WANG

VS

Symbolic Debugger Reference
Release 7 Series

VS

Symbolic Debugger Reference Release 7 Series

1st Edition — October 1988

715-1144

WANG

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 TEL. (508) 459-5000, TELEX 172108

CONTENTS

CONTENTS (continued)

CONTENTS (continued)

CONTENTS (continued)

FIGURES

TABLES

HOW TO USE THIS MANUAL

INTENDED AUDIENCE

This manual is written for VS programmers who need to use the VS Symbolic Debugger. Familiarity with the VS Operating System is assumed.

ORGANIZATION

This manual is organized in the following way:

- Chapter 1 provides an overview of the VS Symbolic Debugger. It describes the features and functions of the Debugger and the debugging process. It also compares the Debugger with the previous Debugger and describes its operating environment.

- Chapter 2 describes the Debugger user interface. It summarizes all the commands and describes the screen display, command invocation, PF keys, macro instructions, menus, and startup files.

- Chapter 3 describes how to use the EASY command to enable you to use the Debugger in a simplified mode that is PF-key driven and designed for the new or infrequent user.

- Chapter 4 describes how to manage the debugging environment. It describes cursor and text movement, creating and manipulating windows, displaying files, modifying text, and modifying PF key assignments.

- Chapter 5 describes various debugging activities, using many of the debugging functions.

- Chapter 6 describes debugging with shared subroutine libraries and MSMAPed files.

- Chapter 7 describes all of the Debugger commands in reference format. The commands are organized alphabetically in this chapter.
- Appendix A describes how to perform, with the new Debugger, the functions that you performed with the previous Debugger.
- Appendix B provides a list of Debugger terms with definitions.
- Appendix C describes the Debugger error messages.
- Appendix D lists the shortest abbreviation that can be substituted for the full command name for each Debugger command.

RELATED DOCUMENTS

This manual refers to the following manuals:

- *VS System User's Introduction* (715-0417)
- *VS System Security Reference* (715-1736)
- *VS Advanced Security Features Guide* (715-1268)
- *VS Principles of Operation* (715-0422)
- *VS Editor Reference* (715-1143)

CHAPTER 1
INTRODUCTION TO THE VS SYMBOLIC DEBUGGER

1.1 INTRODUCTION

Debugging is the process of locating and correcting programming errors in a program. The OS SYMBOLIC DEBUGGER (Debugger) provides a means of referencing elements of a program using the actual data names and control structures of the language in which the program is written. The Debugger allows you to reference variables by the names used in the program (symbolic names) instead of using addresses to search through memory.

The Debugger enables you to monitor the behavior of a program while it is running. In addition, you can set stopping mechanisms called traps, inspect and modify values of variables, and alter the flow of execution of the program.

Specifically, the Debugger enables you to perform the following tasks:

- Interrupt the execution of a program
- Inspect a program's code, data, and internal state
- Reference variables by their symbolic program names
- Change specified data values in program memory
- Modify a program's control flow

The Debugger can be used with any object program running on the VS. If the program has been produced by a compiler, which produced a program listing and symbolic information about the program, the Debugger provides full symbolic debugging support. Symbolic support means that the Debugger accepts references to program components by their symbolic names.

However, if symbolic information is not present, machine level debugging using offsets with commands instead of symbolic values is still available. An offset is a hexadecimal value that represents the number of bytes from a specified starting address.

Note: *In cases where confusion might arise, the current version of the Debugger is referred to as the new Debugger, and the previous version as the previous Debugger. Otherwise, the current version is referred to as the Debugger.*

1.2 HOW THE DEBUGGER WORKS

The Debugger functions as an operating system service on the VS. The Debugger does not run as an independent program on the system, but in association with a program for which you have explicitly requested debugging assistance. This program is called the subject program. When the Debugger runs, it always runs as a task subordinate to the subject program task. Figure 1-1 shows the how the Debugger interacts with the operating system:

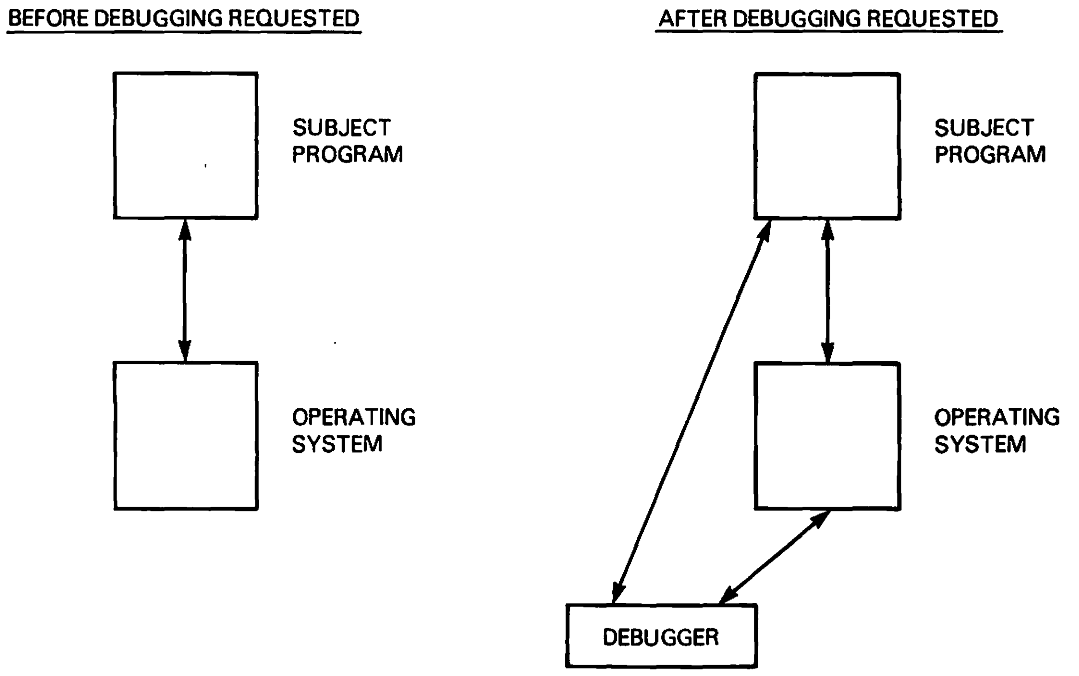


Figure 1-1. Interaction of the Debugger With the Operating System

1.2.1 Symbolic Information in the Program File

The OS translators optionally incorporate information in the program file during translation. This information enables symbolic or named examination of the program data elements.

When you compile a program, you have a choice of whether to include symbolic information. You include symbolic information by specifying YES for both the SOURCE option and the SYMBOLIC option on the Compiler or Assembler Options screen. If you choose to include the information, the OS programming languages place symbolic information into the Symbolic block of the program file during assembly or compilation. This information is used by the Debugger when it is invoked to debug at the symbolic level.

1.2.2 Program Listing Display

When you run the Debugger, it attempts to display the subject program listing. Any of the following conditions can prevent the listing from being displayed:

- You did not specify YES for the SYMBOLIC option for compilation or assembly (from the Compiler Options or the Assembler Options screen).

  Note: *Since the file name for the program listing is maintained in the subject program's Symbolic block, full symbolic debugging requires a YES response to both the SYMBOLIC option and the SOURCE option (from the Compiler Options screen or the Assembler Options screen). If you specify YES for the SYMBOLIC option and NO for the SOURCE option, partial symbolic debugging only is available; partial symbolic debugging allows you to examine variables in the subject program only.*

- The execution of the program was interrupted during the processing of a routine for which the source code is not available (such as a system external subroutine).
- The Debugger could not access the program listing.
- The Linker removed symbolic information from the program file.

1.3 DEBUGGER FEATURES

The Debugger functions as a powerful and flexible tool to help you locate and correct errors in your source program. A summary of the significant Debugger features follows:

Full symbolic support -- The Debugger provides full symbolic debugging support for all OS languages.

Full machine-level support -- The Debugger provides full machine-level support using addresses instead of symbolic names when symbolic information is not available, or when you are debugging an Assembler program.

Flexibility -- The Debugger allows you to debug at the symbolic level or at the machine level without changing modes. In addition, the PF keys and the startup file can be customized to meet individual needs.

Extensive command set -- The new Debugger provides a more extensive set of commands than the previous version, which makes debugging simpler and more efficient.

EASY command for simplified Debugger operation -- The EASY command allows you to run the Debugger in a simplified mode that is designed for the new or infrequent user. Refer to Chapter 3 for more details on using EASY.

1.4 DEBUGGER FUNCTIONS

The Debugger performs a number of functions that facilitate program debugging. A summary of the significant Debugger functions follows:

Control program execution -- The Debugger provides commands to suspend program execution, resume program execution, and resume program execution for a fixed number of statements before suspending again.

Set traps -- The Debugger enables you to set traps. Traps suspend program execution at specified locations when certain conditions are met. You can then use other debugging commands to examine or modify data.

Display various program information -- The Debugger enables you to display program information, such as variables and portions of memory, at certain stages of program execution to help reveal errors.

Modify program data -- The Debugger enables you to modify program values, excluding those values in protected memory.

Note: In the previous Debugger, it was possible to modify variables when executing in system code. As a result of enhanced security in OS Operating System Release 7.10, you must now be executing in user code in order to modify program variables.

If your program is interrupted while in system code, single step your program to the next executable statement. Then modify the desired values in memory using the ALTER command.

1.5 ACCESSING THE DEBUGGER

Only the operating system can create a Debugger task. The operating system does this in response to an explicit request to debug a subject program.

The Debugger can be accessed by the user or by a subprogram, at any of the following times:

Before program execution -- You can request the Debugger at the start of program execution if you run the program from the Command Processor menu. First, select the Run command (PF1) from the Command Processor menu. The system displays the Run screen, which prompts you to enter the file, library, and volume names of the program. Next, enter that information and press PF1. The system then invokes the Debugger for the subject program and displays the program listing starting at the initial entry point.

During program execution -- You can request the Debugger after you have interrupted program execution by pressing the system HELP key (the key marked HELP in the upper left corner of the keyboard). When you press the HELP key, the system displays the modified Command Processor menu. From that menu, you can access the Debugger by pressing PF10. The Debugger displays the program listing starting from the point at which execution was interrupted.

You can follow the same method if a program halts at some unscheduled point (e.g., cancel, program check, etc.).

Whenever the Debugger is requested, the operating system performs the following actions:

1. Pauses subject program execution

2. Creates a Debugger subtask for the subject program (if one does not already exist)

3. Passes control to the Debugger

At this point, the debugging session is in effect.

To provide the symbolic features of the Debugger for a program, you must satisfy the following conditions:

- The program must be compiled with YES specified for the SYMBOLIC option on the Compiler Options screen or the Assembler Options screen.

- The Linker option to retain symbolic debugging information from the program must be set to YES. The source listing for the program to be debugged must be available. A source listing is available if it meets all of the following conditions:

  - The source listing is in a file class that is accessible to the user.

  - The source listing is on a volume that is currently mounted.

  - The source listing is not in exclusive use by another user.

1.6 EXITING THE DEBUGGER

You can exit the Debugger by either of the following methods:

- Execute the CANCEL command. The Debugger then displays the Cancel screen, from which you can terminate the debugging session.

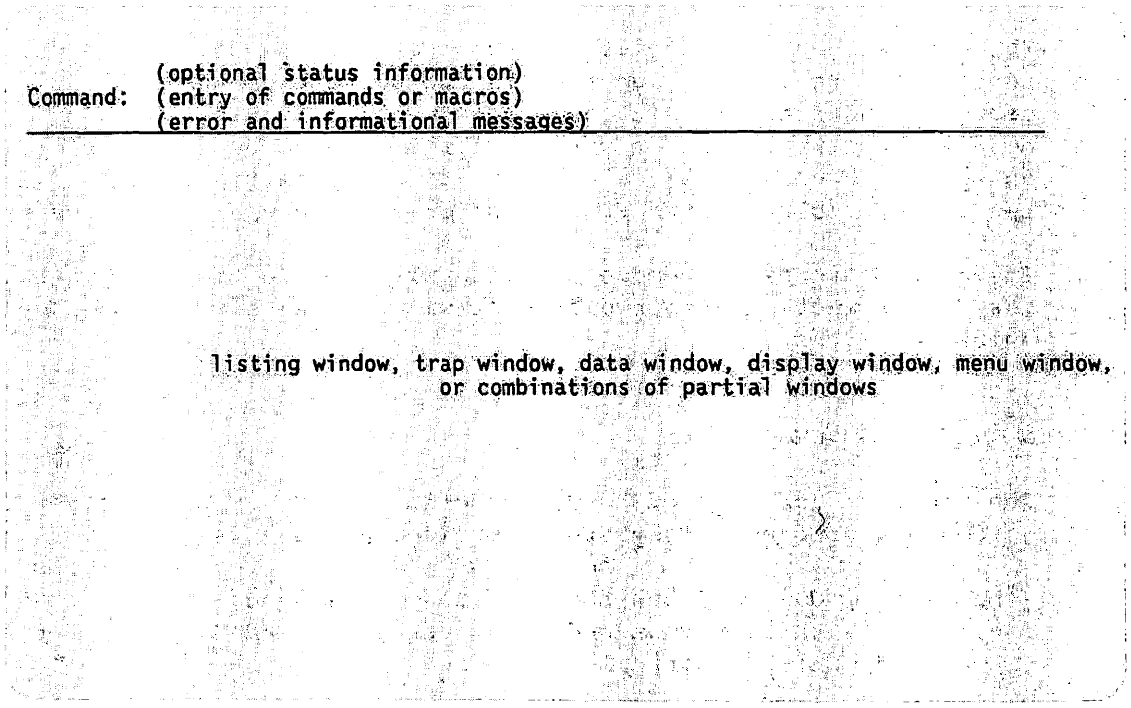
- Press the HELP key (the key marked HELP in the upper left corner of the keyboard). The Debugger then displays the modified Command Processor menu, from which you can terminate the debugging session.

1.7 OVERVIEW OF DEBUGGING

This section provides a general overview of debugging and introduces several key Debugger features.

1.7.1 The Debugger Workstation Screen

The Debugger interface operates from a screen called the Debugger Workstation screen. The structure of the Debugger Workstation screen is shown in Figure 1-2.



```
          (optional status information)
Command:  (entry of commands or macros)
          (error and informational messages)
________________________________________________________________

      listing window, trap window, data window, display window, menu window,
                    or combinations of partial windows
```

Figure 1-2. Structure of the Debugger Workstation Screen

The Debugger Workstation screen is divided into two sections: the Control section and the Window section. The Control section, which occupies the first three lines, contains the following items:

- A status line that displays information about the topmost window in the Window section.
- A command line that is the primary input mechanism to the Debugger. You enter commands on the command line.
- A message line where the Debugger displays error or informational messages.

The Window section occupies the remainder of the screen and contains a portion of the information that is the subject of the current debugging action. Within the Window section, you can display one or more windows that contain various types of program information. For example, you can display a portion of the subject program on which to set traps, and a menu of PF key assignments to use to set the traps.

If you want to examine more than one window at a time, you can divide the Window section into two or more windows, each with a reduced number of lines.

1.7.2 Debugger Windows

The Debugger consists of three predefined windows on which you operate to debug a program. They are summarized as follows:

Listing window -- Shows the section of the program listing at which control is currently paused. You can scroll the listing forward, backward, and horizontally.

Trap window -- Shows the existing traps and provides basic information about each trap. You can activate, deactivate, and modify the counts of traps.

Data window -- Shows the current values of specified data items. You can switch the display format for a specific data item between symbolic format and hexadecimal format.

In addition, the Debugger provides commands that enable you to perform the following window related tasks:

Manipulate windows -- Manipulate the size and contents of windows to view specific portions of data.

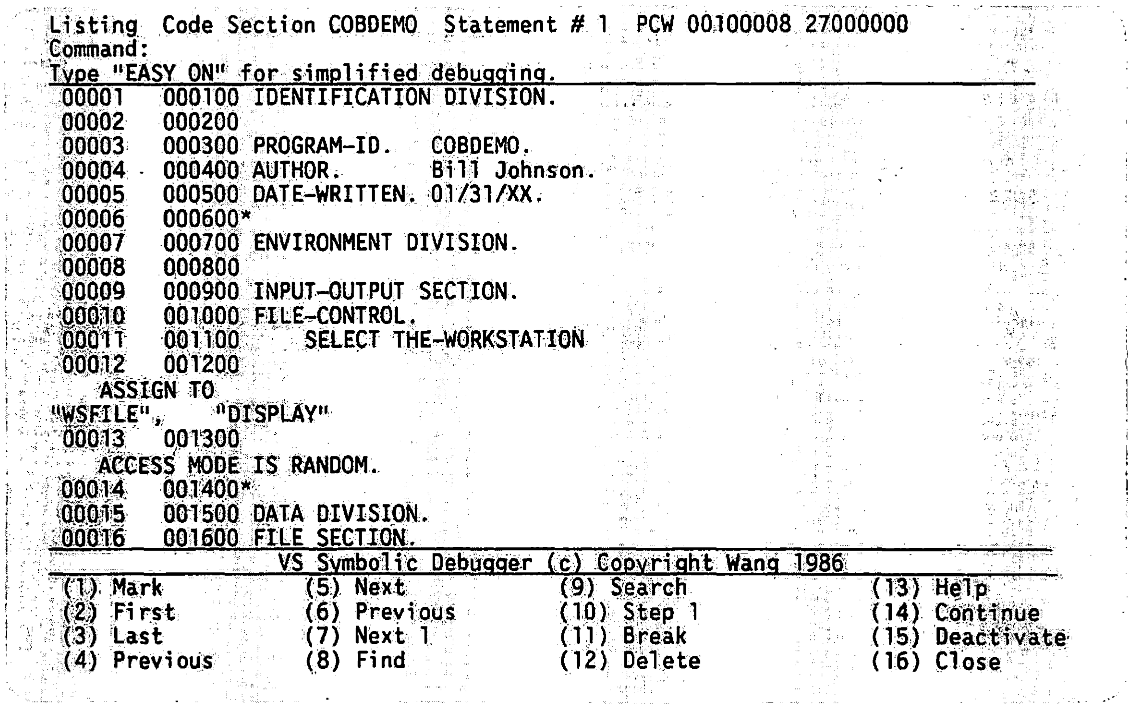
Display PF key assignments -- Display the current PF key assignments in the menu window.

Display a OS file -- Display a specified OS file on an additional window called the display window. One window is used for this purpose; a subsequent display of a different file replaces the contents of the display window.

Invoke Easy mode -- Invoke a simplified version of the Debugger for the new or infrequent user.

1.7.3 Managing the Program Listing

The first window displayed when you enter the Debugger, or whenever execution of the Debugger resumes, is the listing window (Figure 1-3). The listing window shows the portion of the program listing at which control is currently paused. The menu window is also displayed by default on the bottom four lines of the Window section.



```
Listing  Code Section COBDEMO  Statement # 1  PCW 00100008 27000000
Command:
Type "EASY ON" for simplified debugging.
 00001   000100 IDENTIFICATION DIVISION.
 00002   000200
 00003   000300 PROGRAM-ID.    COBDEMO.
 00004   000400 AUTHOR.        Bill Johnson.
 00005   000500 DATE-WRITTEN. 01/31/XX.
 00006   000600*
 00007   000700 ENVIRONMENT DIVISION.
 00008   000800
 00009   000900 INPUT-OUTPUT SECTION.
 00010   001000 FILE-CONTROL.
 00011   001100     SELECT THE-WORKSTATION
 00012   001200
     ASSIGN TO
"WSFILE",     "DISPLAY"
 00013   001300
     ACCESS MODE IS RANDOM.
 00014   001400*
 00015   001500 DATA DIVISION.
 00016   001600 FILE SECTION.
                  VS Symbolic Debugger (c) Copyright Wang 1986
 (1) Mark         (5) Next          (9) Search        (13) Help
 (2) First        (6) Previous      (10) Step 1       (14) Continue
 (3) Last         (7) Next 1        (11) Break        (15) Deactivate
 (4) Previous     (8) Find          (12) Delete       (16) Close
```

Figure 1-3. The Listing Window

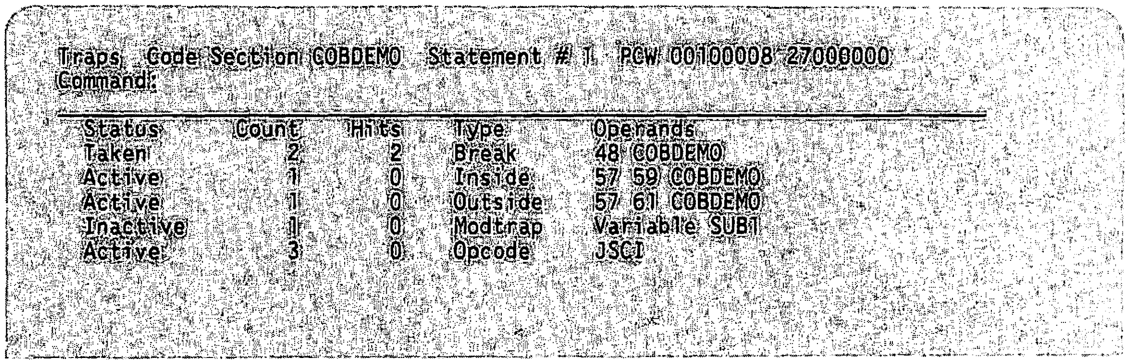
The line of the program at which control is paused appears highlighted in the listing window. You can scroll and search through the program listing to locate data names and statements that are of interest. For detailed information about the listing window, refer to Section 2.3.2.

1.7.4 Managing Traps

Traps halt program execution at desired locations in the subject program to help reveal programming errors. Generally, you set one or more traps to interrupt execution at locations where you want to examine program data or the program state.

The Debugger supports a variety of trap features. The most common trap use is to set a trap on a particular statement or range of statements in the subject program. After you have indicated the target statement(s) of the trap in the program, you set the trap by executing the appropriate trap command. After program execution has halted at a trap and after you have examined the information of interest, you can then resume the program by executing the CONTINUE command or the STEP command.

The Debugger maintains a complete list of all existing traps, called the trap list, in the trap window (Figure 1-4).



```
Traps  Code Section COBDEMO  Statement # 1  PCW 00100008 27000000
Command:
______________________________________________________________________
  Status        Count    Hits   Type        Operands
  Taken             2       2   Break       48 COBDEMO
  Active            1       0   Inside      57 59 COBDEMO
  Active            1       0   Outside     57 61 COBDEMO
  Inactive          1       0   Modtrap     Variable SUB1
  Active            3       0   Opcode      JSCI
```

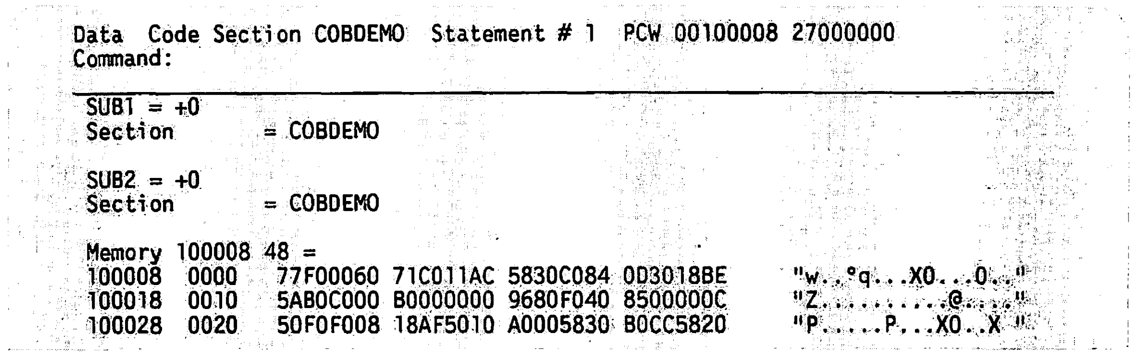
Figure 1-4. The Trap Window

Existing traps can be deactivated and later reactivated, or deleted. The Debugger automatically updates the trap list to reflect any changes. Any command that sets a trap defines the conditions under which the trap is to be taken and activates the trap.

