editor.Char.Delete_Backward	DELETE	DELETE
Editor.Char.Delete_Forward	C_D	C_D
		CS_D
Editor.Char.Delete_Spaces	ESC_BACKSLASH	C_DELETE
		CS_DELETE
Editor.Char.Insert_Character ( 1...		MS_SPACE
		CS_SPACE
		CM_SPACE
		M_SPACE
		CMS_SPACE
		C_SPACE
		S_SPACE
Editor.Char.Insert_String ( """)		C_RIGHT_PAREN

Editor.Char.Insert_String ( """)		C_0
		C_9
Editor.Char.Insert_String ( ":=")		C_LEFT_PAREN
		C_COLON
Editor.Char.Insert_String ( "=>")		C_SEMICOLON
		C_EQUAL
		C_PLUS
Editor.Char.Lower_Case		C_LESS_THAN
		C_COMMA
Editor.Char.Quote	ESC_TICK	C_TICK
	ESC_STAR	C_QUOTE
Editor.Char.Tab_Backward	ESC_C_I	CS_TICK
		CS_TAB
Editor.Char.Tab_Forward	C_I	C_TAB
		S_TAB
Editor.Char.Tab_To_Comment		TAB
		MS_TAB
		OBJECT.TAB
		M_TAB
Editor.Char.Transpose	C_T	C_T
		CS_T
Editor.Char.Upper_Case		C_PERIOD
		C_GREATER_THAN
Editor.Cursor.Backward		CS_B
		C_B
Editor.Cursor.Down	C_N	C_N
	DOWN	LINE.DOWN
		CS_N
		DOWN
Editor.Cursor.Down ( 8)	ESC_C_N	CM_N
		CMS_N
Editor.Cursor.Forward		C_F
		CS_F
Editor.Cursor.Left	C_H	CS_H
	LEFT	C_H
		LEFT
Editor.Cursor.Left ( 8)	ESC_C_H	CM_H
		CMS_H
Editor.Cursor.Next	ESC_S_N	M_DOWN
	ESC_N	
Editor.Cursor.Next ( Prompt => F...		S_F18
Editor.Cursor.Next ( Prompt => T...		M_F18
Editor.Cursor.Next ( Prompt => T...		F18
		M_N
		MS_N
Editor.Cursor.Previous	ESC_U	M_U
	ESC_S_U	MS_U
		M_UP
Editor.Cursor.Previous ( Prompt ...		CS_F18
Editor.Cursor.Previous ( Prompt ...		CM_F18
Editor.Cursor.Previous ( Prompt ...		C_F18
Editor.Cursor.Right	C_J	C_J
	RIGHT	CS_J
		RIGHT
Editor.Cursor.Right ( 8)	ESC_C_J	CMS_F
		CM_F
		CMS_J
		CM_J
Editor.Cursor.Up	C_U	C_U
	UP	CS_U



Editor.Cursor.Up ( 8)	ESC_C_U	UP LINE.UP CMS_U CM_U REGION.'p' REGION.'P' REGION.DELETE M_Y MS_Y M_C REGION.RIGHT MS_C REGION.LEFT C_C REGION.DOWN CS_C REGION.'r' REGION.'R' REGION.'T' REGION.'t' REGION.UP CS_Y C_Y S_BEGIN_OF IMAGE.BEGIN_OF
Editor.Hold_Stack.Copy_Top		
Editor.Hold_Stack.Delete_Top		
Editor.Hold_Stack.Next	ESC_Y ESC_C X2.RIGHT ESC_S_Y ESC_S_C	
Editor.Hold_Stack.Previous	X2.LEFT	
Editor.Hold_Stack.Push	C_C X2.DOWN	
Editor.Hold_Stack.Rotate		
Editor.Hold_Stack.Swap		
Editor.Hold_Stack.Top	C_Y X2.UP	
Editor.Image.Beginning_Of	PF1.'b' PF1.BEGIN_OF PF1.'B'	
Editor.Image.Down	C_V PF1.DOWN	
Editor.Image.End_Of	PF1.END_OF PF1.'E' PF1.'e' PF1.TAB	
Editor.Image.Find ( "" )		IMAGE.'?' IMAGE.'/' C_F11
Editor.Image.Find ( "Help Window" )	ESC_F5	
Editor.Image.Find ( Name => "Nam..." )	PF1.'.' PF1.'/' PF1.'?' PF1.LEFT	
Editor.Image.Left		MS_LEFT IMAGE.LEFT MS_RIGHT IMAGE.RIGHT IMAGE.UP CS_Z S_UP C_Z C_Q CS_Q M_F11
Editor.Image.Right	PF1.RIGHT	
Editor.Image.Up	PF1.UP C_Z	
Editor.Key.Name	C_F5 ESC_AT_SIGN ESC_QUOTATION ESC_Z ESC_S_Q ESC_Q	
Editor.Key.Prompt		M_Q MS_Q F12 CMS_B CM_B CM_A CS_A C_A
Editor.Key.Prompt ( Key_Code => "" )		
Editor.Line.Beginning_Of	ESC_C_B PF2.'b' C_B C_A ESC_C_A	