A maximum of 99 traps can be defined at one time; however, certain conditions may exist under which not all 99 traps can be active. The number of traps that can be active at one time varies with the nature of the traps. Memory modification traps (set by executing the MODTRAP command) for lengths greater than 8 require more storage space than other traps. For detailed information about the trap window, refer to Section 2.3.3.

1.7.5 Monitoring Data Values and Displaying Program Information

The data window (Figure 1-5) contains the names and current values of program entities (variables, registers, etc.) that you specify. The Debugger monitors all entries in the data window, and always displays the current value of each entry.



```
Data  Code Section COBDEMO  Statement # 1  PCW 00100008 27000000
Command:
________________________________________________________________________
 SUB1 = +0
 Section        = COBDEMO

 SUB2 = +0
 Section        = COBDEMO

 Memory 100008 48 =
 100008  0000   77F00060 71C011AC 5830C084 0D3018BE     "w..°q...X0...0.."
 100018  0010   5AB0C000 B0000000 9680F040 8500000C     "Z.........@....."
 100028  0020   50F0F008 18AF5010 A0005830 B0CC5820     "P.....P...X0..X "
```

Figure 1-5. The Data Window

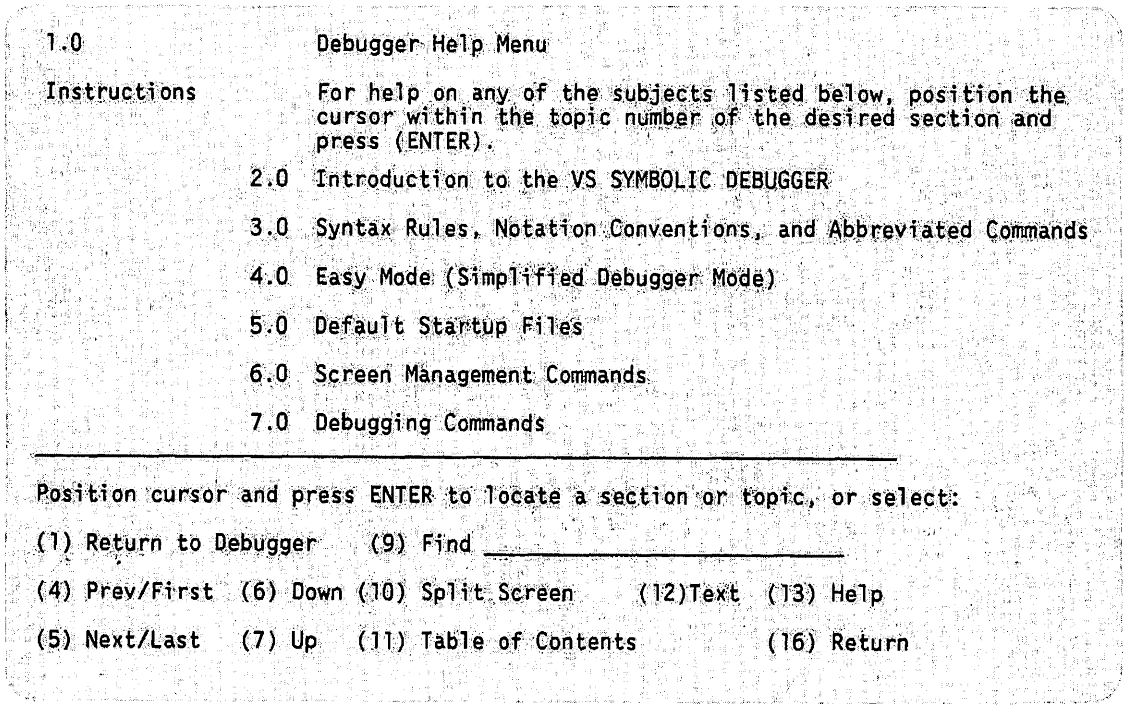
All entries in the data window are updated to contain their current values whenever they are displayed. Entries remain in the data window until you remove them by executing the DELETE command. For detailed information about the data window, refer to Section 2.3.4.

1.8 DEBUGGER HELP TEXT

The Debugger enables you to examine Help text through the HELP command. Help text consists of on-line Debugger information about the following topics:

- Debugging concepts
- Command syntax, notation, and abbreviations
- Default startup files
- Screen management commands
- Debugging commands

When you execute the HELP command, the Debugger accesses the OS INFO utility, which displays the Debugger Help menu (Figure 1-6). To access help information about a topic, position the cursor within the topic number of the desired section and press ENTER.



```
 1.0                 Debugger Help Menu

 Instructions        For help on any of the subjects listed below, position the
                     cursor within the topic number of the desired section and
                     press (ENTER).
               2.0   Introduction to the VS SYMBOLIC DEBUGGER

               3.0   Syntax Rules, Notation Conventions, and Abbreviated Commands

               4.0   Easy Mode (Simplified Debugger Mode)

               5.0   Default Startup Files

               6.0   Screen Management Commands

               7.0   Debugging Commands
_____________________________________________________________________

Position cursor and press ENTER to locate a section or topic, or select:

(1) Return to Debugger    (9) Find ___________________________

(4) Prev/First  (6) Down  (10) Split Screen     (12)Text  (13) Help

(5) Next/Last   (7) Up    (11) Table of Contents          (16) Return
```

Figure 1-6. The Debugger Help Menu

The topics that appear on the Debugger Help menu are summarized as follows:

Introduction to the OS SYMBOLIC DEBUGGER -- Displays a menu of conceptual informational topics about the Debugger.

Syntax Rules, Notation Conventions, and Abbreviated Commands -- Displays a menu of syntax rules, notation conventions, and abbreviated commands.

Easy -- Displays information about the Easy command that allows you to run the Debugger in a simplified mode that is designed for the new or infrequent user.

Default Startup Files -- Displays information about the system startup file and the optional use of a user startup file.

Screen Management Commands -- Displays a menu of the screen management commands. That information includes the syntax of each command and a short description.

Debugging Commands -- Displays a menu of the debugging commands. That information includes the syntax of each command and a short description.

Figure 1-7 shows the general screen flow for Debugger Help text.

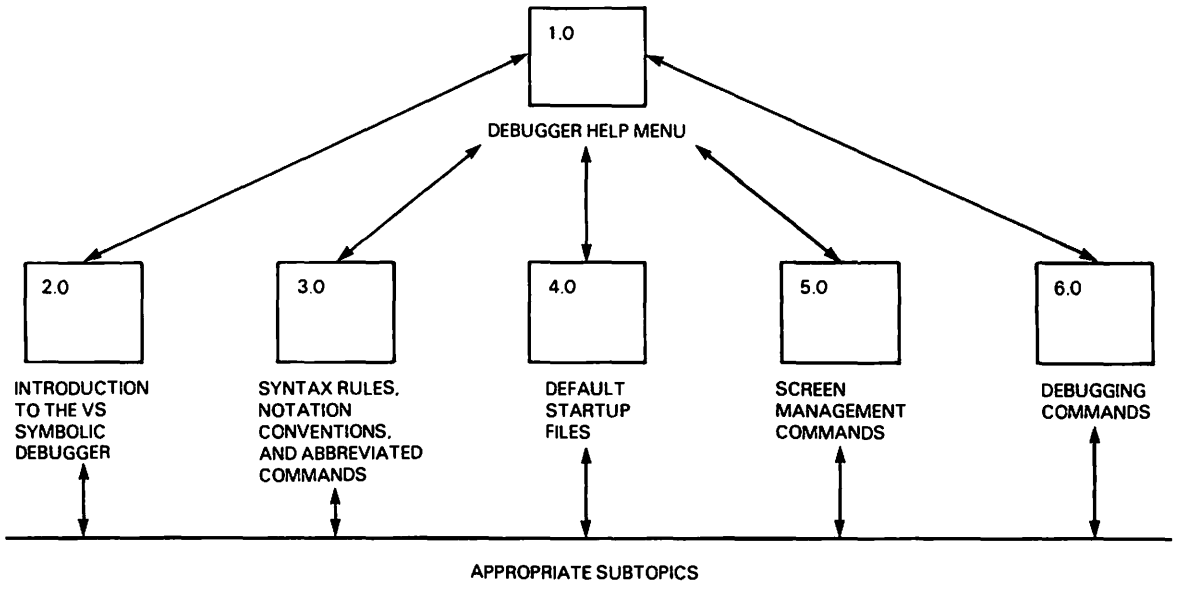

1.0
DEBUGGER HELP MENU
2.0
INTRODUCTION TO THE VS SYMBOLIC DEBUGGER
3.0
SYNTAX RULES, NOTATION CONVENTIONS, AND ABBREVIATED COMMANDS
4.0
DEFAULT STARTUP FILES
5.0
SCREEN MANAGEMENT COMMANDS
6.0
DEBUGGING COMMANDS
APPROPRIATE SUBTOPICS

Figure 1-7. Debugger Help Text Screen Flow

To select a topic, position the cursor within the topic number of the section and press ENTER. The OS INFO utility automatically takes you to the portion of the Help text with that topic number. You can continue searching through the Help text by positioning the cursor on a topic number and pressing ENTER.

When you use a topic number to go to another portion of the Help text, you can return to the previous reference by pressing PF1. By pressing PF1 repeatedly, you can revert to the reference numbers you invoked in reverse sequence until you arrive at the Debugger Help menu.

You can scroll through the Help text by using the following PF keys:

PF Key	**Description**
4 (20)	First -- Positions you at the beginning of the Help text for the Debugger, i.e., 1.0 Debugger Help menu.
5 (21)	Last -- Positions you at the end of Help text for the Debugger.
4	Prev -- Positions you at the previous screen of Help text.
5	Next -- Positions you at the next screen of Help text.
6	Down -- Positions you one line lower in the Help text.
7	Up -- Positions you one line higher in the Help text.

When you finish using the Help text, press PF16 to return to the previous display.

Note: *It is also possible to view Debugger Help text directly from the OS INFO utility. For more information about this feature, refer to the* OS System User's Introduction.

1.9 DEBUGGER TRAINING FACILITY

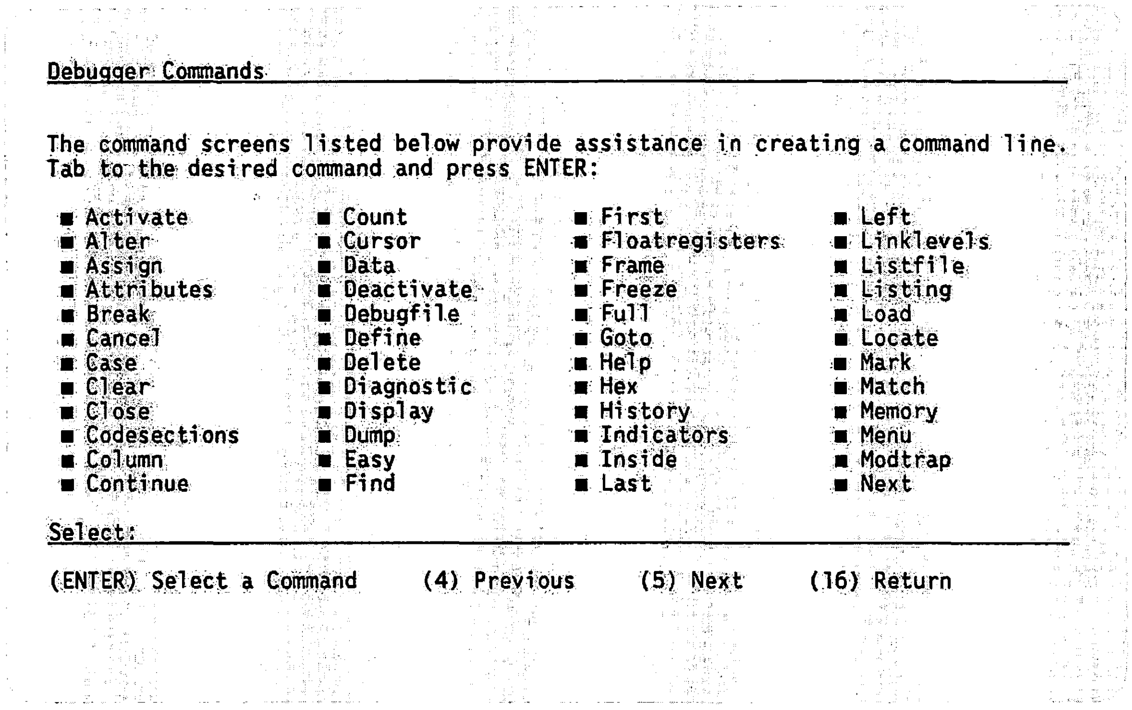
The Debugger Training facility assists you in the creation of minimally valid command strings. A minimally valid command string satisfies only the fundamental criteria for a valid command string.

The Debugger Training facility enables you to enter a command and its associated operands in the command line without regard to syntax. A fill-in-the-blank template automatically arranges the operands in the correct syntax. This allows you to make full use of the Debugger commands without regard to syntax.

There are two ways to reach the Training screen for a particular command (Figure 1-9):

- Enter *TRAINING [command]* on the command line. The *[command]* operand can be abbreviated to the shortest unique abbreviation. If any ambiguous abbreviations are encountered, the Debugger selects the first command containing the abbreviation. Invalid abbreviations cause the Debugger to display the Debugger Command screen (Figure 1-8).

- Enter *TRAINING* on the command line without an operand. The Debugger then displays the Debugger Commands screen (Figure 1-8). This method allows you to view all of the available commands.



```
Debugger Commands

The command screens listed below provide assistance in creating a command line.
Tab to the desired command and press ENTER:

 ■ Activate         ■ Count          ■ First            ■ Left
 ■ Alter            ■ Cursor         ■ Floatregisters   ■ Linklevels
 ■ Assign           ■ Data           ■ Frame            ■ Listfile
 ■ Attributes       ■ Deactivate     ■ Freeze           ■ Listing
 ■ Break            ■ Debugfile      ■ Full             ■ Load
 ■ Cancel           ■ Define         ■ Goto             ■ Locate
 ■ Case             ■ Delete         ■ Help             ■ Mark
 ■ Clear            ■ Diagnostic     ■ Hex              ■ Match
 ■ Close            ■ Display        ■ History          ■ Memory
 ■ Codesections     ■ Dump           ■ Indicators       ■ Menu
 ■ Column           ■ Easy           ■ Inside           ■ Modtrap
 ■ Continue         ■ Find           ■ Last             ■ Next
Select:

(ENTER) Select a Command     (4) Previous     (5) Next     (16) Return
```

Figure 1-8. The Debugger Commands Screen

The following functions are available from the Debugger Commands screen.

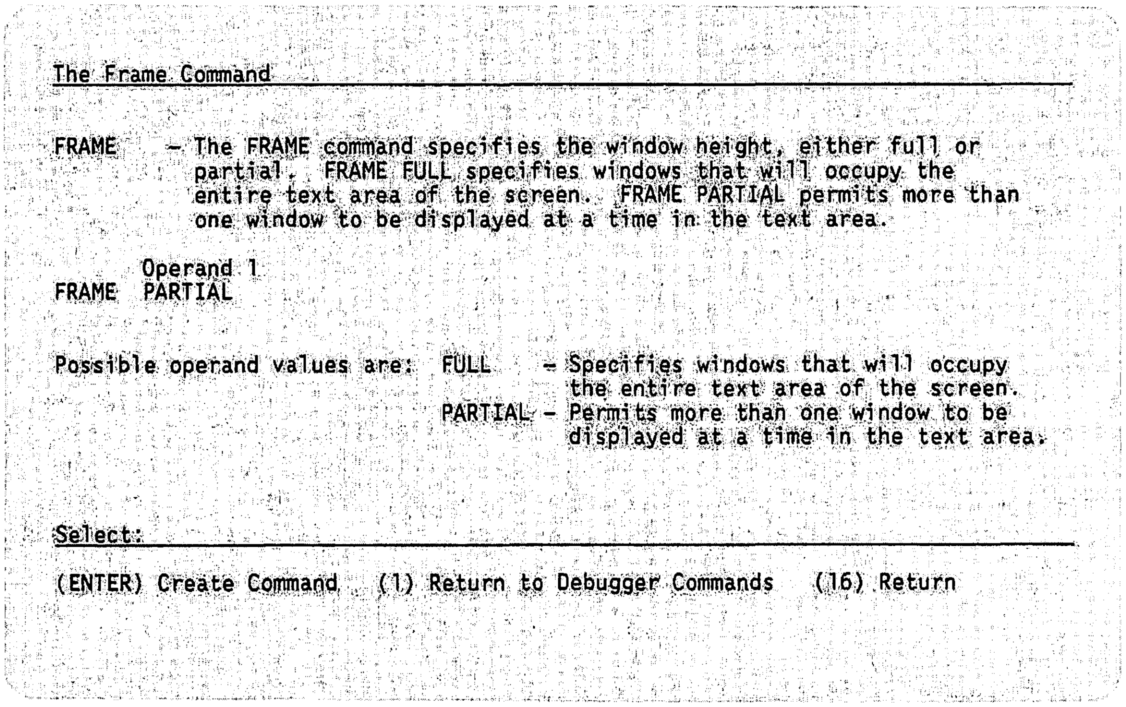
PF Key	**Function**
ENTER	Select a Command -- Selects the command at which the cursor is positioned. The Debugger then displays the Training screen for the selected command (Figure 1-9).
4	Previous -- Scrolls backwards to display the previous screen of commands.

5	Next -- Scrolls forward to display the next screen of commands.
16	Return -- Cancels the Training Facility and returns to the previous Debugger screen.

1.9.1 Using the Training Screen for a Selected Command

Either of the steps described in Section 1.9 display the Training screen for a particular command (Figure 1-9). This screen provides a short description of the command, a choice of available operands, and corresponding blanks in which to enter those operands.

In this section, the FRAME command is used as an example of a *Training command*. This section, therefore, will reference the *Frame Command screen*. The Debugger Training facility can support the rest of the Debugger commands in the same way.



```
The Frame Command
_____________________________________________________________________

FRAME    - The FRAME command specifies the window height, either full or
           partial.  FRAME FULL specifies windows that will occupy the
           entire text area of the screen.  FRAME PARTIAL permits more than
           one window to be displayed at a time in the text area.

         Operand 1
FRAME    PARTIAL

Possible operand values are:  FULL    - Specifies windows that will occupy
                                        the entire text area of the screen.
                              PARTIAL - Permits more than one window to be
                                        displayed at a time in the text area.


Select:
_____________________________________________________________________
(ENTER) Create Command   (1) Return to Debugger Commands    (16) Return
```

Figure 1-9. The Frame Command Screen

The following functions are available from the Frame Command screen.

PF Key	Function
ENTER	Create Command -- Creates the selected command in the proper syntax, based on the entered operands, and places it in the command line of the Debugger.
1	Return to Debugger Commands -- Returns you to the Debugger Commands screen (Figure 1-8) without creating a command.
16	Return -- Returns to the previous Debugger screen.

On the Frame Command screen, the Debugger displays

- A brief description of the command and its functions.
- A fill-in-the-blank template in which to enter the operands for the command line. The default operand is filled in.
- A list of the possible operand values.

On the Frame Command screen, the default operand is PARTIAL. To execute the FRAME PARTIAL command for the Debugger, press ENTER. The other possible operand value is FULL. To execute the FRAME FULL command, type FULL over PARTIAL and delete the extra characters.

1.9.2 Minimal Editing on the Training Screen

Minimal editing is defined as the fundamental validation process of the entered operand. It is done by the Debugger on the Training screen for the selected command. The Debugger will reject any attempt to enter a value besides the available listed operands. Operand fields that must contain a limited number of values or a range of values are edited accordingly. However, free formatted operands are not validated until executed from the command line. The Debugger does not verify which section names are in the program or which trap statements are valid. If you make a mistake, the Debugger does not display an error message until you attempt to execute an invalid command string from the command line.

1.10 SECURITY

Generally, you should not use the Debugger in a production environment, since a majority of the programs operate on sensitive data. For example, security problems could arise if an operator could use the Debugger to read money-card passwords. The *OS System Security Reference* contains detailed information about security on the VS.

1.11 COMPILERS THAT SUPPORT SYMBOLIC DEBUGGING

The Debugger supports symbolic debugging to the maximum extent provided by the following OS compilers:

- BASIC Compiler
- C Compiler
- COBOL 74 Compiler
- COBOL 85 Compiler
- FORTRAN 77 Compiler
- PL/I Compiler
- RPG II Compiler

The OS ASSEMBLER provides no support for symbolic data names, but BREAK traps can be set by positioning the cursor in the listing window.

1.11.1 Debugging in the Presence of Optimized Code

A number of OS compilers allow you to specify that your program undergo object code optimization. Certain aspects of object code optimization may interfere with the Debugger's ability to correctly utilize symbolic information. For example, a compiler might not store a variable's value when it is incremented, but keep that value in a register. As a result, the Debugger will display the incorrect value of that variable.

It may be necessary to recompile a program without specifying object code optimization in order to successfully perform symbolic debugging on that program.

1.12 COMPARISON TO THE PREVIOUS DEBUGGER

The following list summarizes the major differences between the new Debugger and the previous Debugger. Appendix A describes how to use the new Debugger to perform the tasks that you performed with the previous Debugger.

User interface -- The user interface of the new Debugger is command-driven and can be customized to user requirements; the user interface of the previous Debugger is menu-driven and cannot be customized.

Trap management -- The new Debugger provides extended and more comprehensive trap management.

Multiple value display -- The new Debugger enables you to display multiple program values that are updated each time the subject program is halted; the previous Debugger provides snapshots of single values only.

Symbolic memory modification traps -- The new Debugger includes memory modification traps on the symbolic level in addition to the machine level.

Symbolic RPG II indicator support -- The new Debugger enables you to display RPG II indicators symbolically.

Shared subroutine libraries and MSMAPed files -- The new Debugger lets you debug shared subroutine libraries and MSMAPed files along with your main program.

PF key assignment -- The new Debugger enables you to assign one or more selected Debugger commands to specified PF keys.

Macros -- The new Debugger enables you to build special commands called macros.

Help text -- The new Debugger provides on-line Help text through the OS INFO facility.

Note: *In the previous Debugger, it was possible to modify variables when executing in system code. As a result of enhanced security in OS Operating System Release 7.10, you must now be executing in user code in order to modify program variables.*

If your program is interrupted while in system code, single step your program to the next executable statement. Then modify the desired values in memory with the ALTER command.

1.13 NOTATION CONVENTIONS

1.13.1 Special Notation Used in Command Specifications

The special notation used in the command specifications shown in Chapter 7 is as follows:

1. All words in uppercase letters are Debugger keywords.

2. All lowercase words are user-supplied.

3. A horizontal list of several elements, such as

   ```
   x y z
   ```

   indicates that each element is to be specified in that order, one after the other.

4. The use of brackets ([]) signifies that the enclosed item(s) are optional depending on the debugging requirements. When brackets contain a vertical list of two or more items, one or none of the items can be used.

5. Brackets are not part of Debugger commands and are not included when you enter commands. All other punctuation, when included in the format, is required.

6. The operands n and m denote positive integers that you specify.

7. The operand "text" (not including quotes) denotes a text operand.

8. The volume.library.file operands denote a file specification. You can assign a full file specification in the format volume.library.filename, or specify only the file name if you want the Debugger to use the appropriate default volume and library names set in the usage constants. For information about setting usage constants, refer to the *OS System User's Introduction*.

1.13.2 Common Command Components

Some of the Debugging commands described in Chapter 7 use the syntactic components described in the following list:

Address	Some of the Debugging commands described in Chapter 7 use addresses as operands. Addresses in the subject program can be specified in either of the following ways: absolute form: hexadecimal number Example: 100598 base-index-displacement form: [disp]([ix,]b) Disp refers to a hexadecimal offset; ix and b refer to general registers. The address is the sum of the contents of the one or two registers and disp. Example: 2A(R7,RA)
hex-offset	A hexadecimal number that represents a byte offset from a specified base location; the first byte at that base location has a hexadecimal offset of 0.
hexval	A sequence of hexadecimal characters.

relop	Any of the following relational operators:

=	Equal
^= or <>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Note: *For the Not equal operator, an up-arrow (↑) can be substituted for the caret symbol (^) before the equal sign. Either the up-arrow or the caret symbol is on the 6-key, shifted (not PF6). For keyboards that do not provide the up-arrow or the caret for the shifted 6-key, you can substitute the less than/greater than relational operator combination (<>).*

section-name	The name of a section in the subject program.
statement-id	The ordinal number of a program statement as defined by the source language.

1.14 CURRENT WINDOW AND CURRENT CURSOR POSITION

The current window is the window in which the cursor is currently located. If the cursor is not in a window, the current window is the window in which the cursor was located when you last pressed ENTER or a PF key.

The current cursor position is the column and line location in the current window. If the cursor is not in a window, the current cursor position is the column and line location of the last cursor position before you last pressed ENTER or a PF key.

You can display the current cursor position on the status line if you execute the STATUS command with CURSOR as an operand, prior to a text search.

1.15 OPERATING ENVIRONMENT

The Symbolic Debugger runs on Release 7.10 and all subsequent releases of the OS Operating System. The Symbolic Debugger will not run on any operating system releases prior to Release 7.10.

To debug within shared subroutine libraries (SSLs), Version 1.05 (or greater) of the debugger is required. Symbolic Debugger 1.05 will not run on any operating system releases prior to Release 7.20.

CHAPTER 2
THE DEBUGGER USER INTERFACE

The Debugger user interface is the means by which you enter input to the Debugger and the way resultant output is displayed. The Debugger is command driven; the commands entered on the command line (or executed by PF key or macro) are the primary input mechanism, and the screen display is the primary output mechanism. This chapter describes the following aspects of the Debugger user interface:

- Summary of commands
- Screen layout
- Windows
- Command invocation
- PF keys
- Macros
- Startup files

2.1 SUMMARY OF COMMANDS

Each Debugger command belongs to one of the following categories:

Screen management commands -- Commands that enable you to perform screen management functions such as scrolling text, searching text, marking lines, and displaying windows

Debugging commands -- Commands that enable you to perform debugging actions on program data displayed in windows

Reference information about all of the Debugger commands is provided in Chapter 7. However, since commands of both types are referenced frequently prior to that chapter, the following section provides a summary of commands.

2.1.1 Screen Management Commands

The following list provides a brief description of each screen management command. Refer to Chapter 7 for reference descriptions of the screen management commands.

Command for Simplified Debugger Operation

EASY	Allows you to run the Debugger in a simplified mode that is designed for the new or infrequent user.

Commands for Positioning the Cursor

CURSOR	Performs several cursor related functions.
COLUMN	Moves the cursor to the specified text column.
ROW	Moves the cursor to the specified text row.

Commands for Scrolling Text

FIRST	Scrolls to the first line in the window.
LAST	Scrolls to the last line in the window.
LEFT	Scrolls to the left.
RIGHT	Scrolls to the right.
PREVIOUS	Scrolls backward.
NEXT	Scrolls forward.
GOTO	Scrolls to the specified line.
NOTE	Saves the window position and cursor location.

Commands for Searching Text

CASE	Specifies case sensitive or case insensitive searching.
FIND	Sets string for string searching.
MATCH	Sets string for pattern matching.
SEARCH	Locates search string (searching one direction only).
LOCATE	Locates search string (searching ahead, then back).

Commands for Marking and Clearing Lines

MARK	Marks specified line or range of lines.
CLEAR	Removes one or more marks.

Commands for Managing Windows

FRAME	Specifies full or partial window format.
WINDOW	Displays a different window.
DATA	Displays the data window.
TRAPS	Displays the trap window.
LISTING	Displays the listing window.
CLOSE	Closes the menu window or the display window only.
FULL	Makes the current window fill the screen.

Commands for Displaying Debugger Program Information

DISPLAY	Displays the specified file on the display window.
HELP	Accesses the Info facility to display Help text.
MENU	Displays current PF key assignments.
POSITION	Displays current row and column position of cursor.
TRAINING	Assists in the creation of minimally valid command strings.
VERSION	Displays the version number of the current Debugger.

Commands for Modifying the Debugger Command Set

ASSIGN	Assigns the specified command(s) to the named PF key.
DEFINE	Defines a specified macro.
HISTORY	Controls the recording of commands entered at the command line.
LOAD	Loads a command file.
RECALL	Recalls and displays a command that was entered at the command line.

Commands for Outputting Information

PRINT	Prints entire window or marked portion of window.
SNAPSHOT	Prints an image of the Debugger Workstation screen.
DUMP	Creates a memory dump.

Command to Restore the Program Screen

SCREEN	Redisplays the program screen.

Command to Exit the Debugger

CANCEL	Exits from the Debugger.

2.1.2 Debugging Commands

The following list provides a brief description of each debugging command. For reference descriptions of the debugging commands, refer to Chapter 7.

Commands for Controlling Program Execution

CONTINUE	Resumes program execution
STEP	Executes a fixed number of statements or instructions.

Commands for Changing the Section of Reference

SECTION	Selects a new default section of reference.
PROGRAMS	Displays a list of SSLs and MSMAPed files currently mapped with the main program.

Commands for Setting Traps

BREAK	Sets a trap on a particular statement or instruction.
INSIDE	Sets a trap covering the interior of a range.
OUTSIDE	Sets a trap covering the exterior of a range.
LINKLEVELS	Sets a trap on all LINK SVC or UNLINK SVC instructions.
MODTRAP	Sets a trap on an address, variable, or register.
OPCODE	Sets a trap covering all instances of an instruction.

Commands for Modifying Traps

ACTIVATE	Activates a trap.
DEACTIVATE	Deactivates a trap.
COUNT	Changes the count for a trap.

Command for Deleting Traps or Data Displays

DELETE	Deletes a trap or data display.

Commands for Displaying and Modifying High-level Values

VARIABLE	Displays the current value of a variable.
ATTRIBUTES	Displays information about a specified variable.
HEX	Changes the format of a displayed variable.
INDICATORS	Displays all RPG indicators.
ALTER	Changes a program value.
FREEZE	Specifies whether an entry in the data window is to be updated to its current value.

Commands for Displaying and Modifying Memory

MEMORY	Displays a portion of memory.
ALTER	Modifies a specified value.
FREEZE	Specifies whether an entry in the data window is to be updated to its current value.

Commands for Displaying and Modifying Program State

REGISTERS	Displays one or more general registers.
FLOATREGISTERS	Displays one or more floating-point registers.
STATE	Displays components of the program state.
ALTER	Modifies a specified value.
FREEZE	Specifies whether an entry in the data window is to be updated to its current value.
STACKTRACE	Displays the call chain.
DIAGNOSTIC	Displays the full text of a runtime diagnostic message.
PROCEDURES	Displays all subprogram names in the program.
CODESECTIONS	Displays all code sections in the program.
STATICSECTIONS	Displays all static sections in the program.

Commands for Displaying Subject Program Information

STATUS	Displays requested information in the status line.
LISTFILE	Allows you to select an alternate copy of a listing file during a debugging session.

Command for Debugging Non-Symbolic Data

DEBUGFILE	Allows you to query and reset linkage and Debugger information in non-symbolic programs by using the symbolic and linkage information in a fully symbolic copy of the same program.

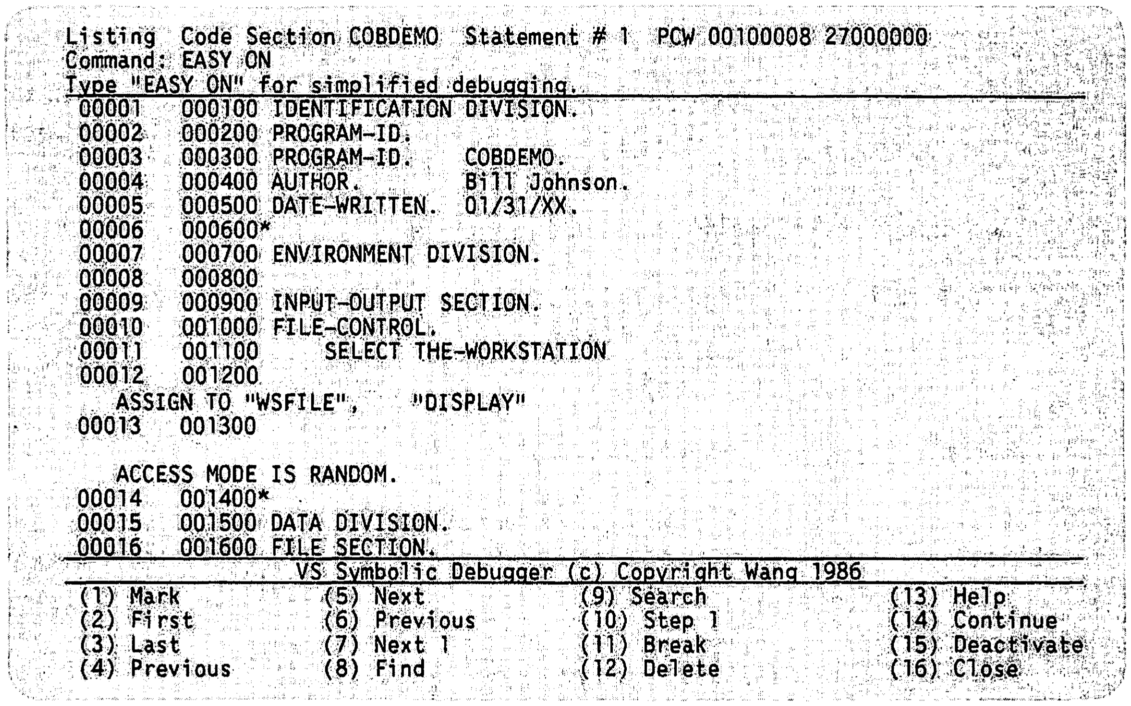
Command to Reset Default Values

SET	Allows you to reset the default values of MEMORYLENGTH and VARECHO that were set when you initially invoked the Debugger.

2.2 THE DEBUGGER WORKSTATION SCREEN

The Debugger Workstation screen (Figure 2-1) is displayed when you access the Debugger. The Debugger Workstation screen serves as the foundation for all Debugging functions; it is from this screen that you display and manage all Debugger windows.

Note: *If you are a new or infrequent user, at this point you may want to access the Easy interface to the Debugger by typing "EASY ON" in the command line of the Workstation screen. Chapter 3 describes the Easy interface.*



```
Listing  Code Section COBDEMO  Statement # 1  PCW 00100008 27000000
Command: EASY ON
Type "EASY ON" for simplified debugging.
 00001   000100 IDENTIFICATION DIVISION.
 00002   000200 PROGRAM-ID.
 00003   000300 PROGRAM-ID.     COBDEMO.
 00004   000400 AUTHOR.         Bill Johnson.
 00005   000500 DATE-WRITTEN.   01/31/XX.
 00006   000600*
 00007   000700 ENVIRONMENT DIVISION.
 00008   000800
 00009   000900 INPUT-OUTPUT SECTION.
 00010   001000 FILE-CONTROL.
 00011   001100     SELECT THE-WORKSTATION
 00012   001200
    ASSIGN TO "WSFILE",     "DISPLAY"
 00013   001300

    ACCESS MODE IS RANDOM.
 00014   001400*
 00015   001500 DATA DIVISION.
 00016   001600 FILE SECTION.
                 VS Symbolic Debugger (c) Copyright Wang 1986
 (1) Mark           (5) Next           (9) Search            (13) Help
 (2) First          (6) Previous       (10) Step 1           (14) Continue
 (3) Last           (7) Next 1         (11) Break            (15) Deactivate
 (4) Previous       (8) Find           (12) Delete           (16) Close
```

Figure 2-1. The Debugger Workstation Screen

As Figure 2-1 illustrates, the Debugger Workstation screen is divided into two sections: the Control section and the Window section.

2.2.1 The Control Section

The Control section provides window status information, enables you to enter commands, and displays error and informational messages. It consists of the following lines:

- Status line
- Command line
- Message line

The Status Line

The status line in the Control section displays user-requested information about the current debugging session relative to the topmost window. If other windows are present, each window has its own status line. You select the status items to be displayed in a window by executing the STATUS command from that window. STATUS can take several operands that indicate the desired contents of the status line. Those operands are summarized as follows:

Operand	**Description**
CODESECTION	The name of the current code section
COLUMNS	The visible column numbers of the file
CURSOR	The cursor location (after executing SEARCH or LOCATE)
LINES	The lines that are currently visible
MARK	The lines that are currently marked
MEMORY	The amount of memory available for debugging
NAME	The name of the current window
OFF	Turns the status display off for this window
PCW	Program control word information
PROGRAM	The program being executed during the current debugging session
STATEMENT	The name of the statement at which control is paused
TASKID	The number of the current user task
TIME	The current system time
USERID	The user ID of the current user
WINDOW	The current window number

Each window can have a different set of STATUS operands. You can specify as many operands as will fit on the command line in any order. The name and value of the specified operands appear on the status line of the window for which they are specified. Status items are not displayed if their contents are not meaningful or have undefined values. For example, MARK is not displayed unless there are actually lines marked. Therefore, the combined length of the status items chosen can exceed the screen width.

If you are displaying more than one window, the status display for the window nearest the top of the screen refers to that window. Status lines for lower windows are displayed as the first line of the window. If you want to add an extra line to a window for display, you can disable the status display by executing STATUS with the OFF operand, with the cursor positioned in the appropriate window. STATUS OFF, when executed for the topmost window, leaves the status line blank but does not add an additional line to the window display.

The Command Line

The command line is the primary input mechanism to the Debugger. Since this line is the first modifiable field on the screen, you can use the HOME key to move the cursor to the command line. Most commands return the cursor to the command line after the command has been executed. For information about how to enter commands on the command line, refer to Section 2.4.

The Message Line

The message line is the location at which the Debugger displays error or informational messages. These include error messages about commands and general information about the execution of commands.

2.2.2 The Window Section

The Window section of the Debugger Workstation screen can display a maximum of 21 lines of information. For example, if you display the full listing window, 21 lines of the subject program from the point at which control was paused are displayed in the Window section. When you display a partial window format for two or more windows, the number of lines shown for each window is reduced according to your specifications. For more information about the partial window format, refer to Section 4.6.1 and Section 5.5.

A Debugger window exists for each of the following entities:

- The subject program listing
- A list of the current traps
- Specified monitored program data
- Specified arbitrary VS files
- A list of PF key assignments

You can use the screen management commands (refer to Chapter 4) to perform operations such as scrolling, searching, and marking in the Window section, and the Debugging commands (refer to Chapter 5) to perform debugging operations.

If you display an arbitrary VS file in a window, there is no limit on the size of the file other than to stay within normal VS constraints.

2.3 WINDOWS

The following windows are the main Debugger windows. These windows are predefined and always appear in the Debugger:

- The listing window
- The trap window
- The data window

This section describes the format and contents of the main Debugger windows. The Debugger also creates the following additional windows:

- The display window (described in Section 2.3.5) which displays a requested VS file

- The menu window (described in Section 2.3.6) which displays the current PF key assignments

- The easy mode window (described in Chapter 3) that appears when the Debugger is operating in Easy mode

2.3.1 Text Editing Rules for Windows

The following rules govern the availability of text editing (the ability to enter data in modifiable fields) in windows:

- The text in the listing window (Figure 2-2) is not modifiable.

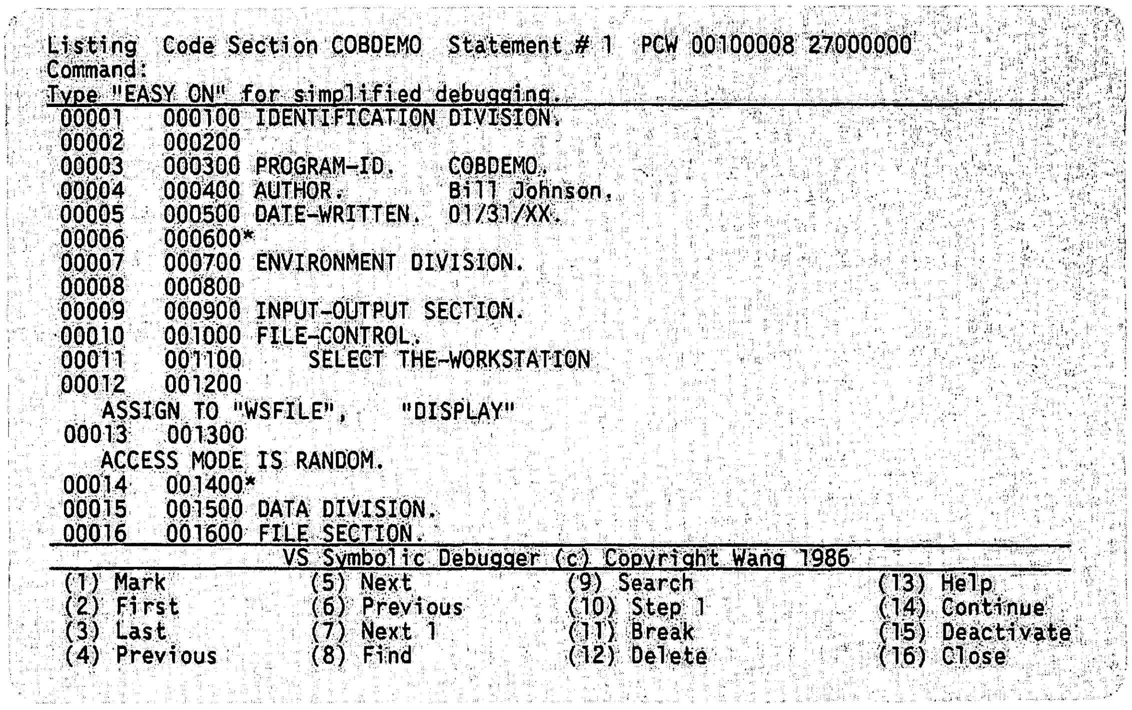
- The text in the trap window (Figure 2-3) is not modifiable. However, all changes made to traps through the ACTIVATE, DEACTIVATE, and COUNT commands are subsequently reflected on the trap window.

- The text in the data window (Figure 2-4) is modifiable. The modified value in the data window is not changed in program memory unless you execute the ALTER command. For more information about ALTER, refer to Chapter 7.

- The text in the display window and the menu window is not modifiable.

2.3.2 The Listing Window

The first window displayed on the Debugger Workstation screen when you access the Debugger is the listing window (Figure 2-2). The listing window shows the portion of the subject program listing at which control is currently paused. The menu window also appears by default with the listing window, on the last four lines of the Window section.



```
Listing  Code Section COBDEMO  Statement # 1  PCW 00100008 27000000
Command:
Type "EASY ON" for simplified debugging.
 00001   000100 IDENTIFICATION DIVISION.
 00002   000200
 00003   000300 PROGRAM-ID.     COBDEMO.
 00004   000400 AUTHOR.         Bill Johnson.
 00005   000500 DATE-WRITTEN.   01/31/XX.
 00006   000600*
 00007   000700 ENVIRONMENT DIVISION.
 00008   000800
 00009   000900 INPUT-OUTPUT SECTION.
 00010   001000 FILE-CONTROL.
 00011   001100     SELECT THE-WORKSTATION
 00012   001200
    ASSIGN TO "WSFILE",    "DISPLAY"
 00013   001300
    ACCESS MODE IS RANDOM.
 00014   001400*
 00015   001500 DATA DIVISION.
 00016   001600 FILE SECTION.
                  VS Symbolic Debugger (c) Copyright Wang 1986
 (1) Mark          (5) Next          (9) Search          (13) Help
 (2) First         (6) Previous      (10) Step 1         (14) Continue
 (3) Last          (7) Next 1        (11) Break          (15) Deactivate
 (4) Previous      (8) Find          (12) Delete         (16) Close
```

Figure 2-2. The Listing Window

The listing is initially positioned so that the statement at which control is paused appears highlighted in the listing window.

For programs that consist of a number of modules linked together, each program module has its own listing. The name of the VS listing file for each code section in the program is determined by the Debugger, from information contained in the program file. On any given entry into the Debugger, the listing that is displayed is the one for the code section in which control is paused. If you change the current program section by executing the SECTION command, the listing for the new section is displayed in the listing window, replacing the listing for the previous section.

The listing can be scrolled and searched freely. Since the text and statement numbers are intelligible to the Debugger, you can use them as arguments for many commands.

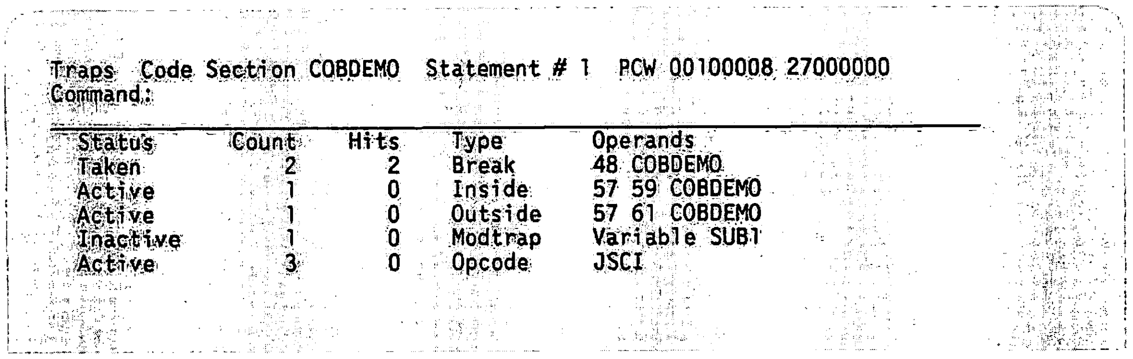
You can access the listing window from other windows by either of the following methods:

- Execute LISTING.
- Execute WINDOW with an operand of 3.

2.3.3 The Trap Window

The trap window (Figure 2-3) displays a list of all current traps. You can access the trap window from other windows by either of the following methods:

- Execute TRAPS.
- Execute WINDOW with an operand of 1.



```
Traps  Code Section COBDEMO  Statement # 1  PCW 00100008 27000000
Command:
 Status      Count    Hits    Type        Operands
 Taken           2       2    Break       48 COBDEMO
 Active          1       0    Inside      57 59 COBDEMO
 Active          1       0    Outside     57 61 COBDEMO
 Inactive        1       0    Modtrap     Variable SUB1
 Active          3       0    Opcode      JSCI
```

Figure 2-3. The Trap Window

After you create a trap, the trap is entered in the trap window. If you set the trap from a window other than the trap window, the trap appears by itself when you access the trap window, although other previously set traps may exist. To view other traps, scroll the display backward. If you set a trap while displaying the trap window, the new trap is appended to the end of the list.

Each trap, whether active or inactive, is represented by a line in the trap list. An entry for each new trap is appended to the end of the list when the trap is created.

The trap window contains the following information for each trap:

Status -- The status of the trap entry. The status can be one of the following items:

Active -- Indicates a trap that is operational and has not been taken.

Taken -- Indicates a trap that has just been taken.

Inactive -- Indicates a trap that is not operational.

Each newly created trap is displayed with an ACTIVE status. You can change the status of desired traps by executing the ACTIVATE command or the DEACTIVATE command. For information about how to execute those commands, refer to Section 5.3.1.

Count -- The count associated with the trap. The count is the number of times that the condition for the trap is to be encountered before the trap is taken. The default (1) interrupts execution the first time the specified trap condition is met. Each newly created trap is assigned a count of 1. For information about managing the count of a trap, refer to Section 5.3.2.

Note: *The counts for INSIDE, OUTSIDE, and MODTRAP traps are set at 1 and cannot be changed with the COUNT command.*

Hits -- The number of times the trap has been encountered by the subject program. When the Hits value equals the Count value, the trap is taken.

Type -- The type of trap (BREAK, INSIDE, etc.).

Operands -- The operands associated with the trap.