	PF2.'B' PF2.BEGIN_OF	LINE.BEGIN_OF CMS_A BEGIN_OF CM_CIRCUMFLEX CM_6 LINE.'.' LINE.'6' LINE.'@' LINE.'4' LINE.'C' CM_C LINE.'c' CMS_C CM_D CMS_D LINE.'D' LINE.'d' LINE.DELETE CM_DELETE CMS_DELETE CS_X LINE.'K' LINE.'k' C_K CM_E CS_E CMS_E END_OF C_E LINE.END_OF CARRIAGE_RETURN S_CARRIAGE_RETURN LINE.'I' LINE.'I' C_I CS_I LINE.'J' M_O CM_O MS_O CMS_O LINE.'j' LINE.'J' LINE.'<' LINE.'<' CM_LESS_THAN CM_COMMA C_O CS_O LINE.'o' LINE.'O' CM_T LINE.'t' LINE.'T' CMS_T CM_GREATER_THAN LINE.'.' CM_PERIOD LINE.'>' MARK.F10 M_X
Editor.Line.Capitalize	PF2.'.' PF2.'6' PF2.'''	
Editor.Line.Center	PF2.'4'	
Editor.Line.Copy	PF2.'@' ESC_C_C PF2.'c' PF2.'C'	
Editor.Line.Delete	ESC_C_D PF2.'D' PF2.'d'	
Editor.Line.Delete_Backward	ESC_C_F PF2.DELETE	
Editor.Line.Delete_Forward	PF2.'k' C_K ESC_C_K PF2.'K' PF2.END_OF PF2.TAB PF2.'e' PF2.'E' C_E ESC_C_E C_M PF2.'i' PF2.'I'	
Editor.Line.Indent		
Editor.Line.Insert		
Editor.Line.Insert ( - ( 1 ) )		
Editor.Line.Join	PF2.'J' ESC_C_O ESC_S_O ESC_O PF2.'j'	
Editor.Line.Lower_Case	PF2.'<'	
Editor.Line.Open	PF2.'o' C_O PF2.'O'	
Editor.Line.Transpose	ESC_C_T PF2.'T' PF2.'t'	
Editor.Line.Upper_Case	PF2.'>'	
Editor.Macro.Bind	PF4.F4	
Editor.Macro.Explode	PF4.ENTER	