Note: *If the link level for the subject program changes, the Debugger displays an asterisk (*) before each trap that corresponds to a prior link level. Traps with an asterisk are not changed in any way but are no longer operational until the appropriate link level is returned to.*

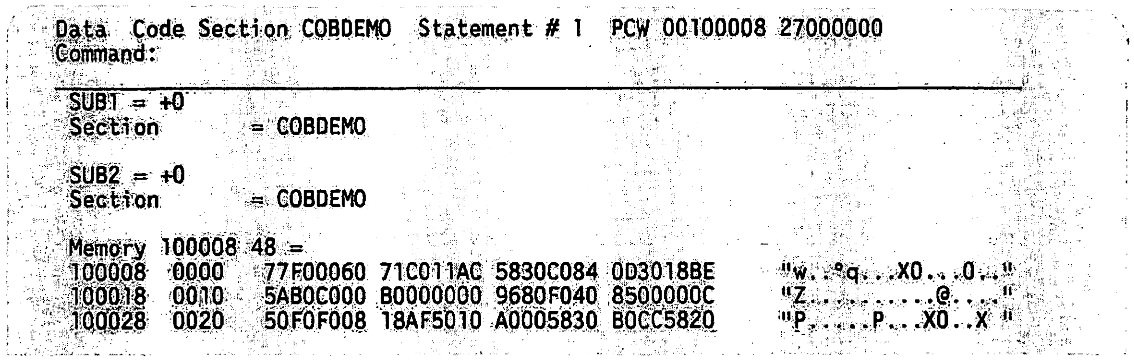
For information about modifying the trap window, refer to Section 5.3; for reference information about commands for setting traps, refer to Chapter 7.

2.3.4 The Data Window

The data window (Figure 2-4) contains specified data names, the current values of the data names, and other requested program information. The value fields of all entries in the data window are monitored by the Debugger to always contain current values.

You can access the data window from other windows by either of the following methods:

- Execute DATA.
- Execute WINDOW with an operand of 2.



```
Data  Code Section COBDEMO  Statement # 1  PCW 00100008 27000000
Command:
________________________________________________________________________
 SUB1 = +0
 Section        = COBDEMO

 SUB2 = +0
 Section        = COBDEMO

 Memory 100008 48 =
 100008  0000   77F00060 71C011AC 5830C084 0D3018BE    "w..°q...X0...0.."
 100018  0010   5AB0C000 B0000000 9680F040 8500000C    "Z..........@...."
 100028  0020   50F0F008 18AF5010 A0005830 B0CC5820    "P.....P...X0..X "
```

Figure 2-4. The Data Window

You use the VARIABLE command to enter variables in the data window; you use other various debugging commands to display information about program entities.

The information about variables that appears in the data window consists of background text and data value text. Other entries (such as program registers and procedures) appear in the format best suited to the particular entity.

Background Text

Background text refers to the names of entities that have been placed in the data window through commands that display program information. For example, if EMPLOYEE-NAME is some declared variable in the program, when the command VARIABLE EMPLOYEE-NAME is executed, a line such as the one that follows is appended to the Data list:

```
EMPLOYEE-NAME = Johnson, Bill
```

The data name EMPLOYEE-NAME and the equal sign constitute the background text in this example.

Data Value Text

Data value text represents the actual data values. For example, if in the previous example, EMPLOYEE-NAME is a 24 character string, the data value part of that entry is

```
"Johnson, Bill           "
```

The Debugger regards data values as fields of text in the data window. They have special significance; it is in them that the Debugger places the updated values of the associated data items each time control returns from the subject program. It is also from these fields that the Debugger obtains modifications to the program's data values.

You can alter the value of a field by changing the value and executing the ALTER command (after either positioning the cursor or marking the line); the value of the data item in the subject program is then changed to the new value.

Display of Variable Types

You can display variables of static storage class at any point; variables of other storage classes such as parameter, automatic, or dynamic, may or may not be available.

Variables of dynamic storage are resolved at runtime. Examples of dynamic storage variables are parameters, PL/I automatic variables, and FORTRAN dynamic variables.

The display of inactive variables that have dynamic storage has a special form. If a dynamic variable is being monitored and control returns from the subprogram in which the variable is declared, the display of the variable is not removed. It is changed to the following form:

```
variable-name is not available
```

If the subprogram is re-entered, the display of the most recent instance of that variable is resumed.

For reference information about modifying data values in the subject program, refer to the description of the ALTER command in Chapter 7.

2.3.5 The Display Window

In the course of a debugging session, you may want to display one or more arbitrary VS files. To perform this task, you execute the DISPLAY command with the appropriate file operands. The Debugger then creates a new window called the display window, and displays the file. The display window can be a full window or a partial window format depending on the status of the FRAME command.

You can display one file at a time in the display window. When you display a subsequent file, any previous file display is discarded.

To terminate the file display and close the window, you execute the CLOSE command with the cursor positioned at any location within the display window. For more information about displaying VS files, refer to Section 4.7.

2.3.6 The Menu Window

To display all of the current PF key assignments in a non-modifiable window called the menu window, execute the MENU command. The menu window can be a full window or a partial window format depending on the status of the FRAME command. The menu window appears by default in the last four lines of the Window section when you access the Debugger.

When you execute MENU with FRAME FULL in effect, the menu window is displayed as a full window with both unshifted and shifted PF keys visible. When you execute MENU with FRAME PARTIAL in effect, the menu window may be displayed as a partial window format, depending on the position of the cursor. If you display the partial window format, the entire set of PF key assignments may not be visible on the menu window. You can scroll the window display to view all assignments.

To remove the menu window, execute the CLOSE command with the cursor positioned at any location in the menu window.

2.4 COMMAND INVOCATION

You can execute Debugger commands or macros by using either the command line or PF keys that are assigned to commands. For information about macros, refer to Section 2.6.

Typically, the cursor position has meaning for the executed command or macro. The current cursor position is transmitted from the workstation to the Debugger each time you press ENTER or a PF key. You can position the cursor on a particular column of a line in a window or on the command line.

In most situations, the Debugger positions the cursor on the command line following the execution of a command. However, some commands position the cursor at some location in the window to indicate the result. For example, the SEARCH command positions the cursor at the first instance of the search string. In such cases, you can use the workstation HOME key to move the cursor to the command line.

To execute any Debugger command or macro from the command line, perform the following steps:

1. Enter the command and any desired operands on the command line. Follow the syntax rules described in Section 2.4.1.

2. If the command requires cursor positioning or line marking perform the appropriate function.

3. Press ENTER.

To execute any Debugger command or macro instruction that is assigned to a PF key, perform the following steps:

1. If the command requires cursor positioning or line marking perform the appropriate function.

2. Press the PF key assigned to the command.

2.4.1 Syntax Rules for Entering Commands

The syntax rules for entering commands are as follows:

1. You can enter commands in lowercase, uppercase, or mixed modes.

2. Operands to a command are entered on the command line next to the command, separated by one or more spaces.

3. Any operand that contains a space or a semicolon must be quoted. Single or double quote delimiters can be used; embedded quotes within quotes must have double quote delimiters. *This rule does not apply to the ASSIGN and DEFINE commands* since both commands take the remainder of the command line as a single operand.

4. You can enter two or more commands at once separated by semicolons (;). The commands are executed in the specified order when you press ENTER. *This rule does not apply to the ASSIGN and DEFINE commands* for the same reason described in rule 3.

5. You can shorten commands (not macros) to the nearest unique abbreviation; the case (upper case or lower case) of commands is ignored. The BREAK command, for example, can be shortened to B. The shortest abbreviation varies with the command in question; Appendix D contains a list that shows the shortest unique abbreviation for each command. If the Debugger encounters ambiguous or unknown command names, it displays an error message and takes no action.

6. Macros must be spelled out fully and have the correct case.

Note: *A macro that has the same name as a command or a command abbreviation overrides that command or abbreviation.*

2.5 PF KEYS

A total of 32 PF keys are available for use with the Debugger; 16 unshifted keys and 16 shifted keys. PF keys are indicated in this manual in the form PFn, where n is any integer from 1 to 32.

Note: *PF keys 33 through 256 are accessible through the ASSIGN command but are reserved for future use.*

PF keys 1 through 16 are unshifted and are executed simply by pressing the key. PF keys 17 through 32 are called shifted keys since they use the uppercase versions of PF keys 1 through 16.

To access PF keys 17 through 32, hold down the SHIFT key and press the appropriate key. You determine the correct key to press by subtracting 16 from the shifted number. For example, to execute PF25, you press PF9 (25 - 16 = 9) *while simultaneously holding down the SHIFT key.*

Default command assignments are provided for all 32 PF keys in the system startup file (refer to Section 2.7.1). You can view current assignments by executing the MENU command. In addition, you can exercise either of the following options to change PF key assignments:

- Assign one or more commands to any PF key by executing the ASSIGN command (refer to section 2.5.1). The default PF key assignment is overridden for the current session only.

- Place a user startup file in your input library to override PF key assignments in the system startup file when the Debugger is invoked. At any time during the debugging session, you can reload the user startup file by executing LOAD DEBSTART to restore all initial contents. For information about user startup files, refer to Section 2.7.2.

When a PF key is pressed, its assigned command is executed. If you assigned two or more commands to a PF key, they are executed one at a time in the order in which they are specified.

2.5.1 Assigning Commands to PF Keys

You can assign one or more commands to a PF key by either of the following methods:

- Enter ASSIGN followed by the command(s) (each command separated by a semicolon) that you want to assign, and then press the PF key to which the commands are to be assigned. For example, to assign a command to mark the first line of the window to PF1, you would enter

  ```
  ASSIGN FIRST; MARK
  ```

  and press PF1. This assigns the command string FIRST; MARK to PF1, removing any commands previously assigned to PF1.

- Enter ASSIGN followed by the number of the PF key you want to assign the command(s) to, and the commands (each command separated by a semicolon) to be assigned. Then, press ENTER. To do the previous example by this method, you would enter

  ```
  ASSIGN 1 FIRST; MARK
  ```

  and press ENTER.

Note: *Validation of the assigned command string occurs during the execution of the string, not at the time you assign the string. For example, if you execute ASSIGN 1 MAKK, the Debugger assigns the command string MAKK to PF1 even though MAKK is not a valid command. When you execute PF1, the Debugger informs you that the command string (MAKK) is invalid.*

2.5.2 Examining PF Key Assignments

You can determine which commands are currently assigned to a PF key by either of the following methods:

- Enter ASSIGN and press the PF key. The PF key assignment is displayed on the command line. To change the assignment, you enter the new assignment over the previous one and press the PF key.

- Enter ASSIGN followed by the number of the PF key and press ENTER. The PF key assignment is displayed on the command line following the PF key number. To change the assignment, you enter the new assignment over the previous one and press ENTER.

The system startup file contains default command assignments for all 32 PF keys. To examine each default assignment, refer to Section 2.7.1.

2.5.3 Changing a PF Key Assignment During Execution

As a rule, if a command takes an operand and it is assigned to a PF key, you cannot change the operand in the process of executing the command. The exception is the subset of commands that set a condition in the debugging environment, such as FIND and MATCH. For all other commands, to use an operand different from the one specified by the command, you have the following three options:

- Redefine the PF key with the desired change (using the ASSIGN command) before you perform the operation.

- Enter the command on the command line with the change instead of using the PF key.

- Assign variations of the command to different PF keys.

After you execute ASSIGN, the system assigns the specified command(s) to the indicated PF key, removing any prior command string assigned to that key.

2.5.4 Removing a PF Key Assignment

To remove a PF key assignment, reassign the key as one or more spaces. For example, if you execute ASSIGN 2 ' ', the previous command assignment of PF2 is removed and no commands are assigned to that key. For reference information about assigning commands to PF keys, refer to the ASSIGN command in Chapter 7.

2.6 MACROS

In addition to assigning one or more commands to a PF key, you can assign one or more commands to a name of your choice. A command of this type is called a macro; the process is called "defining a macro".

You define a macro by entering DEFINE followed by the name of the macro and the command(s) you want to assign. Then, you press ENTER. For example, to define a macro called FULLDISP that displays the file DISPFILE in a full display window, you enter the following information, then press ENTER:

```
DEFINE FULLDISP FRAME FULL; DISPLAY INVOL.INLIB.DISPFILE
```

Whenever you execute FULLDISP, the string of commands associated with that macro is executed. Once a macro has been defined, you can go a step further and assign that macro to a PF key if desired, as explained in Section 2.5.1.

Note: *Validation of the defined command string occurs during the execution of the string, not at the time you assign the string. For example, if you execute DEFINE M5 MAKK 5, the Debugger defines the macro M5 as the command string MAKK 5, even though MAKK 5 is not a valid command. When you execute M5, the Debugger informs you that the command string (MAKK 5) is invalid.*

In certain cases, it is useful to use a macro as a shorter name for a command with a long name. For example, since STATICSECTIONS has a minimum abbreviation of 5 letters, you might want to define a macro called "st" as a shorter abbreviation for that command.

2.6.1 Differences Between Predefined Commands and Macros

There are two significant differences between predefined commands and macros. First, commands are insensitive to case and can be abbreviated if the abbreviation is unambiguous. Macros are case sensitive; thus MAC is different from Mac or mac. Second, macros must always be fully spelled out.

Since macros are a sophisticated feature, this behavior minimizes the possibility of undesirable reactions between macros and commands.

Note: *A macro that has the same name as a command or a command abbreviation overrides that command or abbreviation.*

2.6.2 Examining Macro Definitions

To examine the current definition of a macro, execute DEFINE followed by the macro name. If a definition exists for the specified macro, that definition is displayed on the command line. You can either change the displayed definition or enter a new one. If no definition exists, only the entered command is displayed on the command line. You can enter a macro definition if desired.

2.6.3 Removing a Macro Definition

Removing a macro definition also removes the macro from your Debugger command set. To remove the definition of a macro, redefine the macro as itself. For example, if you execute DEFINE FULLDISP FULLDISP, the previous macro definition of FULLDISP is removed, and the macro FULLDISP no longer exists.

2.7 DEFAULT STARTUP FILES

A default startup file contains defaults for PF key assignments and various other options to be used by the Debugger. A system startup file is supplied with the Debugger. The system startup file is automatically loaded, executed, and closed at the start of each debugging session.

Using the VS Editor, you can create a user startup file which when executed, overrides the system startup file. If a user startup file with the correct volume, library, and file names exists on the system, the Debugger executes that file in addition to the system startup file. In this case, PF key assignments and commands in the user startup file override the same PF key assignments in the system startup file, and any additional functions are included. Any PF key assignments or commands in the system startup file that were not overridden remain operational.

In addition to PF key assignments, other startup types of commands (STATE, REGISTERS, etc.), and macro definitions can be included in a default startup file.

2.7.1 The System Startup File

The system startup file (called DEBSTART) that is supplied with the Debugger has the following attributes:

```
VOLUME  = SYSVOL
LIBRARY = @SYSTEM@
FILE    = DEBSTART
```

When you invoke the Debugger initially, it reads the system startup file to establish the default PF keys and to perform any other specified initialization. The Debugger then looks for a user startup file; if one exists, it is executed in addition to the system startup file. The following list shows the PF key assignments in the system startup file.

(1)	MARK	(17)	CLEAR
(2)	FIRST	(18)	DATA
(3)	LAST	(19)	LISTING
(4)	PREVIOUS	(20)	TRAPS
(5)	NEXT	(21)	STATE
(6)	PREVIOUS 1	(22)	REGISTERS
(7)	NEXT 1	(23)	FLOATREGISTERS
(8)	FIND	(24)	ALTER
(9)	SEARCH	(25)	SEARCH BACKWARD
(10)	STEP 1	(26)	STEP INSTRUCTION 1
(11)	BREAK	(27)	CODESECTIONS
(12)	DELETE	(28)	STATICSECTIONS
(13)	HELP	(29)	STACKTRACE
(14)	CONTINUE	(30)	DUMP
(15)	DEACTIVATE	(31)	ACTIVATE
(16)	CLOSE	(32)	CANCEL

If you have changed a number of command assignments (via the ASSIGN command), and then want to go back to the original assignments of the system startup file, you execute LOAD SYSVOL.@SYSTEM@.DEBSTART. The system startup file is reloaded and the original PF key assignments are reinstated. For reference information about ASSIGN, refer to Chapter 7.

2.7.2 The User Startup File

Using the VS Editor, you can create a user startup file for the Debugger to use in addition to the system startup file. Your user startup file can be a slightly edited version of the system startup file or a completely different file. Typical uses of the user startup file are to load a personal set of PF key or macro definitions, or to execute a set of commands repeatedly.

The Debugger will not execute a user startup file unless it is an 80-byte consecutive file. The Debugger verifies and then executes each line of the user startup file separately. Any invalid lines cause the Debugger to halt execution of the file and inform you with a message. If the file is not an 80-byte consecutive file, an error message stating that the file is not a startup file is displayed.

You create a user startup file through the VS Editor. All commands must appear in the first 72 bytes of the record. You can create files with line numbers in the correct columns by specifying PL/I, FORTRAN, Assembly language, or the procedure interpreter as the source language to the Editor. If you specify BASIC, COBOL, COBOL85, or RPG II as the source language, *and do not specify NO for the NUMBER option,* the Editor will create an invalid startup file, since the line numbers precede the text for these languages. For more information about creating files, refer to the *VS Editor Reference*.

PF key assignments in a loaded user startup file override the same PF key assignments in the system startup file. Any PF keys or commands defined in the system startup file that are not redefined by the user startup file remain active.

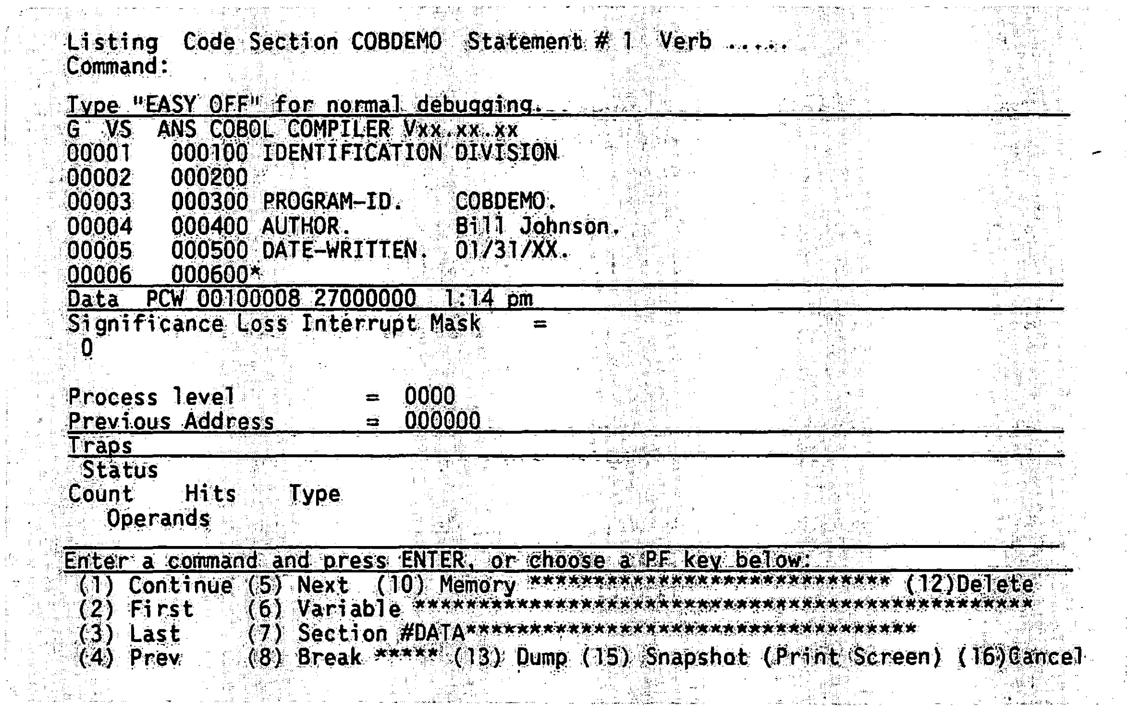
For the Debugger to recognize a user startup file and execute it after the system startup file, you must name the user startup file DEBSTART, and place it in your INLIB on the INVOL.

It is also possible to load a user startup file by executing the LOAD command with the volume, library, and file names as operands. For reference information about how to load files, refer to the LOAD command in Chapter 7.

CHAPTER 3
EASY MODE

3.1 INTRODUCTION

The EASY command enables you to use the Debugger in a simplified mode that is PF-key driven and designed for the new or infrequent user. To begin using EASY, type EASY ON, on the Debugger command line and press ENTER. The system then displays the Easy Interface screen shown in Figure 3-1.



```
Listing  Code Section COBDEMO  Statement # 1  Verb ......
Command:

Type "EASY OFF" for normal debugging.
G  VS  ANS COBOL COMPILER Vxx.xx.xx
00001    000100 IDENTIFICATION DIVISION
00002    000200
00003    000300 PROGRAM-ID.     COBDEMO.
00004    000400 AUTHOR.         Bill Johnson.
00005    000500 DATE-WRITTEN.   01/31/XX.
00006    000600*
Data   PCW 00100008 27000000   1:14 pm
Significance Loss Interrupt Mask     =
 0

Process level           =  0000
Previous Address        =  000000
Traps
 Status
Count    Hits    Type
   Operands

Enter a command and press ENTER, or choose a PF key below:
 (1) Continue (5) Next  (10) Memory ****************************** (12)Delete
 (2) First    (6) Variable ************************************************
 (3) Last     (7) Section #DATA***********************************
 (4) Prev     (8) Break ***** (13) Dump (15) Snapshot (Print Screen) (16)Cancel
```

Figure 3-1. The Easy Interface Screen

3.2 EASY FUNCTION KEYS

The Easy mode of Debugger operation enables you to use PF keys to perform many standard debugging operations. The following list describes each of the operations that you can access by using PF keys from the Easy interface screen. Note that these are not the only Debugger operations that you can access through EASY.

PF Key	**Function**	**Description**
PF1	CONTINUE	Allows you to continue the execution of your program after it has halted at a breakpoint or trap.
PF2	FIRST	Allows you to display the first seven source lines in the window.
PF3	LAST	Allows you to display the last seven source lines in the window.
PF4	PREVIOUS	Allows you to display the previous seven source lines in the window.
PF5	NEXT	Allows you to display the next seven source lines in the window.
PF6	VARIABLE	Allows you to display specified program variables in the window. The current value of the variable is always displayed.
PF7	SECTION	Allows you to display the source code of a specific code section.
PF8	BREAK	Allows you to set a trap or a breakpoint to halt the execution of your program.
PF10	MEMORY	Allows you to display a specified range of the subject program's memory in the window.
PF12	DELETE	Allows you to delete specified traps from the trap window, or specified entries from the data window.
PF13	DUMP	Allows you to obtain a dump of the modifiable data area. This function creates a print file of the program's variables, buffers, and control blocks.
PF15	SNAPSHOT	Allows you to print the Debug screen currently displayed.

PF Key	Function	Description
PF16	CANCEL	Allows you to cancel the Debugging session and return to the Command Processor screen or to the program where the interrupt occurred.

3.3 LIMITATIONS OF EASY

Easy does not provide you with the use of the entire Debugger command set. The commands that modify PF key assignments and manipulate windows are disabled. However, you can execute any of the remaining commands at the command line. In addition, commands are assigned to only 16 PF keys, instead of all 32 PF keys. Finally, the CLOSE command works differently.

The following list describes each of the commands that work differently while you are using the Easy interface.

ASSIGN -- You cannot reassign PF keys while you are in Easy mode. When you attempt to reassign PF keys, the system displays the following error message:

"PF key assignments cannot be modified in simplified mode."

CLOSE -- The cursor must be positioned outside the easy mode window when using CLOSE. When you attempt to close the window with the cursor position inside the easy mode window, the system displays the following error message:

"CLOSE is not legal in the easy mode window."

DATA, DISPLAY, FRAME, LISTING, MENU, TRAPS, and WINDOW -- These commands are not valid while you are in Easy mode. When you attempt to use any of these commands, the system displays the following error message:

"The Debugger window structure cannot be altered while in simplified mode."

3.4 EXITING EASY MODE

To exit Easy mode and return to normal Debugging, type EASY OFF on the command line and press ENTER.

CHAPTER 4
SCREEN MANAGEMENT

4.1 INTRODUCTION

The Debugger provides a variety of screen management commands that enable you to perform tasks such as those that follow:

- Position the cursor
- Scroll text
- Search text
- Mark lines
- Manage windows
- Display information
- Output information
- Assign commands to PF keys

This chapter describes how to perform these tasks. Its objective is to provide a general set of guidelines for using commands and various operands to perform common screen management tasks. Familiarity with the tasks described in this chapter will enable you to successfully understand and use the screen management commands described in Chapter 7.