	PF4.X4 ESC_S_X ESC_Y	MARK. ENTER MARK. PROMOT MARK. CARRIAGE_RETURN MS_X
Editor.Macro.Finish	PF4.'e' PF4.'E' PF4.'j' PF4.END_OF PF4.'j'	MARK.'j' M_RIGHT_BRACE MARK.'j' MARK.END_OF M_RIGHT_BRACKET
Editor.Macro.Start	PF4.BEGIN_OF PF4.'b' PF4.'{' PF4.'B' PF4.'{'	M_LEFT_BRACKET MARK.'{' M_LEFT_BRACE MARK.'{' MARK.BEGIN_OF
Editor.Mark.Copy_Top	PF4.'{'	MARK.'p' MARK.'P'
Editor.Mark.Delete_Top		MARK.DELETE
Editor.Mark.Next	ESC_M PF4.RIGHT ESC_S_M	MS_M M_M MARK.RIGHT
Editor.Mark.Previous	PF4.LEFT	MARK.LEFT
Editor.Mark.Push	NUL PF4.DOWN	MARK.DOWN CS_M C_M
Editor.Mark.Rotate		MARK.'R' MARK.'r'
Editor.Mark.Swap		MARK.'t' MARK.'T'
Editor.Mark.Top	PF4.UP	MARK.UP
Editor.Region.Beginning_Of	X2.'b' X2.'B' X2.BEGIN_OF X2.'6' X2.'-' X2.'--'	REGION.BEGIN_OF  REGION.'6' REGION.'--'
Editor.Region.Capitalize		REGION.'_' REGION.'-'
Editor.Region.Comment		REGION.'c' REGION.'C' REGION.'K' REGION.'k' REGION.'d' REGION.'D'
Editor.Region.Copy	X2.'c' X2.'C'	REGION.'d' REGION.END_OF
Editor.Region.Delete	X2.'D' X2.'d' X2.'K' X2.'k' X2.'E' X2.END_OF X2.'e'	
Editor.Region.End_Of	X2.'E' X2.END_OF X2.'e'	
Editor.Region.Fill	X2.X6	REGION.FORMAT
Editor.Region.Finish	X2.'}' X2.'J' X2.'j' ESC_RIGHT_BRACE	REGION.'}' C_RIGHT_BRACKET C_RIGHT_BRACE REGION.'}'
Editor.Region.Justify	X2.X5	REGION.COMPLT
Editor.Region.Lower_Case	X2.'<'	REGION.'<' REGION.'_' REGION.'M' REGION.'m' REGION.'x' REGION.'X'
Editor.Region.Move	X2.'m' X2.'M'	
Editor.Region.Off	X2.'x' X2.'X'	
Editor.Region.Start	ESC_LEFT_BRACE X2.'{'	C_LEFT_BRACKET REGION.'{'

	X2.'{'	REGION.'{' C_LEFT_BRACE REGION.'-' REGION.'+' REGION.'_' REGION.'>'
Editor.Region.Uncomment		M_L MS_L CS_DOWN CS_LEFT CMS_RIGHT CMS_LEFT CMS_DOWN CS_L C_L CS_RIGHT CMS_UP CMS_BEGIN_OF CS_UP C_S CS_S CS_R C_R M_S MS_S M_R MS_R
Editor.Region.Upper_Case	X2.'>'	
Editor.Screen.Clear	ESC_L ESC_S_L	
Editor.Screen.Down		
Editor.Screen.Left		
Editor.Screen.Next		
Editor.Screen.Previous		
Editor.Screen.Push		
Editor.Screen.Redraw	C_L	
Editor.Screen.Right		
Editor.Screen.Top		
Editor.Screen.Up		
Editor.Search.Next	C_F	
Editor.Search.Previous	C_R	
Editor.Search.Replace_Next	ESC_S_F ESC_F	
Editor.Search.Replace_Previous	ESC_R ESC_S_R	
Editor.Set.Argument_Digit ( 0 )	NUMERIC_0	M_NUMERIC_0 C_NUMERIC_0 NUMERIC_0 S_NUMERIC_0 M_NUMERIC_1 C_NUMERIC_1 NUMERIC_1 S_NUMERIC_1 NUMERIC_2 M_NUMERIC_2 S_NUMERIC_2 C_NUMERIC_2 M_NUMERIC_3 S_NUMERIC_3 NUMERIC_3 C_NUMERIC_4 M_NUMERIC_4 S_NUMERIC_4 C_NUMERIC_5 M_NUMERIC_5 S_NUMERIC_5 NUMERIC_5 C_NUMERIC_6 M_NUMERIC_6 NUMERIC_6 C_NUMERIC_7 NUMERIC_7 M_NUMERIC_7 S_NUMERIC_7 C_NUMERIC_8
Editor.Set.Argument_Digit ( 1 )	NUMERIC_1	
Editor.Set.Argument_Digit ( 2 )	NUMERIC_2	
Editor.Set.Argument_Digit ( 3 )	NUMERIC_3	
Editor.Set.Argument_Digit ( 4 )	NUMERIC_4	
Editor.Set.Argument_Digit ( 5 )	NUMERIC_5	
Editor.Set.Argument_Digit ( 6 )	NUMERIC_6	
Editor.Set.Argument_Digit ( 7 )	NUMERIC_7	
Editor.Set.Argument_Digit ( 8 )	NUMERIC_8	