This chapter does not provide complete information about syntax and usage of the commands described; refer to Chapter 7 for that information.

4.2 POSITIONING THE CURSOR

The current cursor position is transmitted from the workstation to the Debugger each time ENTER or a PF key is pressed. You can position the cursor on a particular column of a selected row within a window, or on the command line.

Many commands automatically return the cursor to the command line after they are executed. For those commands that leave the cursor within the current window after they are executed, you can press the HOME key to automatically move the cursor to the command line.

The Debugger includes several screen management commands for positioning the cursor at desired locations. For reference information about those commands, refer to Chapter 7.

4.3 SCROLLING TEXT

The FIRST, LAST, PREVIOUS, and NEXT commands perform forward and backward scrolling. FIRST displays the first portion of the window display and LAST displays the last portion of the window display. PREVIOUS scrolls backward through the displayed data and NEXT scrolls forward. Both PREVIOUS and NEXT accept a numeric operand that indicates the number of lines to be scrolled.

The LEFT and RIGHT commands perform horizontal scrolling. LEFT scrolls the window display a specified number of columns to the left, which is indicated by a numeric operand. RIGHT scrolls the window display a specified number of columns to the right, which is also indicated by a numeric operand.

For reference information about commands for scrolling text, refer to Chapter 7.

4.4 SEARCHING TEXT

When you search for text, you first specify a string to be searched for, called the search string, and then execute the SEARCH command or the LOCATE command to locate it. The search string is specified by either the FIND command or the MATCH command and serves as the search string for both SEARCH and LOCATE. Each window can have one search string.

The string that was last specified by either FIND or MATCH serves as the search string. If a search string was last specified by FIND, the search criteria for FIND are applied in any subsequent search. If a search string was last specified by MATCH, the search criteria for MATCH are applied in any subsequent search.

4.4.1 Specifying a Search String

The FIND command specifies a search string for a simple text search. For example, assume that SECTION-1 is some section in the subject program. If you execute FIND SECTION-1, SECTION-1 becomes the search string.

The MATCH command specifies a more sophisticated search string (also called a regular expression) for pattern matching. Pattern matching involves searching for match strings using both text and symbols as the search criteria. MATCH enables you to specify an approximation of the string you want to locate, for which a number of different strings within the text may match.

Regular expressions use certain special symbols to indicate the desired type of search. The brackets ([]) and the asterisk (*) are examples of special symbols. The brackets describe a character class; [abc] stands for a single character that is either "a", "b", or "c". The asterisk means to repeat the preceding expression zero or more times; a* means zero or more letter a's. For reference information about additional special symbols, refer to Chapter 7.

The status of the CASE command applies only to a search string specified by FIND. CASE ANY indicates that the search should ignore the case of the text; CASE EXACT indicates that the search should be case sensitive. A search string specified by MATCH is always case sensitive regardless of the status of the CASE command.

The search string specified by either the FIND command or the MATCH command is saved until a new search string is specified; only one search string can exist per window. The current search string for the window is the last one that was specified by either command. For example, if you issue a search string through the MATCH command and then issue a search string through the FIND command, the string issued by FIND is the current search string.

4.4.2 Locating the Search String

After you have established a search string, you execute either SEARCH FORWARD or SEARCH BACKWARD to perform a search. SEARCH FORWARD searches forward from the current cursor position; SEARCH BACKWARD searches backward from the current cursor position. If a search string is found, the Debugger positions the cursor to the first character of the found string. If the search string is not found, a message is displayed.

You can use the LOCATE command to search first forward and then backward. LOCATE searches forward for the search string, and if it is not found, then searches backward from the starting point. In this way if the search string is not found, you know that it does not exist anywhere in the file.

To find all occurrences of a particular string, specify the search string using FIND or MATCH, then execute SEARCH (from the first line of the window) or LOCATE repeatedly until no more occurrences of the string are found. For reference information about commands for searching text, refer to Chapter 7.

4.4.3 Pattern Matching

Pattern matching is an advanced type of search made possible by the MATCH command (MATCH ON). It enables you to locate strings with variable components. When MATCH is set to OFF, pattern matching is not in effect, and any search is a simple, literal search.

Pattern matching uses certain symbols as wild cards for specifying characters and positions within the search string. The symbols are punctuation marks and other common symbols, such as a period (.), brackets ([]), and the asterisk (*) (refer to Table 4-1). You use these symbols together with the usual characters to specify the object of your search, the find-string. (You can specify the find-string with the FIND, SEARCH, or LOCATE command.) The ON setting for the MATCH command causes any of the pattern matching symbols of the find-string to be interpreted according to their meaning within pattern matching, instead of their literal representation.

Some of the pattern matching symbols represent the repetition of characters, some represent one or more members of a class, some represent the location of the expression within a string (first or last), and two symbols serve as escape characters. For example, if MATCH is set to ON and a pair of brackets is used to enclose abc in a find-string [abc], the pattern matching interpretation is "Find every string that is an a, b, or c."

Your setting of the CASE command (CASE ANY or CASE EXACT) affects pattern matching just as it does simple searches. If CASE ANY is in effect, both "a" and "A" match a pattern in which "a" or "A" is indicated. The examples in Table 6-1 are the result of a setting of CASE EXACT.

Pattern matching in the Debugger is done on a line by line basis. There is no facility for matching a string that occurs partly on one line and partly on the next line.

Table 4-1. Pattern Matching Symbols

Symbol	Description
any	A character that does not have one of the following special meanings stands for itself.
.	A period represents any single character.
[]	Brackets represent a character group from which one character must be chosen for the match. Characters, character ranges, and the caret symbol (description follows) can be combined within a single set of square brackets. In addition, you can use more than one set of brackets in a pattern matching expression. *Examples* [abc] represents a single character that is either lowercase a, b, or c. [A-Z] represents one character in the set of uppercase letters A through Z. [A-Za-z] represents a single alphabetic character. The letter can be uppercase (A-Z) or lowercase (a-z). [A-Z] [A-Z] [A-Z] represents any three consecutive uppercase letters.
^	The caret symbol, when it appears as the first character inside brackets, represents a single character <u>not</u> in the character class. The caret represents the word "not" when it is in this position. When it appears at the beginning of an expression, the caret symbol indicates that the text must start at the beginning of the line in order to match.
	Note: *The caret symbol (^) may appear as an up-arrow (↑) or some other character, depending on the workstation being used.*

(continued)

Table 4-1. Pattern Matching Symbols (continued)

Symbol	Description
	Examples [^0-9] represents any single character not in the set 0 through 9. [^A-Za-z0-9] represents any single character that is not alphabetic or numeric. [^BEGIN] represents any single character that is not the uppercase letter B, E, G, I, or N. ^BEGIN matches the string BEGIN only when that string starts at the beginning of the line.
*	The asterisk indicates that the preceding character or expression can be repeated zero or more times. *Examples* .* indicates zero or more characters. [A-K]* represents zero or more uppercase letters in the range A-K.
+	The plus sign indicates that the preceding character or expression can be repeated one or more times. *Examples* .+ indicates one or more characters. [0-9]+ represents one or more digits that can range from 0 through 9 inclusive.
?	The question mark indicates that the preceding expression cannot appear at all or can appear only once. *Example* mp? indicates that a lowercase p can appear zero or one time to qualify for a match. For example, "m" and "mp" are matches; "mpp" is not a match (because lowercase p appears more than once).

(continued)

Table 4-1. Pattern Matching Symbols (continued)

Symbol	Description
$	The dollar sign at the end of an expression indicates that the search string must appear at the end of a line in order to match. *Example* END$ matches the string END only when that string appears at the end of a line.
/	The slash is used as an escape character; that is, the slash removes any special meaning *from the following single character only*. If the following single character has no special meaning, the slash is ignored. The backslash (\\) used in place of the slash produces the same result. *Examples* /*ab.* represents an expression that begins with *ab followed by zero or more characters. The slash indicates that the special use of the following asterisk should be ignored within pattern matching. Thus, the search is made for an actual asterisk followed by "ab". The second asterisk has its unique use within pattern matching; the repetition zero or more times, of the preceding character. The period represents any character. [0-9]/+ represents any single digit followed by a plus sign. The slash removes the unique meaning within pattern matching of the plus sign.

4.5 MARKING LINES

Most debugging functions act upon a single line or a range of lines. You can indicate a single line or delimit a series of lines by marks. Most line-oriented commands act on the line that contains the cursor *only if no lines are marked*. For example, you delete a single line by positioning the cursor at any location within the line and executing DELETE. However, you can also delete the line by first marking it and then executing DELETE. The marked line is deleted without regard to the position of the cursor.

4.5.1 How to Mark Lines and Ranges

To mark a line, position the cursor at any location on the line and execute the MARK command. You can position the cursor by using commands for scrolling text. As a result of executing MARK, a triangle appears in the first column to the left of the specified line.

An alternate way to mark a single line is to execute MARK followed by the number of the line to be marked.

To mark a range of lines, first position the cursor on the line to serve as one range boundary and execute MARK. The Debugger marks the specified line. Then, position the cursor on the line to serve as the other range boundary and execute MARK again. The Debugger marks all lines from the initially marked line to the line at which the cursor is positioned, including that line.

To mark all lines in the window, execute the MARK ALL command.

Only one range of marked lines can exist at one time in a window. When a line or range of lines is already marked, and MARK is again executed, the Debugger marks all lines between the existing range and the new line.

Note: *MARK operates on one entire entry in the data window; if you execute MARK on one line of an entry, the entire entry is marked. Only one entry can be marked at a time. Thus, if an entry is marked in the data window, and MARK is executed against another entry in the data window, the previously marked entry is cleared.*

4.5.2 How to Clear Marks

The CLEAR command removes marks from one or more marked lines. To unmark a single line (if that line is the only one marked), position the cursor at that line and execute CLEAR. The Debugger removes the corresponding mark.

To unmark a portion of a marked range of lines, position the cursor at the desired location within the range and execute either CLEAR NEXT or CLEAR PREVIOUS. CLEAR NEXT removes marks from all lines that follow the current cursor line; CLEAR PREVIOUS removes marks from all lines that precede the current cursor line.

To unmark an entire range of lines, execute the CLEAR or CLEAR ALL command. The Debugger removes all marks from the range. To unmark an entire range of lines regardless of the position of the cursor, execute the CLEAR ALL command.

For reference information about commands for marking and clearing lines, refer to Chapter 7.

4.6 MANAGING WINDOWS

The following windows are the main Debugger windows, which are predefined and always appear in the Debugger:

- The listing window
- The trap window
- The data window

The CLOSE command (which removes a window) has no effect on the predefined Debugger windows.

The Debugger optionally builds the following windows that are not predefined:

Menu window -- Displays the current PF key assignments.

Display window -- Displays a requested VS file.

Easy mode window -- Appears when the Debugger is operating in Easy mode.

The CLOSE command operates on the menu window and the display window.

4.6.1 Full Window and Partial Window Formats

The following two formats of window display are supported:

Full window format -- Each accessed window occupies the entire 21-line window section area.

Partial window format -- Each accessed window can occupy all or a portion of the window section, depending on the cursor position when the command to access the window is executed.

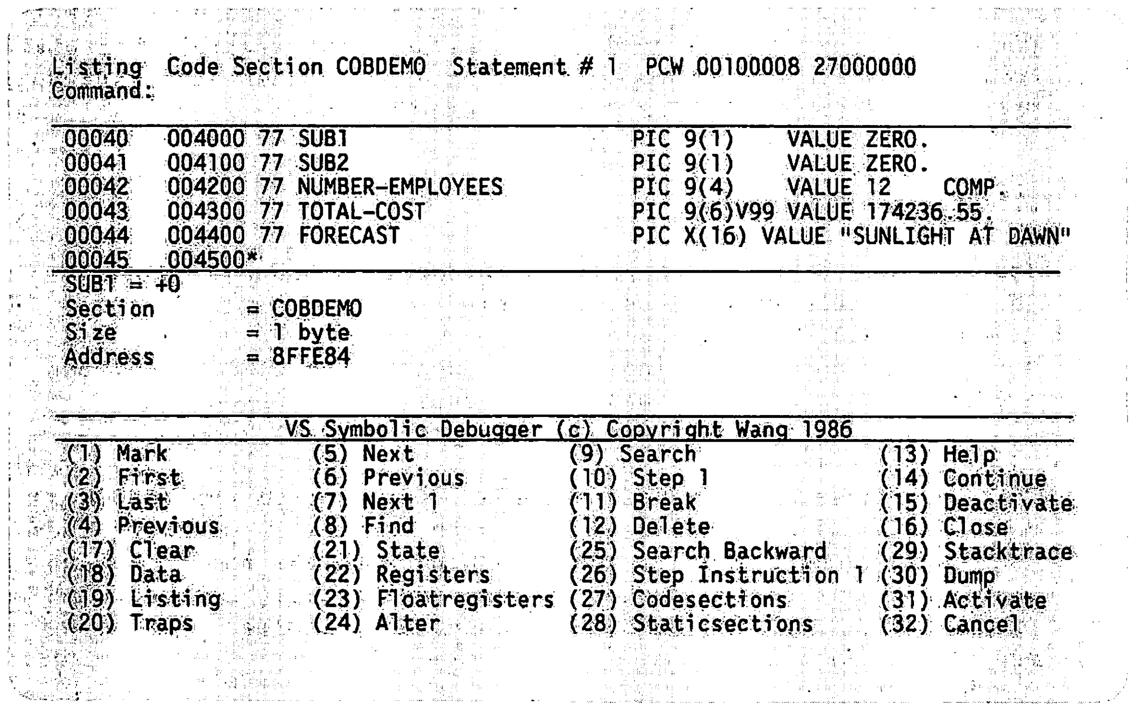
Full Window Format

Full window format provides the maximum possible viewing area. This format is set by executing the FRAME FULL command. If that format is in effect and a window is accessed or created (via the MENU or DISPLAY commands), all other windows are hidden by the new one; i.e., one full window is displayed at a time.

Partial Window Format

Partial window format (the default format) enables you to display multiple windows using the position of the cursor to determine location and size. This format is set by executing the FRAME PARTIAL command.

In partial window format, if the cursor is on the command line and you execute a command to access a window, the accessed window occupies the full window section, the same as for the FRAME FULL setting. However, if the cursor is somewhere in the window section when you enter a command, the accessed window is displayed at the line of the cursor, and extends downward either to the bottom of the window section, or to the top of the next lower window if one exists. Only the upper boundary of a new window is selected by the cursor; the lower boundary is determined automatically. Figure 4-1 shows a sample 3-window display.



```
Listing  Code Section COBDEMO  Statement # 1  PCW 00100008 27000000
Command:
 00040  004000 77 SUB1                     PIC 9(1)     VALUE ZERO.
 00041  004100 77 SUB2                     PIC 9(1)     VALUE ZERO.
 00042  004200 77 NUMBER-EMPLOYEES         PIC 9(4)     VALUE 12     COMP.
 00043  004300 77 TOTAL-COST               PIC 9(6)V99 VALUE 174236.55.
 00044  004400 77 FORECAST                 PIC X(16) VALUE "SUNLIGHT AT DAWN"
 00045  004500*
 SUB1 = +0
 Section      = COBDEMO
 Size         = 1 byte
 Address      = 8FFE84

                 VS Symbolic Debugger (c) Copyright Wang 1986
 (1) Mark          (5) Next            (9) Search              (13) Help
 (2) First         (6) Previous        (10) Step 1             (14) Continue
 (3) Last          (7) Next 1          (11) Break              (15) Deactivate
 (4) Previous      (8) Find            (12) Delete             (16) Close
 (17) Clear        (21) State          (25) Search Backward    (29) Stacktrace
 (18) Data         (22) Registers      (26) Step Instruction 1 (30) Dump
 (19) Listing      (23) Floatregisters (27) Codesections       (31) Activate
 (20) Traps        (24) Alter          (28) Staticsections     (32) Cancel
```

Figure 4-1. Sample Three-Window Display

For reference information about commands for managing windows, refer to Chapter 7.

4.6.2 Window Management Functions

The Debugger enables you to display and manage one or more windows in the Window section. When you display more than one window, each window appears as a separate partial window format. The partial window format is accessed by executing the FRAME PARTIAL command. The number of lines displayed for each window is reduced depending on the number of windows being displayed simultaneously. Note that Debugger windows have no relation to VS Multistation windows.

A maximum of five windows can appear at one time, although the amount of each window that you can display simultaneously is limited by the available space in the Window section. Windows can exist without being displayed but they must be displayed in order to use them interactively.

Each displayed window has a status line unless you disable the status line for a window by executing the STATUS OFF command. In that case, you gain an extra line of display in that window. This does not apply to the topmost window; no information appears where the status line was if you execute STATUS OFF from that window.

4.6.3 Moving Between Windows

You can move directly between the predefined Debugger windows by using the appropriate command as follows:

- To move to the listing window, execute the LISTING command.
- To move to the trap window, execute the TRAPS command.
- To move to the data window, execute the DATA command.

To move between windows sequentially, you execute the WINDOW command. Each window is assigned a number by the system. The Trap Window is assigned to the number 1, the Data Window to the number 2, and the listing window to the number 3. Any additionally created windows (menu or display) are numbered sequentially increasing from 4 in their order of creation.

If you close a window (only the menu window or the display window can be closed) by executing the CLOSE command, the following window (if one exists) takes the number of the closed window.

Once you have accessed the final window in the series, the initial window is redisplayed when you execute the WINDOW command from that window, beginning the sequence again.

You can execute WINDOW with a numeric operand that specifies the number of the window to be brought into view. For example, to access the DATA window, you execute WINDOW 2; to access a window that is fifth in the sequence, you execute WINDOW 5. If WINDOW is executed with no operand, the next higher numbered window is brought into view; if WINDOW is executed from Window 5, Window 1 is brought into view.

4.6.4 Window Context

A collection of information called the window context is associated with each window. The window context consists of the following attributes that are unique to that window:

- Case setting
- Current search string
- Contents of the status display
- Range of marked lines

When you display a file on the display window, certain context information is copied from the last window selected. This information includes the search string, which is modifiable. The case setting is not copied, nor are any line marks.

4.7 DISPLAYING VS FILES

You can display a requested VS file on the display window. To display a file, execute the DISPLAY command followed by the volume, library, and file names of the file you want to display. The Debugger then displays the file on the display window and numbers that window one number greater than the highest window number.

You can display one file at a time only; when you display a subsequent file, the current file is removed. To terminate a file display, you execute the CLOSE command from the display window.

4.8 DISPLAYING PF KEY ASSIGNMENTS

To display the current PF key assignments, execute the MENU command. The Debugger displays the menu window either in its entirety or partially, depending on the status of the FRAME command and the location of the cursor. The menu window lists the command assignments for PF keys 1 through 32. If the menu window is displayed partially, you can scroll the display by executing the NEXT command with the cursor positioned in the menu window.

To remove the menu window, execute the CLOSE command with the menu window as the current window. For conceptual information about the menu window, refer to Section 2.3.6.

4.9 DISPLAYING HELP TEXT

Help text provides on-line reference information about the Debugger. To display Help text, execute the HELP command. The Debugger then accesses the VS INFO utility which displays the Debugger Help menu. The Debugger Help menu enables you to locate desired help information. For conceptual information about Help text, refer to Section 1.8.

4.10 PRINTING DATA

The Debugger enables you to print a hardcopy of specified data. To print a hardcopy of a marked range of lines, position the cursor in the window that contains the marked lines and execute the PRINT command. The Debugger then queues the marked range to the appropriate printer.

To print the entire contents of a window, first execute the CLEAR ALL command to remove any marks, then execute the PRINT command. You can get the same result by marking all lines in the window (execute MARK ALL) and then executing PRINT.

4.11 ASSIGNING COMMANDS TO PF KEYS

To simplify the debugging process, you can assign a single command or string of commands to any PF key except ENTER. When the PF key is subsequently pressed, the assigned commands are executed one at a time, as if they were entered on the command line.

To assign one or more commands to a PF key, execute the ASSIGN command using either of the following methods:

- Enter ASSIGN followed by the command(s) you want to assign, and then press the PF key to which the commands are to be assigned.

- Enter ASSIGN followed by the number of the PF key you want to assign the commands to, and the commands to be assigned. Then, press ENTER.

For example, to assign the CLEAR command to PF1 by the latter method, you execute the following command:

```
ASSIGN 1 CLEAR
```

If you are assigning more than one command, follow each command by a semicolon (you can optionally include spaces). For example, to assign a series of commands to position the cursor at column 20 of line 40 to PF2, you execute the following command:

```
ASSIGN 2 GOTO 40; COLUMN 20
```

Note: *Validation of the assigned command string occurs during the execution of the string, not at the time you assign the string. For example, if you execute ASSIGN 1 MAKK, the Debugger assigns the command string MAKK to PF1 even though MAKK is not a valid command. When you execute PF1, the Debugger informs you that the command string (MAKK) is invalid.*

For conceptual information about assigning commands to PF keys, refer to Section 2.5.1.

4.12 DEFINING MACROS

To further simplify the debugging process, you can assign a single command or string of commands to a macro. When the macro is subsequently executed, the assigned commands are executed one at a time, as if they were entered on the command line.

To define a macro, enter DEFINE followed by the name of the macro and the command(s) you want to assign. Then, press ENTER. For example, to define a macro called MYTRAPS that sets a BREAK trap on line 48 of the subject program and sets a MODTRAP trap on variable SUB1, you enter the following information, then press ENTER:

```
DEFINE MYTRAPS BREAK 48; MODTRAP VARIABLE SUB1
```

Whenever you execute MYTRAPS, the string of commands associated with that macro is executed. For conceptual information about defining macros, refer to Section 2.6.

Note: *Validation of the defined command string occurs during the execution of the string, not at the time you assign the string. For example, if you execute DEFINE M5 MAKK 5, the Debugger defines the macro M5 as the command string MAKK 5, even though MAKK 5 is not a valid command. When you execute M5, the Debugger informs you that the command string (MAKK 5) is invalid.*

CHAPTER 5
USING DEBUGGING COMMANDS

This chapter describes how to use various debugging commands to perform common debugging actions. In addition, it shows the results of those actions when they are performed against a sample subject program.

The objective of this chapter is to provide information about using the appropriate debugging commands and associated operands to perform desired actions. Familiarity with the activities described will allow you to successfully understand and use the reference debugging command descriptions in Chapter 7. Those descriptions enable you to perform more sophisticated debugging operations.

This chapter does not describe each debugging command and does not give examples of each possible use of command syntax. Sufficient information is provided to give a general description of the command being referenced. In addition, this chapter does not specify which method to use in executing the command; you can execute commands by entering them on the command line or by pressing the appropriate PF key.