		S_NUMERIC_8 M_NUMERIC_8 NUMERIC_8 M_NUMERIC_9 S_NUMERIC_9 NUMERIC_9 C_NUMERIC_9
Editor.Set.Argument_Digit ( 9)	NUMERIC_9	
Editor.Set.Argument_Minus	DASH	C_DASH DASH S_DASH M_DASH
Editor.Set.Argument_Prefix	NUMERIC_COMMA	NUMERIC_COMMA M_NUMERIC_COMMA C_NUMERIC_COMMA S_NUMERIC_COMMA
Editor.Set.Designation_Off	C_X	CS_X C_F17 C_X
Editor.Set.Fill_Mode ( False)	PF1.'X'	IMAGE.'X'
	PF1.'x'	IMAGE.'x'
Editor.Set.Fill_Mode ( True)	PF1.'f'	IMAGE.'f'
	PF1.'F'	IMAGE.'F'
Editor.Set.Insert_Mode ( False)	PF1.'O'	IMAGE.'o'
	PF1.'o'	IMAGE.'O'
Editor.Set.Insert_Mode ( True)	PF1.'I'	IMAGE.'i'
	PF1.'I'	IMAGE.'I'
Editor.Window.Beginning_Of	X3.'b'	WINDOW.BEGIN_OF
	X3.BEGIN_OF	CM_BEGIN_OF
	X3.'B'	
Editor.Window.Child	X3.RIGHT	WINDOW.RIGHT
Editor.Window.Copy	X3.'c'	WINDOW.'c'
	X3.'C'	WINDOW.'C'
Editor.Window.Delete	X3.'D'	WINDOW.'x'
	X3.'X'	WINDOW.'X'
	X3.'K'	WINDOW.'x'
	X3.'d'	WINDOW.'K'
	X3.'k'	WINDOW.'d'
	X3.'x'	WINDOW.'D'
Editor.Window.Demote	X3.F7	WINDOW.S_F14
	X3.S_F7	WINDOW.F14
Editor.Window.Directory	X3.'/'	WINDOW.F10
	X3.F4	WINDOW.'?'
	X3.'+'	WINDOW.'/'
	X3.'?'	
Editor.Window.End_Of	X3.'e'	CM_END_OF
	X3.END_OF	WINDOW.END_OF
	X3.'E'	
Editor.Window.Expand	X3.'I'	WINDOW.'I'
	X3.'I'	WINDOW.'I'
Editor.Window.Expand ( - ( 4))	X3.'.'	WINDOW.'>'
		WINDOW.'
Editor.Window.Focus	X3.X5	WINDOW.FORMAT
Editor.Window.Join ( - ( 1))	X3.DELETE	WINDOW.DELETE
Editor.Window.Join ( 1)	X3.'J'	WINDOW.'j'
	X3.'j'	WINDOW.'J'
Editor.Window.Next	ESC.V	CM_DOWN
	S_DOWN	WINDOW.DOWN
	X3.DOWN	MS_V
	ESC_S_V	M_V
Editor.Window.Parent	X3.LEFT	WINDOW.LEFT