5.1 SAMPLE SUBJECT PROGRAM

The COBOL program shown in Figure 5-1 is used as the sample subject program for debugging commands described in this chapter. The first column of values in Figure 5-1 contains the compiler-generated statement numbers that the Debugger uses. The second column of values contains sequence line numbers within the text of the program; the sequence line numbers have no significance to the Debugger.

```
000100 IDENTIFICATION DIVISION.
000200
000300 PROGRAM-ID.   COBDEMO.
000400 AUTHOR.       Bill Johnson.
000500 DATE-WRITTEN. 01/31/XX.
000600*
000700 ENVIRONMENT DIVISION.
000800
000900 INPUT-OUTPUT SECTION.
001000 FILE-CONTROL.
001100    SELECT THE-WORKSTATION
001200        ASSIGN TO "WSFILE",    "DISPLAY"
001300        ACCESS MODE IS RANDOM.
001400*
001500 DATA DIVISION.
001600 FILE SECTION.
001700 FD THE-WORKSTATION
001800    LABEL RECORDS ARE OMITTED.
001900 01 CRTREC                       PIC X(1924).
002000
002100 WORKING-STORAGE SECTION.
002200 01 TWO-OCCURS-LEVELS            USAGE IS DISPLAY-WS.
002300    03 FILLER                    PIC X(45)
002400           ROW 5  COLUMN  24
002500       VALUE IS "TABLE DISPLAY: 3 ROWS * 6 COLUMNS".
002600    03 FILLER      OCCURS 3 TIMES    ROW 7.
002700       05 FILLER   OCCURS 6 TIMES    ROW 7   COLUMN 8   PIC X(10)
002800          SOURCE IS LEVEL-2 OBJECT IS LEVEL-2.
002900
003000 01 TWO-LEVEL-TABLE.
003100    03 LEVEL-1     OCCURS 3 TIMES.
003200       05 LEVEL-2  OCCURS 6 TIMES.
003300          07 TABLE-ENTRY.
003400             09 FILLER        PIC X(6).
003500             09 FIRST-INDEX   PIC 9(1).
003600             09 COMMA-LITERAL PIC X(1).
003700             09 SECOND-INDEX  PIC 9(1).
003800             09 RIGHT-PAREN   PIC X(1).
003900
004000 77 SUB1                         PIC 9(1)    VALUE ZERO.
004100 77 SUB2                         PIC 9(1)    VALUE ZERO.
004200 77 NUMBER-EMPLOYEES             PIC 9(4)    VALUE 12    COMP.
004300 77 TOTAL-COST                   PIC 9(6)V99 VALUE 174236.55.
004400 77 FORECAST                     PIC X(16) VALUE "SUNLIGHT AT DAWN".
004500*
004600 PROCEDURE DIVISION.
004700 DISPLAYIT.
004800    PERFORM INIT1 VARYING SUB1 FROM 1 BY 1 UNTIL SUB1 > 3.
004900    MOVE 372 TO NUMBER-EMPLOYEES.
005000    MOVE 987654.33 TO TOTAL-COST.
```

Figure 5-1. Sample Subject Program Listing

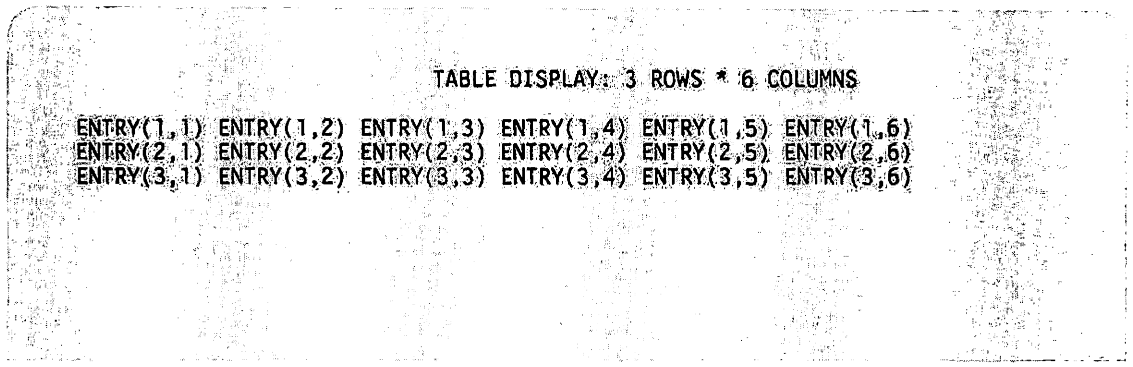
```
00051  005100     MOVE "SLEET BY TONIGHT" TO FORECAST.
00052  005200     DISPLAY AND READ TWO-OCCURS-LEVELS ON THE-WORKSTATION.
00053  005300     STOP RUN.
00054  005400  INIT1.
00055  005500     PERFORM INIT-TABLE VARYING SUB2 FROM 1 BY 1 UNTIL SUB2 > 6.
00056  005600  INIT-TABLE.
00057  005700     MOVE "ENTRY(" TO TABLE-ENTRY (SUB1, SUB2).
00058  005800     MOVE SUB1     TO FIRST-INDEX (SUB1, SUB2).
00059  005900     MOVE ","      TO COMMA-LITERAL (SUB1, SUB2).
00060  006000     MOVE SUB2     TO SECOND-INDEX (SUB1, SUB2).
00061  006100     MOVE ")"      TO RIGHT-PAREN (SUB1, SUB2).
```

Figure 5-1. Sample Subject Program Listing (continued)

The sample subject program produces the screen shown in Figure 5-2. That screen consists of the screen title, starting on row 5, column 24, and a 2-dimensional table (3 rows by 6 columns) starting on row 7, column 8. Each entry is 10 bytes in length and contains the value "ENTRY(m,n)", where m is the row number of the entry and n is the column number.

The source field is LEVEL-2, an entry of the table (TWO-LEVEL-TABLE) with two levels of OCCURS. The first level (LEVEL-1) occurs three times and indicates the number of repetitions of fields down the screen. The second level (LEVEL-2) occurs six times and indicates the number of repetitions of fields across the screen.

In the PROCEDURE DIVISION, the paragraph INIT-TABLE, coded on statements 56 through 61, is performed 18 times, initializing each entry to the value "ENTRY(m,n)" using two nestings of the PERFORM VARYING statement. The DISPLAY AND READ of the USAGE IS DISPLAY-WS screen format TWO-OCCURS-LEVELS, coded on statement 52, maps the table onto the screen (six entries across and three entries down), producing the screen shown in Figure 5-2.



```
                        TABLE DISPLAY: 3 ROWS * 6 COLUMNS

  ENTRY(1,1) ENTRY(1,2) ENTRY(1,3) ENTRY(1,4) ENTRY(1,5) ENTRY(1,6)
  ENTRY(2,1) ENTRY(2,2) ENTRY(2,3) ENTRY(2,4) ENTRY(2,5) ENTRY(2,6)
  ENTRY(3,1) ENTRY(3,2) ENTRY(3,3) ENTRY(3,4) ENTRY(3,5) ENTRY(3,6)
```

Figure 5-2. Output From Sample Subject Program

5.2 PERFORMING DEBUGGING ACTIONS

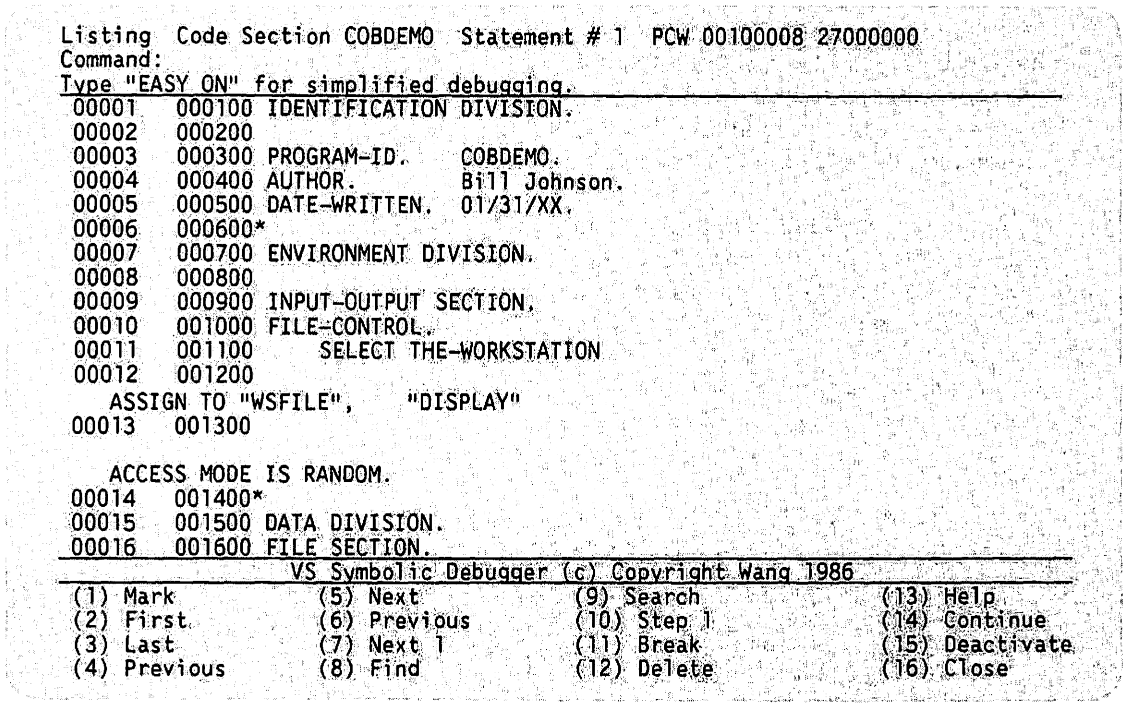
Debugging consists mainly of the following tasks:

- Setting traps
- Monitoring the values of specified program entities
- Displaying program information

5.2.1 Setting Traps

Traps halt execution of the subject program when certain conditions occur. This section describes how to set five types of debugging traps. The results of setting these traps in the trap window is shown in Figure 5-4.

The initial window display (Figure 5-3) shown on the Debugger Workstation screen appears as it would if you loaded the program for interactive debugging. The display on the listing window begins from the first entry point of the subject program. The menu window occupies the last four lines of the Window Section.



```
Listing  Code Section COBDEMO  Statement # 1  PCW 00100008 27000000
Command:
Type "EASY ON" for simplified debugging.
 00001   000100 IDENTIFICATION DIVISION.
 00002   000200
 00003   000300 PROGRAM-ID.     COBDEMO.
 00004   000400 AUTHOR.         Bill Johnson.
 00005   000500 DATE-WRITTEN.   01/31/XX.
 00006   000600*
 00007   000700 ENVIRONMENT DIVISION.
 00008   000800
 00009   000900 INPUT-OUTPUT SECTION.
 00010   001000 FILE-CONTROL.
 00011   001100     SELECT THE-WORKSTATION
 00012   001200
    ASSIGN TO "WSFILE",    "DISPLAY"
 00013   001300

    ACCESS MODE IS RANDOM.
 00014   001400*
 00015   001500 DATA DIVISION.
 00016   001600 FILE SECTION.
                   VS Symbolic Debugger (c) Copyright Wang 1986
 (1) Mark          (5) Next          (9) Search        (13) Help
 (2) First         (6) Previous      (10) Step 1       (14) Continue
 (3) Last          (7) Next 1        (11) Break        (15) Deactivate
 (4) Previous      (8) Find          (12) Delete       (16) Close
```

Figure 5-3. Initial Window Display

Setting a Break Trap

The BREAK command sets a trap at a specified location in the subject program. That location can be a program statement, address, or offset. For reference information about BREAK, refer to Chapter 7.

You may want to set a BREAK trap in the subject program at statement 48. That trap is set by executing the following command:

```
BREAK 48
```

where 48 specifies the statement number.

The Debugger enters the following information in the trap window:

```
Status      Count      Hits     Type      Operands
Active          1         0     Break     48 COBDEMO
```

Note that statement numbers appear as they are displayed in the listing file (Figure 5-3).

You can set the same trap by positioning the cursor at any location on statement 48 and executing BREAK with no operand. While this trap is active, each time the program encounters the first instruction of that statement, the trap is taken and control is transferred to the Debugger.

Setting an INSIDE Trap

The INSIDE command sets traps at all points inside of a specified range or a marked range, including the boundary statements. The range is indicated by program statements, addresses, or offsets. For reference information about INSIDE, refer to Chapter 7.

You may want to set INSIDE traps in the subject program from statement 57 to statement 59. That series of traps is set by executing the following command:

```
INSIDE 57 59
```

where 57 and 59 specify the boundary statement numbers.

The Debugger enters the following information in the trap window:

```
Status      Count      Hits     Type       Operands
Active          1         0     Inside     57 59 COBDEMO
```

You can set the same trap by marking the appropriate lines and executing INSIDE with no operand. While this trap is active, each time the program encounters any instruction within the specified range, the trap is taken and control is transferred to the Debugger.

Setting an OUTSIDE Trap

The OUTSIDE command sets traps at all points outside of a specified range or a marked range. The range is indicated by program statements, addresses, or offsets. For reference information about OUTSIDE, refer to Chapter 7.

You may want to set OUTSIDE traps in the subject program from statement 57 to statement 61. That series of traps is set by first executing the MARK command (twice) to mark that range of statements, and then executing OUTSIDE with no operand. The Debugger enters the following information in the trap window:

```
Status      Count       Hits      Type        Operands
Active      1           0         Outside     57 61 COBDEMO
```

You can set the same trap by executing OUTSIDE 57 61. While this trap is active, each time the program encounters any instruction outside of the specified range, the trap is taken and control is transferred to the Debugger.

Setting a MODTRAP Trap

The MODTRAP command sets a trap that monitors a specified program address, variable, or general register. You indicate the entity to be monitored by specifying the appropriate operand. Each time the value of the operand changes (subject to the relop clause if included with the REGISTER operand), the trap is taken. For reference information about MODTRAP, refer to Chapter 7.

You may want to trap the subject program each time the value of variable SUB1 changes. That trap is set by executing MODTRAP as follows:

```
MODTRAP VARIABLE SUB1
```

The Debugger enters the following information in the trap window:

```
Status      Count       Hits      Type        Operands
Active          1          0      Modtrap     Variable SUB1
```

While this trap is active, each time the value of variable SUB1 changes, the trap is taken and control is transferred to the Debugger.

Setting an OPCODE Trap

The OPCODE command sets a trap that is taken at any execution of a specified machine instruction. That instruction can be a mnemonic name or a hexadecimal value. For reference information about OPCODE, refer to Chapter 7.

You may want to set an OPCODE trap in the subject program on machine instruction JSCI. That trap is set by executing either of the following commands:

```
OPCODE JSCI
```

or

```
OPCODE HEX 61
```

If the first form is used, the Debugger enters the following information in the trap window:

```
Status      Count      Hits     Type        Operands
Active          1         0     Opcode      JSCI
```

While this trap is active, each time the appropriate program opcode is encountered, the trap is taken and control is transferred to the Debugger.

5.2.2 Monitoring the Values of Specified Program Variables

When program execution is interrupted for a trap, it is often desirable to examine the value of program variables at that point. The VARIABLE command displays in the data window, program variables that you specify, and monitors the value of each variable. For reference information about VARIABLE, refer to Chapter 7.

You may want to monitor variables SUB1 and SUB2 in the subject program. Those variables are entered in the data window, each as a separate entry, by executing the following commands:

```
VARIABLE SUB1; VARIABLE SUB2
```

The Debugger enters the following information in the data window:

```
SUB1 = +0
Section       = COBDEMO

SUB2 = +0
Section       = COBDEMO
```

You could perform the same function by positioning the cursor at the first character of the desired variable in any window, and executing VARIABLE with no operand for each entry.

All contents of the data window remain present for the current session unless deleted. Each variable present always contains its current value.

Displaying Additional Attributes With Variables

In addition to displaying the most current value of a variable in the data window, you can specify a number of attributes to be appended to the display for each variable. Those attributes apply only to the corresponding variable, and are set by executing the ATTRIBUTES command and any of the following operands:

OFF -- Disables the display of attributes.

SECTION -- Displays the code section of the variable.

SIZE -- Displays the number of storage bytes that the variable occupies.

ADDRESS -- Displays the starting address of the variable.

TYPE -- Displays the data type of the variable (integer, character, etc.).

STORAGE -- Displays the storage class of the variable (parameter, static, etc.).

You may want to show the section and data type of each subsequent variable entered in the data window. This is done by executing the following command:

```
ATTRIBUTES SECTION TYPE
```

The Debugger appends the section and data type of each subsequent variable entered in the Data Window to the end of the entry.

5.2.3 Displaying Program Information

In addition to displaying the current values of program variables in the data window, you can display parts of program memory. That topic is described in this section. Besides program memory, you can display information about the following program items:

- RPG indicators (INDICATORS command)
- General registers (REGISTERS command)
- Floating-point registers (FLOATREGISTERS command)
- Program state (STATE command)
- Active JSI, LINK, and SVC instructions (STACKTRACE command)
- The last runtime diagnostic message (DIAGNOSTIC command)
- Subprograms that exist in the current section (PROCEDURES command)
- All code sections (CODESECTIONS command)
- All static sections (STATICSECTIONS command)

For reference information about the corresponding commands, refer to Chapter 7.

Displaying Program Memory

To determine the cause of program failures, you may have to display portions of program memory. To display a part of the subject program's memory, you execute the MEMORY command followed by the desired operands. For reference information about MEMORY, refer to Chapter 7.

You may want to display the value at address 100008 in the subject program's memory. That value is displayed by executing the following command:

```
MEMORY 100008 48
```

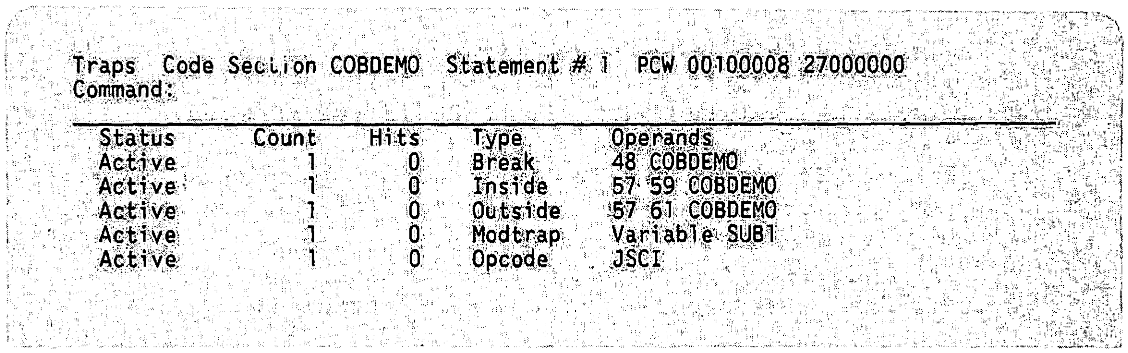
where 100008 is the specified address, and 48 is the decimal length.

The following entry is a sample of the information that could appear on the data window as a result:

```
Memory 100008 48 =
100008  0000   77F00060 71C011AC 5830C084 0D3018BE   "w..°q...X0...0.."
100018  0010   5AB0C000 B0000000 9680F040 8500000C   "Z..........@...."
100028  0020   50F0F008 18AF5010 A0005830 B0CC5820   "P.....P...X0..X "
```

5.3 MODIFYING THE TRAP WINDOW

Figure 5-4 shows the results of the trap setting operations described in Section 5.2.1 in the trap window.



```
Traps  Code Section COBDEMO  Statement # 1  PCW 00100008 27000000
Command:
_____________________________________________________________________
   Status      Count    Hits   Type      Operands
   Active          1       0   Break     48 COBDEMO
   Active          1       0   Inside    57 59 COBDEMO
   Active          1       0   Outside   57 61 COBDEMO
   Active          1       0   Modtrap   Variable SUB1
   Active          1       0   Opcode    JSCI
```

Figure 5-4. Results in Trap Window

The columns in the trap window contain the following information:

Status -- The status of the trap entry. The status can be one of the following items:

- Active -- Indicates a trap that is operational and has not been taken.

- Taken -- Indicates a trap that has just been taken.

- Inactive -- Indicates a trap that is not operational.

Each newly created trap is displayed with an Active status. You can change the status of desired traps by executing the ACTIVATE command or the DEACTIVATE command. For information about how to execute those commands, refer to Section 5.3.1.

Count -- The count associated with the trap. The count is the number of times that the condition for the trap is to be encountered before the trap is taken. The default (1) interrupts execution the first time the specified trap condition is met. Each newly created trap is assigned a count of 1. For information about managing the count of a trap, refer to Section 5.3.2.

Note: *The counts for INSIDE, OUTSIDE, and MODTRAP traps are set at 1 and cannot be changed with the COUNT command.*

Hits -- The number of times the trap has been encountered by the subject program. When the Hits value equals the Count value, the trap is taken.

Type -- The type of trap (BREAK, INSIDE, etc.).

Operands -- The operands associated with the trap.

Note: *If the link level for the subject program changes, the Debugger displays an asterisk (*) before each trap that corresponds to a different link level. Traps with an asterisk are not changed in any way but are no longer operational until the appropriate link level is again in effect.*

You can perform the following types of modifications on existing traps:

- Activate and deactivate traps
- Change the count of a trap
- Delete traps

5.3.1 Activating and Deactivating Traps

Traps can be active or inactive. An active trap (with Active status) is an existing trap that is taken when all the conditions associated with it are satisfied in the subject program. A trap that has been taken (with Taken status) remains active. An inactive trap (with Inactive status) is an existing trap that does not operate against the subject program even if all its conditions are satisfied. It is useful to make certain traps inactive when they are not needed in debugging, rather than to delete them. This enables you to quickly use those traps again by making them active, rather than recreating each trap.

Traps are activated by executing the ACTIVATE command; traps are deactivated by executing the DEACTIVATE command. The following rules apply when you execute ACTIVATE or DEACTIVATE:

- If any traps are marked, those traps are activated or deactivated.

- If no traps are marked, the trap at which the cursor is positioned is activated or deactivated.

- If no traps are marked and the cursor is outside of the trap window (assuming the trap window is the current window), the last trap in the list is activated or deactivated. If there are no traps in the trap window, an error message is displayed.

Within the trap window shown in Figure 5-4, you may want to change the status of the INSIDE and OUTSIDE traps to Inactive. To change the status, first mark both traps using the MARK command, and then execute DEACTIVATE. The status for those traps is changed as follows:

```
Status      Count      Hits     Type        Operands
Active          1         0     Break       48 COBDEMO
Inactive        1         0     Inside      57 59 COBDEMO
Inactive        1         0     Outside     57 61 COBDEMO
Active          1         0     Modtrap     Variable SUB1
Active          1         0     Opcode      JSCI
```

You can perform the same function by positioning the cursor at the desired trap and entering DEACTIVATE. The Debugger will not take the inactive traps until you activate them again.

To reactivate those traps, follow the same process, but execute ACTIVATE instead of DEACTIVATE. The Debugger removes the Inactive status from the specified traps. The Debugger takes a reactivated trap when all conditions associated with the trap have been satisfied.

For reference information about activating and deactivating traps, refer to Chapter 7.

5.3.2 Managing the Count

Every trap has an associated count. The count is the number of times that the conditions for a trap must be satisfied before the trap is taken. For example, if the count for a trap is 3, the conditions for the trap must be satisfied three times before the trap is taken.

The second column in the trap window, titled Count, displays the count of each existing trap. All traps are initially set with a default count of 1, meaning that the trap is taken on the first valid instance after the program is resumed.

The third column in the trap window, titled Hits, indicates the number of times the corresponding trap has been encountered by the subject program. To determine how many more times a trap must be encountered before it is taken, you subtract the Hits value from the Count value. The Hits value is 0 when you create a trap, and is reset to 0 after the associated trap is taken.

For example, assume you set a BREAK trap on statement 48 in the subject program and set the Count field for that trap to 5. Since the trap has not yet been encountered by the subject program, the value displayed in the Hits field is 0.

When the subject program is resumed, statement 48 is executed twice before the Debugger resumes control again for some reason such as a subsequent trap taken. The Count field remains at 5 but the Hits field now contains a 2, since the statement has been encountered twice since the trap was set.

Changing the Count

To change the count of a trap, you indicate the desired trap(s) using the cursor or marking, and execute COUNT followed by the desired operand. The operand is the number to which you want to change the count. Note that COUNT has no effect on INSIDE, OUTSIDE, or MODTRAP traps. COUNT operates on traps according to the following rules:

- If no traps are marked, COUNT operates on the trap at which the cursor is positioned.
- If any traps are marked, those trap counts are changed.
- If no traps are marked and the cursor is positioned outside of the trap window, COUNT operates on the last trap in the list.

You may want to change the count of the BREAK trap to 3. You do this by positioning the cursor at any location on the BREAK trap (or marking the trap) and then executing COUNT 3. The count is changed as shown below:

```
Status      Count      Hits     Type        Operands
Active          3         0     Break       48 COBDEMO
Inactive        1         0     Inside      57 59 COBDEMO
Inactive        1         0     Outside     57 61 COBDEMO
Active          1         0     Modtrap     Variable SUB1
Active          1         0     Opcode      JSCI
```

The Hits field is reset to zero whenever the Count field changes.

5.3.3 Deleting One or More Traps

To delete one or more traps from the trap window, execute the DELETE command. For traps, DELETE operates according to the following rules:

- If any traps are marked, those traps are deleted.
- If no traps are marked, the trap at which the cursor is positioned is deleted.
- If no traps are marked and the cursor is outside of the trap window (assuming the trap window is the current window) the last trap is deleted.

Within the trap window displayed in Figure 5-4, you may want to delete the INSIDE trap. This is done by positioning the cursor at that trap and executing DELETE. An alternate way of performing the same function is to first mark the trap and then execute DELETE. The Debugger removes the INSIDE trap, and modifies the trap window as follows:

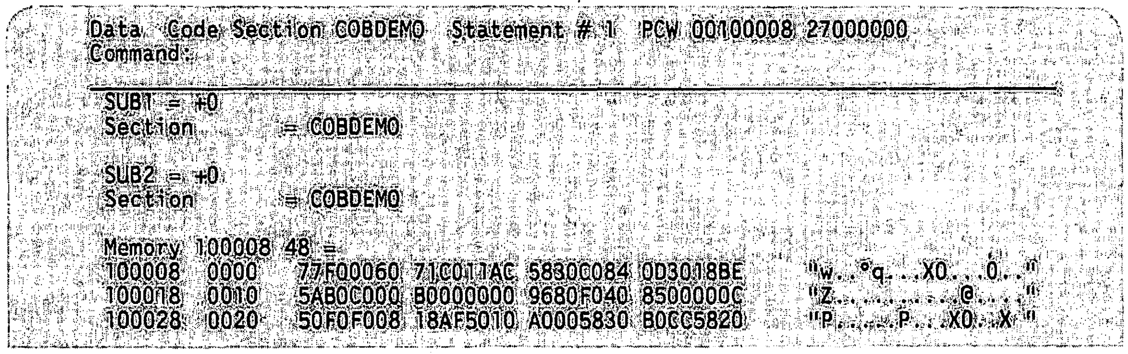
```
Status      Count     Hits    Type        Operands
Active          3        0    Break       48 COBDEMO
Inactive        1        0    Outside     57 61 COBDEMO
Active          1        0    Modtrap     Variable SUB1
Active          1        0    Opcode      JSCI
```

Later, you may want to delete all traps and create new ones. This is done by first marking all traps by executing the MARK ALL command, and then executing DELETE. The Debugger then removes all traps from the trap window.

For reference information about deleting traps, refer to Chapter 7.

5.4 MODIFYING THE DATA WINDOW

The results of the variable and data display operations described in Sections 5.2.2 and 5.2.3 are shown in the data window in Figure 5-5.



```
Data  Code Section COBDEMO  Statement # 1  PCW 00100008 27000000
Command:
 SUB1 = +0
 Section         = COBDEMO

 SUB2 = +0
 Section         = COBDEMO

 Memory 100008 48 =
 100008  0000   77F00060 71C011AC 5830C084 0D3018BE     "w..°q...XO...0.."
 100018  0010   5AB0C000 B0000000 9680F040 8500000C     "Z..........@...."
 100028  0020   50F0F008 18AF5010 A0005830 B0CC5820     "P.....P...XO..X "
```

Figure 5-5. Results in Data Window

You can perform the following types of operations on information that appears in the data window:

- Modify the values of variables in modifiable program memory (if not in privileged code)
- Modify registers, program state, etc. (if not in privileged code)
- Delete entries

5.4.1 Modifying Program Values

Several Debugging commands display the current values of specified modifiable program entities (variables, registers, portions of memory, portions of the program state). These commands include VARIABLE, REGISTERS, FLOATREGISTERS, STATE, and MEMORY.

Each command displays the current value of the specified program entry in the data window, and monitors the value of each entry. The values for all displayed entries are always the current values. To change the value of a desired entry, enter the new value and execute the ALTER command.

In the data window displayed in Figure 5-5, you may want to change the value of the variable SUB1 to 3 in program memory. To change the value, enter 3 over the previous value and execute ALTER. The Debugger then locates the memory associated with the variable and changes it to reflect the new value. You can modify only one entry at a time.

For reference information about modifying program values (the ALTER command), refer to Chapter 7.

5.4.2 Deleting an Entry in the Data Window

You can delete one entry at a time from the data window. To delete an entry, indicate the entry (via the cursor or marking) and execute the DELETE command. DELETE operates according to the following rules in the data window:

- If an entry is marked, that entry is deleted. If a mark appears at any line within an entry, the entire entry is deleted.
- If no entry is marked, the entire entry at which the cursor is positioned is deleted.
- If no entry is marked, and the cursor is outside of the data window (assuming the data window is the current window), the last entry is deleted.

5.5 DISPLAYING A PARTIAL WINDOW FORMAT

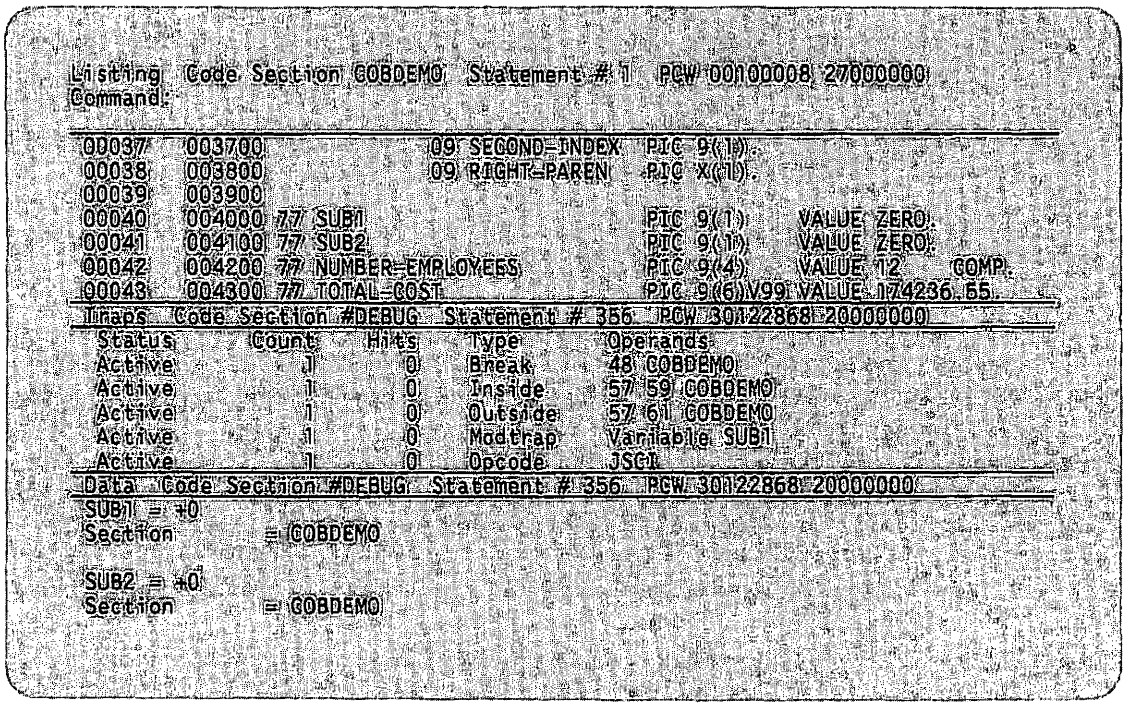
By displaying a partial window format for two or more windows, you can manage several windows at once. Partial window format is set by executing the FRAME PARTIAL command and using the position of the cursor to determine window location and size.

For example, you may want to display the listing window, the data window, and the trap window together, starting from an original full window display of the listing window.

First, execute the FRAME PARTIAL command. Then, position the cursor within the listing window at the location at which you want to begin display of the trap window, and execute the TRAPS command. The display of the listing window is terminated at that location; the trap window is displayed from that location to the bottom of the Window section.

Next, position the cursor at the location in the trap window at which you want to begin display of the data window. Then, execute the DATA command. The display of the trap window is terminated at that location; the data window is displayed from that location to the bottom line of the Window section.

The results of the partial window format display described in this section are shown in Figure 5-6.



```
Listing  Code Section COBDEMO  Statement # 1  PCW 00100008 27000000
Command:

00037  003700                 09 SECOND-INDEX  PIC 9(1).
00038  003800                 09 RIGHT-PAREN   PIC X(1).
00039  003900
00040  004000 77 SUB1                          PIC 9(1)     VALUE ZERO.
00041  004100 77 SUB2                          PIC 9(1)     VALUE ZERO.
00042  004200 77 NUMBER-EMPLOYEES              PIC 9(4)     VALUE 12      COMP.
00043  004300 77 TOTAL-COST                    PIC 9(6)V99 VALUE 174236.65.
Traps  Code Section #DEBUG  Statement # 356  PCW 30122868 20000000
  Status        Count   Hits   Type        Operands
  Active            1      0   Break       48 COBDEMO
  Active            1      0   Inside      57 59 COBDEMO
  Active            1      0   Outside     57 61 COBDEMO
  Active            1      0   Modtrap     Variable SUB1
  Active            1      0   Opcode      JSCI
Data  Code Section #DEBUG  Statement # 356  PCW 30122868 20000000
 SUB1 = +0
 Section          = COBDEMO

 SUB2 = +0
 Section          = COBDEMO
```

Figure 5-6. Partial Window Format Display

The following commands, when executed, generate the window display shown in Figure 5-6:

```
LISTING
FRAME PARTIAL
CURSOR DOWN 8 (or position the cursor to Row 8)
TRAPS
CURSOR DOWN 15 (or position the cursor to Row 15)
DATA
```

An alternative method of specifying the above commands is to create and execute the following macro:

```
DEFINE WINDDISP LISTING;FRAME PARTIAL;CURSOR DOWN 8;TRAPS;CURSOR DOWN
15;DATA
```

For conceptual information about the partial window format, refer to Section 4.6.1; for reference information about the partial window format, refer to Chapter 7.

CHAPTER 6
DEBUGGING WITH SHARED SUBROUTINE LIBRARIES AND MSMAPed FILES

6.1 INTRODUCTION

Some of your programs may contain references to shared subroutine libraries (SSLs) or to files mapped in via the MSMAP system routine. This chapter describes how to debug those SSLs and MSMAPed files along with the program that references them.

Shared subroutine libraries (SSLs) are handled in Version 1.05 (or greater) of the debugger, which requires Operating System 7.20.

6.2 CHAPTER GLOSSARY

To understand this chapter, you should be familiar with the following terms:

Program Name: A name that refers to one of three kinds of files:

- The main program.
- A shared subroutine library (SSL). An SSL is a single VS file containing a collection of subroutines that can be shared by multiple programs running concurrently. For more information about SSLs, refer to the *VS Linker Reference*.
- An MSMAPed file. An MSMAPed file is one that is loaded by the main program via an MSMAP call. For more information about MSMAPed files, refer to the *Operating System Services Reference*.

Note: *The Debugger requires that every mapped file have a unique path name. Thus, if you map the same file into your main program twice, the Debugger recognizes one or the other instance of the mapped program, but not both.*

Alias: A logical name of up to 40 characters that is assigned to a shared subroutine library. The system security administrator assigns an alias to each SSL using the SSL utility, which is described in the *VS System Administrator's Reference*.

Context: The code section that is currently displayed. During a debugging session, the context defaults to the section containing the address where execution has paused, such as at a breakpoint. You can change the context with the SECTION command.

6.3 DEBUGGING SSLs AND MSMAPed FILES

You can use the Debugger to debug not only your main program but also any SSLs or MSMAPed files that your program references. You can, for example

- Display a section of code in an SSL or MSMAPed file with the SECTION command
- Display parts of the memory of an SSL or MSMAPed file with the MEMORY command
- Display lists of static sections, code sections, or subroutines in an SSL or MSMAPed file with the STATICSECTIONS, CODESECTIONS, or PROCEDURES command
- Set a trap in an SSL or MSMAPed file with the BREAK, INSIDE, MODTRAP, or OUTSIDE command
- Establish and access an alternate program file for an SSL or MSMAPed file with the DEBUGFILE command

6.3.1 How to Specify Section Names as Command Operands

To specify a section name as a command operand, use the syntax

```
[program name/]section name
```

For example

```
MATHLIB/#COSINE
```

Note that the program name prefix is optional. If you omit the program name, the Debugger uses the name of the program in the current context by default. *Thus, to refer to a section that is not in the program containing the current context, you must include the program name prefix.*

If, for example, the current context is within an SSL, and you wish to refer to a section name in the main program, you must include the main program name prefix in the section name specification.

Conversely, if the current context is within the main program, and you wish to refer to a section name in an SSL, you must include the SSL program name prefix in the section name specification. In this case, you can use either the SSL's file name or its alias. Both the file name and the corresponding alias appear on the "Programs" data display.

6.3.2 How to Specify the Program Name Prefix

A program name specified as a path name consists of the volume and library names as well as the file name. You can usually omit the volume and library names, however, specifying only the file name. You must include library (or library and volume) names only if duplicate names exist.

Consider, for example, the following three files:

- LIBPAK.MYLIBS.STRLIB
- LIBPAK.MYLIBS.MATHLIB
- LIBPAK.OURLIBS.MATHLIB

You can specify the first file as STRLIB alone. However, you must specify the second and third files as MYLIBS.MATHLIB and OURLIBS.MATHLIB, respectively.

6.4 THE DISPLAY OF PROGRAM NAMES

In its status line and data window displays, the Debugger distinguishes between SSLs or MSMAPed files and the main program file in the following way:

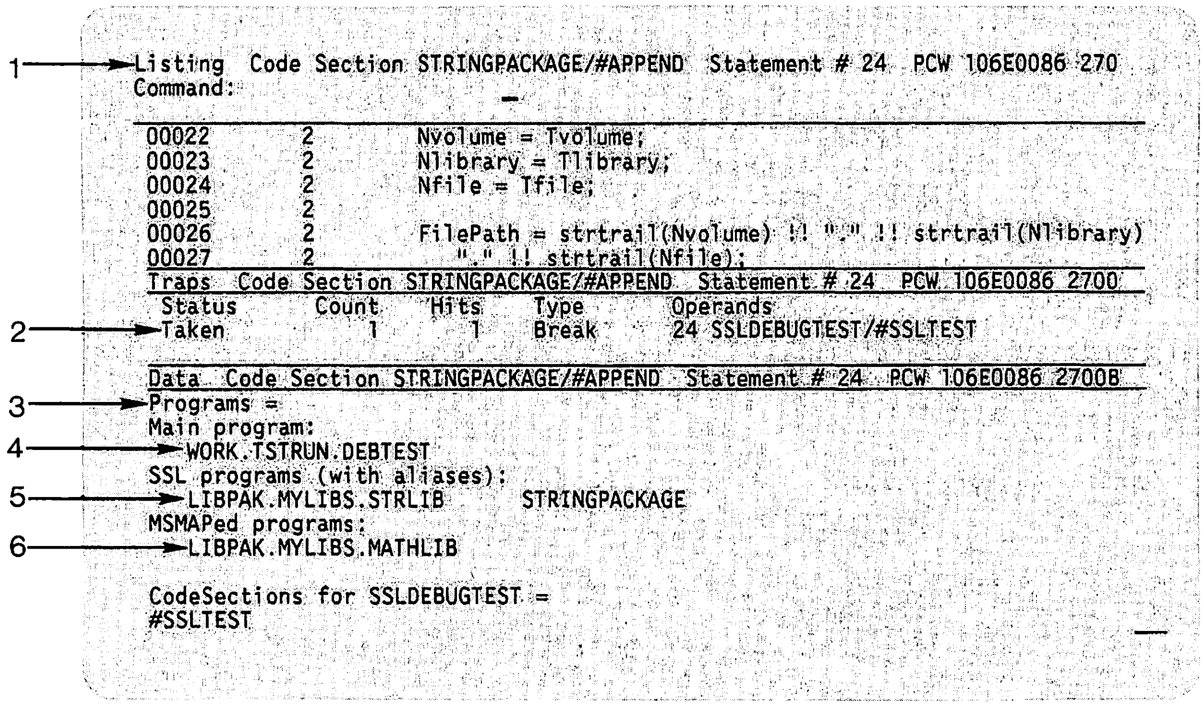
- When the context is within an SSL or an MSMAPed file, the program name is *always* displayed.

- When the context is in the main program, the program name *never* appears. Even if you enter the main program name, it is not displayed on the screen.

6.5 DEBUGGING A PROGRAM THAT REFERENCES AN SSL: TWO SAMPLE DISPLAYS

This section comprises two sample data window displays, Figures 6-1 and 6-2, that demonstrate how the Debugger interprets commands when the subject program references an SSL. In Figures 6-1 and 6-2, the numbers to the left of the screens refer to the numbered explanatory paragraphs that follow each screen.

When a programmer executes the PROGRAMS command, followed by the CODESECTIONS command, the Debugger creates in the data window a display similar to that in Figure 6-1.



```
Listing  Code Section STRINGPACKAGE/#APPEND  Statement # 24  PCW 106E0086 270
Command:
 00022        2         Nvolume = Tvolume;
 00023        2         Nlibrary = Tlibrary;
 00024        2         Nfile = Tfile;
 00025        2
 00026        2         FilePath = strtrail(Nvolume) !! "." !! strtrail(Nlibrary)
 00027        2            "." !! strtrail(Nfile);
 Traps  Code Section STRINGPACKAGE/#APPEND  Statement # 24  PCW 106E0086 2700
  Status      Count   Hits   Type      Operands
  Taken           1      1   Break     24 SSLDEBUGTEST/#SSLTEST

 Data  Code Section STRINGPACKAGE/#APPEND  Statement # 24  PCW 106E0086 2700B
 Programs =
 Main program:
    WORK.TSTRUN.DEBTEST
 SSL programs (with aliases):
    LIBPAK.MYLIBS.STRLIB        STRINGPACKAGE
 MSMAPed programs:
    LIBPAK.MYLIBS.MATHLIB

 CodeSections for SSLDEBUGTEST =
 #SSLTEST
```

Figure 6-1. Example: Data Displayed by the PROGRAMS Command

Explanation of Figure 6-1

1. The Debugger was entered while the user program was executing in an SSL. Thus, the section name in the status line includes the program name prefix (which in this case is an alias).

2. The programmer set a trap at line 24 with the BREAK command. Since the breakpoint is within an SSL, the section name includes the program name prefix.

3. The PROGRAMS command displays the main program name and all program names associated with it.

4. The main program name is always displayed, whether or not SSLs are linked in.

5. Then the file names and corresponding aliases of any SSL program files that were linked and loaded with the main program are displayed.

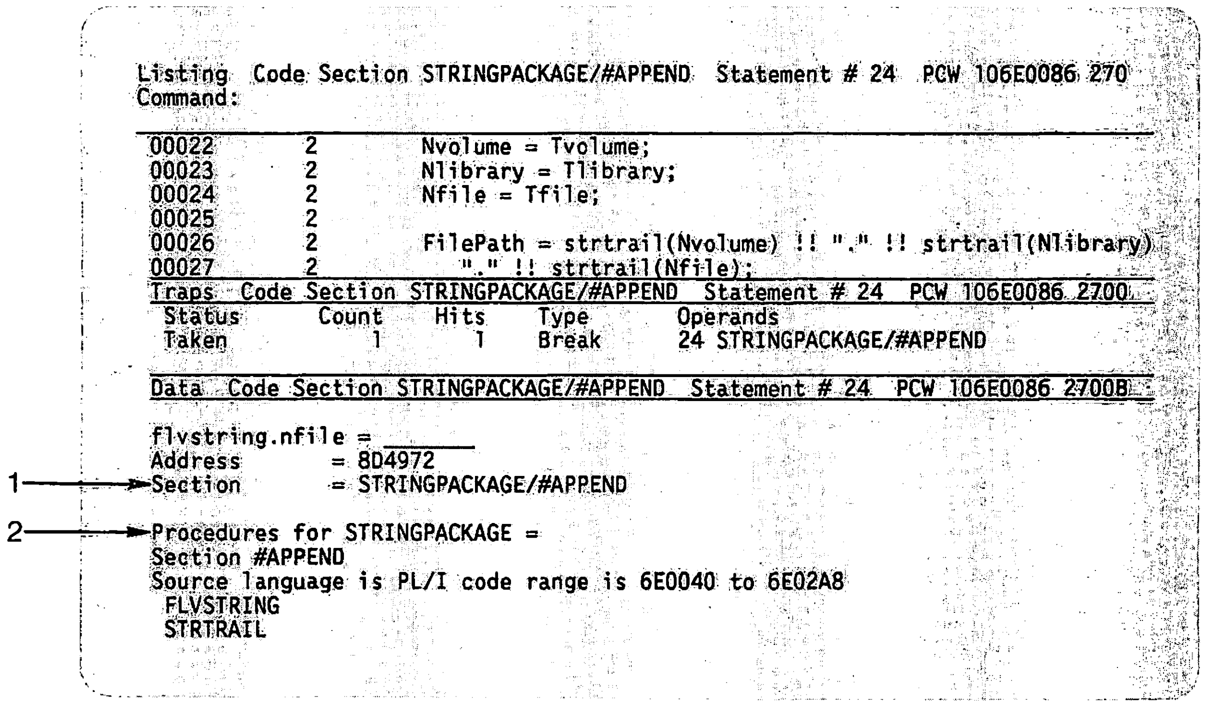
6. Last, the names of any programs that were mapped via MSMAP are displayed.

If the programmer then enters the following commands:

```
ATTRIBUTES ADDRESS SECTION
VARIABLE FLVSTRING.NVOLUME
PROCEDURES
```

a display similar to the one in Figure 6-2 appears in the data window.

Note: *The ATTRIBUTES command is a global setting that affects all subsequent VARIABLE commands.*



```
Listing  Code Section STRINGPACKAGE/#APPEND  Statement # 24  PCW 106E0086 270
Command:
_____________________________________________________________________________
 00022       2        Nvolume = Tvolume;
 00023       2        Nlibrary = Tlibrary;
 00024       2        Nfile = Tfile;
 00025       2
 00026       2        FilePath = strtrail(Nvolume) !! "." !! strtrail(Nlibrary)
 00027       2           "." !! strtrail(Nfile);
 Traps  Code Section STRINGPACKAGE/#APPEND  Statement # 24  PCW 106E0086 2700
  Status      Count   Hits    Type      Operands
  Taken          1       1    Break     24 STRINGPACKAGE/#APPEND

 Data  Code Section STRINGPACKAGE/#APPEND  Statement # 24  PCW 106E0086 2700B

 flvstring.nfile = ______
 Address        = 8D4972
 Section        = STRINGPACKAGE/#APPEND

 Procedures for STRINGPACKAGE =
 Section #APPEND
 Source language is PL/I code range is 6E0040 to 6E02A8
  FLVSTRING
  STRTRAIL
```

Figure 6-2. Example: Display of SSL Data Items

Explanation of Figure 6-2

1. Because the context is within an SSL, the section name is prefixed by the SSL's alias.

2. Again, the program name is displayed when the context is in an SSL.

   Note: *that if the context were in the main program, the programmer would have to enter PROCEDURES STRINGPACKAGE to display this list of procedures; without the operand STRINGPACKAGE, the display would list procedures in the main program rather than in the SSL STRINGPACKAGE.*

CHAPTER 7
DEBUGGER COMMAND SET

7.1 INTRODUCTION

This chapter describes each of the Debugger commands in alphabetical order. In addition, the chapter contains

- Descriptions of the three command classes (Section 7.2)
- Instructions on entering the commands of each class (Section 7.3)

7.2 COMMAND CLASSES

The Debugger commands are grouped into three classes for the purposes of describing how the commands work and how to enter them. The following list describes each command class:

Command Class	**Function**
Set-and-Query	Sets a new value or queries the current setting.
Action	Performs an action, often on marked lines of text. If none are marked, acts on the current cursor line.
Informative	Displays information on the message line or in a window created for this purpose.

Directions for entering each command class follow this section. The class of each command is listed in its description later in this chapter.

7.3 HOW TO ENTER COMMANDS

7.3.1 How to Enter Set-and-Query Commands

To Query the Setting

By PF Key

1. Press the PF key to which the command is assigned. The command is displayed in the command line with its current setting, if any. The cursor is displayed immediately after the command name.

2. After reading the setting, you can press ENTER to clear the command line or issue another command without pressing ENTER. (Issuing another command overwrites the display in the command line.)