Editor.Window.Previous	ESC_S_Z	WINDOW.UP
	ESC_Z	M_Z
	S_UP	MS_Z
	X3.UP	CM_UP
Editor.Window.Promote	X3.X4	WINDOW.PROMOT
Editor.Window.Transpose	X3.'t'	WINDOW.'T'
	X3.'T'	WINDOW.'t'
Editor.Word.Beginning_Of	PF3.BEGIN_OF	M_A
	ESC_A	WORD.BEGIN_OF
	PF3.'B'	MS_A
	PF3.'b'	M_BEGIN_OF
	ESC_S_B	MS_B
	ESC_S_A	M_B
	ESC_B	
Editor.Word.Capitalize	PF3.'6'	WORD.'6'
	PF3.'-'	M_6
	PF3.'~'	M_CIRCUMFLEX
	ESC_TILDE	WORD.'~'
	ESC_6	
	ESC_CIRCUMFLEX	
Editor.Word.Delete	PF3.'D'	MS_D
	ESC_S_D	M_D
	PF3.'d'	WORD.'d'
	ESC_D	WORD.'D'
Editor.Word.Delete_Backward	PF3.DELETE	M_DELETE
	ESC_DEL	WORD.DELETE
		MS_DELETE
Editor.Word.Delete_Forward	ESC_K	MS_K
	PF3.'k'	WORD.'k'
	ESC_S_K	M_K
	PF3.'K'	WORD.'K'
Editor.Word.End_Of	PF3.TAB	M_E
	ESC_S_I	WORD.END_OF
	PF3.'e'	MS_E
	ESC_E	M_END_OF
	PF3.'E'	
	PF3.END_OF	
Editor.Word.Lower_Case	PF3.'<'	M_LESS_THAN
	ESC_LESS_THAN	WORD.'<'
		M_COMMA
		WORD.'<','
		M_J
	PF3.RIGHT	M_RIGHT
	ESC_S_J	WORD.RIGHT
	ESC_J	MS_J
Editor.Word.Previous	PF3.LEFT	M_LEFT
	ESC_S_H	WORD.LEFT
	ESC_H	M_H
		MS_H
Editor.Word.Transpose	ESC_S_T	WORD.'e'
	PF3.'t'	WORD.'T'
	PF3.'T'	MS_T
	ESC_T	M_T
Editor.Word.Upper_Case	PF3.'>'	WORD.'>'
	ESC_GREATER_THAN	WORD.'>'
		M_GREATER_THAN
		M_PERIOD
Job.Connect ( 0)	ESC_F11	M_F19
Job.Disable ( 0)	ESC_C_F11	F19
Job.Enable ( 0)	S_F11	S_F19

Job.Interrupt	C_G	C_G
	C_G	CS_G
Job.Kill ( 0)	ESC_G	C_F19
	C_F11	M_G
	ESC_S_G	MS_G
	ESC_G	
Library.Create_Directory ( Name ...	ESC_F8	M_F15
Library.Create_World ( Name => "")	ESC_C_F8	CS_F15
Queue.Print	F11	CM_F11
Text.Create ( Image_Name => "")	C_F8	CM_F15
Text.End_Of_Input	DOT	DOT
		CM_F19
What.Does ( "")	S_F5	F11
What.Does ( "Help_On_Help")	F5	S_F11
What.Home_Library	ESC_UP	CM_F10
What.Line	PF2.'?'	LINE.'/'
	PF2.'+'	LINE.'?'
	PF2.'/'	
What.Load ( Verbose => True)	S_F12	S_F20
What.Locks ( Name => "<Image>")	ESC_C_F12	M_F20
What.Object	C_F12	CM_F20
What.Tabs		CM_TAB
		CM_TAB
What.Time	F12	F20

# RATIONAL

## READER'S COMMENTS

**Note:** This form is for documentation comments only. You can also submit problem reports and comments electronically by using the SIMS problem-reporting system. If you use SIMS to submit documentation comments, please indicate the manual name, book name, and page number.

Did you find this book understandable, usable, and well organized? Please comment and list any suggestions for improvement.

---

---

---

---

---

If you found errors in this book, please specify the error and the page number. If you prefer, attach a photocopy with the error marked.

---

---

---

---

Indicate any additions or changes you would like to see in the index.

---

---

---

---

How much experience have you had with the Rational Environment?

6 months or less \_\_\_\_\_ 1 year \_\_\_\_\_ 3 years or more \_\_\_\_\_

How much experience have you had with the Ada programming language?

6 months or less \_\_\_\_\_ 1 year \_\_\_\_\_ 3 years or more \_\_\_\_\_

Name (optional) \_\_\_\_\_ Date \_\_\_\_\_

Company \_\_\_\_\_

Address \_\_\_\_\_

City \_\_\_\_\_ State \_\_\_\_\_ ZIP Code \_\_\_\_\_

**Please return this form to:**

**Publications Department  
Rational  
1501 Salado Drive  
Mountain View, CA 94043**