At the Command Line

1. Type the command and press ENTER. If a setting was previously set, it is automatically displayed in the command line immediately after the command.

2. After reading the setting, you can press ENTER to clear the command line or issue another command without pressing ENTER. (Issuing another command overwrites the display in the command line.)

To Specify a Setting

By PF Key

1. Press the PF key to which the command is assigned. The command is displayed in the command line with its current setting, if any. The cursor is displayed immediately after the command name.

2. Type the new setting. (If a setting is already displayed, type the new setting over it.)

3. If only one window is open, press ENTER. If more than one window is open, move the cursor to the window for which the command is intended (it can be anywhere within the window) and press ENTER.

At the Command Line

1. Type the command and press ENTER. If a setting was previously set, it is automatically displayed in the command line immediately after the command.

2. Type the new setting. (If a setting is already displayed, type the new setting over it.)

3. If only one window is open, press ENTER. If more than one window is open, move the cursor to the window for which the command is intended (it can be anywhere within the window) and press ENTER.

7.3.2 How to Enter Action Commands

Many action commands require the use of the MARK command to designate the text to be affected. In these cases, the MARK command is used first, then the action command is invoked. If an Action command requires the use of the MARK command, that is specified in its description in this chapter. Directions for using the MARK command are given in Section 4.5.

By PF Key

1. If required, mark the text first with the MARK command.
2. If you are specifying an operand for the command, enter the operand in the command line.
3. If only one window is open, go to step 4. Otherwise, move the cursor to the window to be affected. The cursor can be anywhere within the window.
4. Press the PF key to which the command is assigned. The command is immediately executed.

At the Command Line

1. If required, mark the text first with the MARK command.
2. Type the command with any operands, if required. If only one window is open, press ENTER. If more than one window is open, move the cursor to the window for which the command is intended (it can be anywhere within the window), and press ENTER.

7.3.3 How to Enter Informative Commands

By PF Key

1. If only one window is open, go to step 3.
2. Move the cursor to the window to be affected. The cursor can be anywhere within the window.
3. Press the PF key to which the command is assigned. The command is immediately executed.

At the Command Line

1. Type the command with any operands, if required. If only one window is open, go to step 3.
2. Move the cursor to the window to be affected. The cursor can be anywhere within the window.
3. Press ENTER.

7.4 DEBUGGER COMMANDS

This section describes each of the Debugger commands in alphabetical order. The following information is presented for each command:

- Description
- Format
- Command class
- Syntax rules (if applicable)
- General rules (if applicable)
- Related commands (if applicable)

The notation conventions described in Section 1.13 apply to the commands described in this section. See Section 2.1 for a listing of the debugger commands arranged in functional categories.

ACTIVATE

Description

The ACTIVATE command activates one or more specified traps. When a trap is activated, its status is set to Active. You indicate the trap(s) to be activated either by positioning the cursor or marking traps.

Format

ACTIVATE

Command Class

Action

General Rules

1. If any traps are marked, those traps are activated.
2. If no traps are marked, and the cursor is positioned at a trap, that trap is activated.
3. If no traps are marked, and the cursor is not positioned at a trap, ACTIVATE activates the last trap in the trap window.
4. If ACTIVATE is executed and no traps are present in the trap window, an error message is displayed.

ALTER

Description

To change a program value, you enter the new value in the data window, and execute the ALTER command. ALTER works on one entry at a time; you can alter one or more values within the entry. Although it is possible to edit value fields in the data window, the actual program value for edited data items is not changed unless you execute ALTER. If ALTER is not executed, no changes are made, and the contents of the data window are restored to their current values the next time you access that window.

Note: *In the previous Debugger, it was possible to modify variables when executing in system code. As a result of enhanced security in VS Operating System Release 7.10, you must now be executing in user code in order to modify program variables.*

If your program is interrupted while in system code, single step your program to the next executable statement. Then modify the desired values in memory with the ALTER command.

Format

ALTER

Command Class

Action

General Rule

1. The ALTER command cannot be applied to displays produced by the CODESECTIONS, DIAGNOSTIC, PROCEDURES, STACKTRACE, or STATICSECTIONS commands or to entries that have been frozen.

ASSIGN

Description

The ASSIGN command assigns one or more commands to a PF key or displays the current PF key assignment.

Format 1

```
ASSIGN [text]    (Press a PF key)
```

Format 2

```
ASSIGN n [text]    (Press ENTER)
```

Command Class

Set-and-Query

Syntax Rules

1. The operand n indicates the PF key to be assigned.

2. The operand text indicates the command(s) you assign to the specified PF key. If two or more commands are entered, each command must be separated by a semicolon (;).

Note: *Validation of the assigned command string occurs during the execution of the string, not at the time you assign the string. For example, if you execute ASSIGN 1 MAKK, the Debugger assigns the command string MAKK to PF1 even though MAKK is not a valid command. When you execute PF1, the Debugger informs you that the command string (MAKK) is invalid.*

General Rules

1. PF key assignments made through ASSIGN are valid from every window.

2. ASSIGN overrides a corresponding PF key assignment for the current session only.

3. Any PF key other than ENTER can be pressed using Format 1; only ENTER can be pressed using Format 2.

4. If the text operand is omitted, then the current string assigned to the specified PF key (either that which is pressed using Format 1 or entered as n using Format 2) is displayed on the command line.

5. If ASSIGN is entered with no operand and ENTER is pressed, an error message is displayed; if ASSIGN is entered with no operand and a PF key is pressed, the current PF key assignment is displayed.

Special Considerations

ASSIGN cannot be executed by PF key.

Related Command

MENU

ATTRIBUTES

Description

The ATTRIBUTES command specifies the statistics to be displayed about each variable that appears in the data window. After you specify information by executing ATTRIBUTES and the desired operands, those statistics appear following each subsequent variable display in the data window. Existing displays are not changed.

Format 1

```
ATTRIBUTES [STORAGE] [TYPE] [SECTION] [ADDRESS] [SIZE]
```

Format 2

```
ATTRIBUTES OFF
```

Command Class

Set-and-Query

Syntax Rules

1. The operand STORAGE indicates the storage class of the corresponding variable (parameter, static, etc.).

2. The operand TYPE indicates the data type of the variable (integer, character, etc.).

3. The operand SECTION displays the code section for the corresponding variable.

4. The operand ADDRESS indicates the starting address of the corresponding variable.

5. The operand SIZE indicates the number of storage bytes that the corresponding variable occupies.

6. The operand OFF suppresses the display of attributes for all variables.

General Rules

1. The default operand for ATTRIBUTES is SECTION.

2. If no operands are chosen, the current attributes are displayed.

Examples

1. Command:

```
ATTRIBUTES SECTION SIZE
VARIABLE SUB1
```

Displays:

```
SUB1 = +1
Section         = COBDEMO
Size            = 1 byte
```

BREAK

Description

The BREAK command sets a trap at a specified program location, called the breakpoint. When this trap is active and execution of the subject program is resumed, the trap is taken when control reaches the breakpoint.

Format

```
BREAK  [statement-id [[program-name/]section-name]]
       [ADDRESS address]
       [OFFSET hex-offset [section-name]]
```

Command Class

Action

Syntax Rules

1. The keywords and operands associated with BREAK specify the location at which to set the trap. Their forms and definitions are described in Section 1.13.

2. The ADDRESS and OFFSET forms set a trap on the instruction at the specified address or offset.

3. The operand "address" can be either an absolute address or the base-index-displacement form of the address. If the base-index-displacement form is used, the address is not re-evaluated unless the trap is deleted and then recreated.

4. If no operands are specified, BREAK applies to the first statement in the current section that is marked. If there is more than one statement on a line, the first statement of the line is chosen. If no statements are marked, BREAK applies to the statement at which the cursor is positioned in the listing window. If none of these conditions are met, the Debugger displays an error message.

Example

1. BREAK 52

The Debugger sets the trap at the first statement on line 52.

CANCEL

Description

The CANCEL command exits from the Debugger and cancels the subject program. After you execute CANCEL, the Debugger displays the Cancel screen. From that screen, you can either confirm the cancel or resume debugging.

Format

```
CANCEL
```

Command Class

Action

CASE

Description

The CASE command is used to specify whether the case of letters must match a search string specified through the FIND command. The case indicates the condition of the letters with regard to capitalization; uppercase means capitalized and lowercase means not capitalized. CASE applies only to the current window.

A search that is case sensitive (CASE EXACT) locates a match only if the case of each letter as well as the letters themselves match the search string. A search that is not case sensitive (CASE ANY) locates a match whenever the letters match, regardless of their case.

To change the setting of CASE for a search, you must specify the case *before* you use the SEARCH or LOCATE commands.

Format

```
CASE [ANY]
     [EXACT]
```

Command Class

Set-and-Query

Syntax Rules

1. The operand ANY indicates a case insensitive search in which the case of the letters in the search string is not a criterion for a match.

2. The operand EXACT indicates a case sensitive search in which the case of the letters in the search string is also a criterion for a match.

General Rules

1. CASE applies only to search strings specified through the FIND command.

2. When CASE is assigned to a PF key, operands cannot be changed during the operation.

3. CASE with no operand displays the current case setting.

Related Commands

FIND, LOCATE, MATCH, SEARCH

CLEAR

Description

The CLEAR command selectively or totally removes marks, depending on the operand you specify.

Format

```
CLEAR [ALL]
      [NEXT]
      [PREVIOUS]
```

Command Class

Set-and-Query

Syntax Rules

1. The operand ALL indicates all marks.

2. The operand NEXT indicates all marks that exist for lines following the line at which the cursor is positioned, excluding that line.

3. The operand PREVIOUS indicates all marks that exist for lines preceding the line at which the cursor is positioned, excluding that line.

General Rules

1. CLEAR affects only the designated window.

2. When CLEAR is assigned to a PF key, operands cannot be changed during the operation.

3. CLEAR with no operand performs the same function as CLEAR ALL.

Related Command

MARK

CLOSE

Description

The CLOSE command removes the display window or the menu window from the Window section. After you execute CLOSE, the current window is removed and if a subsequent display or menu window exists, that window is assigned to the number of the closed window.

Format

```
CLOSE
```

Command Class

Action

General Rule

CLOSE has no effect on the listing window, the trap window, or the data window. If you attempt to execute CLOSE on those windows, an error message is displayed.

CODESECTIONS

Description

The CODESECTIONS command displays in the data window, a list of all code sections in the subject program.

Format

```
CODESECTIONS  [program-name]
```

Command Class

Action

Example

Command: CODESECTIONS

Displays:

```
CODESECTIONS =
COBDEMO
WC3DNR3
WC3SMV2
```

COLUMN

Description

The COLUMN command positions the cursor on a specified column within the current window. Typically, COLUMN is used in conjunction with the ROW command to move the cursor to a specified column within a designated row.

Format

```
COLUMN [n]
```

Command Class

Action

Syntax Rule

1. The operand n indicates a column number. Any value for n that is less than 1 defaults to 1; any value for n that is greater than the width of the window defaults to the window width limit.

General Rules

1. COLUMN operates with the cursor positioned in the current row.

2. Because COLUMN operates relative to text; left or right scrolling may occur depending on the operand entered.

3. If COLUMN is executed with no operands, the default for n is 1. The cursor is positioned at the left margin on column 1.

Related Commands

CURSOR, ROW

CONTINUE

Description

The CONTINUE command resumes execution of the subject program after it has been interrupted for a trap. Execution resumes either from the next instruction or at a specified location. For example, when you have finished examining the program state after a trap has interrupted execution, you execute CONTINUE with no operand to resume execution from the next instruction.

Format

```
CONTINUE        [statement-id]
                [ADDRESS address]
                [OFFSET hex-offset [section-name]]
```

Command Class

Action

Syntax Rule

1. The keywords and operands associated with CONTINUE specify an alternate location at which to resume program execution. Their forms and definitions are described in Section 1.13.

2. The address operand can be either an absolute address or the base-index-displacement form of the address.

General Rules

1. If no operand is specified, execution resumes at the next executable statement following the point at which execution was paused (the current address contained in the PCW).

2. If an operand is specified, the program address of the subject program is first changed to the value indicated by the operand, and then execution is resumed at that address.

3. All active traps remain available to be taken after program execution has resumed.

Examples

1. CONTINUE

   The most common use of CONTINUE is to specify the command without any operands, as shown in this example. Program execution continues until completion or when conditions for a subsequent trap are met.

2. CONTINUE 49

   The subject program resumes execution at the first statement on line 49.

COUNT

Description

The COUNT command changes the count for a specified trap. The count is the number of times that the condition for the trap is to be encountered before the trap is taken. For example, if the count for a trap is 3, the conditions for the trap must be satisfied three times before the trap is taken.

Newly created traps have a count of 1, meaning that the trap is taken on the first instance after the program is resumed. COUNT specifies an alternate count for a chosen trap.

The count for each trap is shown in the trap window. To change the count for a trap, indicate the desired trap(s) using the cursor or marking, and execute COUNT followed by the desired operand. The operand is the number to which you want to change the count.

Format

```
COUNT n
```

Command Class

Action

Syntax Rule

1. The operand n specifies a count for the corresponding trap.

General Rules

1. All commands for setting traps create them with a count of 1.
2. COUNT with no operand causes an error message to be displayed.
3. COUNT has no effect on traps set by the INSIDE, OUTSIDE, or MODTRAP commands.
4. If any traps are marked in the trap window, COUNT operates on each marked trap.
5. If no traps are marked, but the cursor is positioned on a trap, COUNT changes the count for that trap to n.
6. If no traps are marked and the cursor is not positioned on a trap, COUNT changes the count of the last trap in the trap window to n.
7. Whenever the count for a trap is changed, the HITS field for that trap is automatically set to 0.

Example

1. COUNT 5

For the following trap (set by the BREAK command at statement 60 with a count of 5) the trap is taken after its conditions have been satisfied five times in the subject program.

```
Status      Count      Hits     Type      Operands
Active        5         0       Break     60 COBDEMO
```

CURSOR

Description

The CURSOR command positions the cursor at a specified location on the Debugger Workstation screen (columns 1-80, rows 1-24). The cursor only is moved; the window text does not scroll except in certain situations when the FIRST or LAST operands are specified.

Format

```
CURSOR        SAVE
              RESTORE
              FIRST
              LAST
              UP [n]
              DOWN [n]
              LEFT [n]
              RIGHT [n]
```

Command Class

Action

Syntax Rules

1. The operand SAVE saves the most recent cursor position.
2. The operand RESTORE positions the cursor at the location recorded by CURSOR SAVE.
3. The operand FIRST positions the cursor at the first nonblank character of the current row.
4. The operand LAST positions the cursor at the last nonblank character of the current row.
5. The operand UP [n] positions the cursor up n rows. If you do not specify n, UP positions the cursor up 1 row.
6. The operand DOWN [n] positions the cursor down n rows. If you do not specify n, DOWN positions the cursor down 1 row.
7. The operand LEFT [n] positions the cursor left n columns. If you do not specify n, LEFT positions the cursor left 1 column.
8. The operand RIGHT [n] positions the cursor right n columns. If you do not specify n, RIGHT positions the cursor right 1 column.

General Rules

1. When CURSOR is executed by PF key, operands cannot be changed during the operation.

2. If CURSOR is executed with no operands, no action is taken.

3. If the cursor is positioned in the Control section of the Debugger Workstation screen, CURSOR SAVE performs no function.

Related Commands

COLUMN, ROW

DATA

Description

The DATA command accesses the data window. On the data window, you can display chosen variables, portions of memory, registers, and other various program data.

Format

DATA

Command Class

Action

Related Commands

DISPLAY, FRAME, LISTING, MENU, TRAPS, WINDOW

DEACTIVATE

Description

The DEACTIVATE command deactivates one or more specified traps. When a trap is deactivated, its status is set to Inactive. You indicate the trap(s) to be deactivated either by positioning the cursor or by marking traps.

Format

DEACTIVATE

Command Class

Action

General Rules

1. If any traps are marked, those traps are deactivated.

2. If no traps are marked, and the cursor is positioned at a trap, that trap is deactivated.

3. If no traps are marked, and the cursor is not positioned at a trap, DEACTIVATE deactivates the last trap in the trap window.

4. If DEACTIVATE is executed and no traps are present in the trap window, an error message is displayed.

DEBUGFILE

Description

The DEBUGFILE command enables you to query and reset linkage and debugger information in an alternate program file. This command is useful when you are debugging large program files that do not fit into memory with all of the symbolic and linkage information.

Format

```
DEBUGFILE [alternate program file name][program name]
```

Command Class

Set-and-Query

General Rules

1. When your program will not fit into memory with the symbolic and linkage information, you must use the Linker to create two program files: one program file linked with the symbolic and linkage information, and an alternate program file linked without the symbolic and linkage information. It is essential to repeat this step each time you make modifications so that all offsets within the sections, e.g., listing files, etc., remain synchronized.

2. When debugging Shared Subroutine Libraries (SSLs), you can specify the alternate program file for a specific program name. This enables you to specify alternate program files for your SSLs in your DEBSTART file. See Chapter 6 for information on debugging SSLs.

3. When the program is unlinked, the DEBUGFILE command settings are not remembered by the Debugger. You must reissue the command if you wish to query the alternate program after an unlink.

4. When DEBUGFILE is executed by PF key, operands cannot be changed during the operation.

5. If the system displays the following message in the listing window:

   "PCW is not defined for any known section."

   and the PCW contains a valid Program Code Section address, you can use the DEBUGFILE command to access the alternate program file that contains the linkage information for the program.

   If the system displays the following message:

   "Section XXXXXXX is non-symbolic."

   you can use the DEBUGFILE command to access the alternate program file that contains the symbolic information for that section.

DEFINE

Description

The DEFINE command defines a macro or displays the current macro definition.

Format

```
DEFINE macro-name [macro-definition]
```

Command Class

Set-and-Query

Syntax Rules

1. The operand macro-name indicates any valid identifier name that starts with a letter.

2. The operand macro-definition indicates the commands to be assigned to the macro-name.

Note: *Validation of the defined command string occurs during the execution of the string, not at the time you assign the string. For example, if you execute DEFINE M5 MAKK 5, the Debugger defines the macro M5 as the command string MAKK 5, even though MAKK 5 is not a valid command. When you execute M5, the Debugger informs you that the command string (MAKK 5) is invalid.*

General Rules

1. The macro-name can be a maximum of 32 alphanumeric characters, always beginning with a letter.

2. Macro definitions made through DEFINE are valid from every window.

3. A macro that has the same name as a command or a command abbreviation disables the command or abbreviation.

4. DEFINE can be executed only at the command line using ENTER, or through a command file. If DEFINE is entered on the command line and a PF key is pressed instead of ENTER, an error message is displayed and no action is taken.

5. If the operand macro-definition is not specified, then the current definition of the specified macro (if any) is displayed on the command line.

6. If a macro-definition is specified, that macro-name is set to that definition, replacing any previous macro-definition associated with that macro-name.

7. DEFINE with no operand performs no function.

8. The maximum number of nested macro levels (macros that call macros) is 16.

9. A macro definition can be removed by redefining the macro as itself. For example, if you execute DEFINE FULLDISP FULLDISP, the previous macro definition of FULLDISP is removed, and the macro FULLDISP no longer exists.

DELETE

Description

The DELETE command deletes specified traps from the trap window or specified entries from the data window.

Format

DELETE

Command Class

Informative

General Rules

1. If DELETE is executed against any window other than the trap window or the data window, an error message is displayed.

2. When you execute DELETE, entries are deleted according to the following hierarchy:

   a. If any lines are marked, the corresponding entry is deleted. Note that an entire entry (which may span a number of lines) is deleted if one or more lines in the entry are marked.

   b. If no lines are marked, the entry at which the cursor is positioned is deleted.

   c. If no entries are marked and the cursor is outside of the current window, the last entry of the current window is deleted.

3. When an entry is deleted, the following entries (if any) move up in the list.

4. Only one entry at a time can be deleted from the data window due to the marking rules. (Refer to the MARK command, described later in this section, for more information about the marking rules.)

DIAGNOSTIC

Description

The DIAGNOSTIC command displays the full text of the last runtime diagnostic message reported against the subject program, if any message exists. The message is displayed in the data window.

Format

```
DIAGNOSTIC
```

Command Class

Action

General Rule

1. If a runtime diagnostic message does not exist, a message, which states that fact, is displayed.

DISPLAY

Description

The DISPLAY command displays any VS file in the display window. To display a file, execute the DISPLAY command followed by the volume, library, and file name (separated by periods) of the file you want to display. The Debugger displays the file in the display window and numbers that window one number greater than the previous window.

You can display only one file at a time. To cancel a file display, execute the CLOSE command from the display window.

Format

DISPLAY [[volume.] library.] file

Command Class

Action

Syntax Rule

1. The volume, library, and file operands specify the file to be displayed.

General Rules

1. If the volume and/or library specifications are omitted, the input volume and the input library are used.

2. The volume, library, and file names must be separated by periods; no embedded spaces are allowed.

3. When DISPLAY is executed by PF key, operands cannot be changed during the operation.

Related Commands

CLOSE, DATA, FRAME, LISTING, MENU, TRAPS, WINDOW

DUMP

Description

The DUMP command creates a file that contains a core dump of current user memory. The file is assigned a default system name which takes the form DUMPnnnn where nnnn represents the next higher number of all such files in your library.

Format

```
DUMP
```

Command Class

Action

General Rule

1. The dump file is placed in your spool library on the spool volume.

EASY

Description

The EASY command enables you to run the Debugger in a simplified mode that is designed for the new or infrequent user. Chapter 3 presents a detailed description of EASY.

Format

```
EASY [ON]
     [OFF]
```

Command Class

Action

Syntax Rules

1. The operand ON causes the Debugger to function in a simplified mode.

2. The operand OFF returns the Debugger to its natural state.

FIND

Description

The FIND command specifies a string, the find-string, that is to be used in string searching, or queries the current setting for the find-string.

The command FIND text sets the find-string to the value text. If a succeeding search command is given without a find-string operand, "text" is the string that will be searched for. This value of the find-string remains in effect until another FIND text command is given, or until a SEARCH, LOCATE, or CHANGE command is given with a find-string operand.

Format

```
FIND [text]
```

Command Class

Set-and-Query

Syntax Rules

1. The operand text indicates the text entered as the search string.

2. If the FIND command is assigned to a PF key without an operand, you can set the find-string in the following way:

   - Type the find-string in the command line without enclosing quotes.

   - Press the PF key to which FIND is assigned.

General Rules

1. When you specify a search string that includes punctuation or embedded spaces, the following rules apply:

   a. If the string contains one or more embedded spaces or a semicolon, it must be enclosed in quotation marks. Single or double quotation marks can be used.

   ```
   Examples:     FIND 'example one'
                 or
                 FIND "example one"
   ```

   b. If the string contains double quotation marks, it must be enclosed by single quotation marks.

   ```
   Example:      FIND 'This is an "example section"'
   ```

2. The CASE command applies to the search string specified by the FIND command.

3. FIND with no operand displays the current value of the find-string in the command line as part of the FIND command.

4. The maximum length of a search string specified by FIND is 256 characters or the end of the command line.

Related Commands

CASE, LOCATE, MATCH, SEARCH

FIRST

Description

The FIRST command scrolls the display so that the first line of text is positioned at the top of the current window.

Format

```
FIRST
```

Command Class

Action

Related Commands

GOTO, LAST, LEFT, NEXT, PREVIOUS, RIGHT

FLOATREGISTERS

Description

The FLOATREGISTERS command displays in the data window the values of all floating-point registers of the subject program.

Format

```
FLOATREGISTERS
```

Command Class

Informative

Example

```
Command:  FLOATREGISTERS

Displays:

Floatregisters =
(F0) = 41 70000000000000
(F2) = 00 00000000000000
(F4) = 00 00000000000000
(F6) = 00 00000000000000
```

FRAME

Description

The FRAME command is used to control the size of windows. FRAME FULL specifies the full Window section as the window size (21 lines), and enables the display of one window at a time. FRAME PARTIAL specifies partial window format of the same width as a full window. The length of a partial window, which can range from 1 to 21 lines, is determined by the position of the cursor when you access the window, and the current screen display.

When you access a window with FRAME FULL in effect, the entire window is displayed and any previous window is no longer displayed. When you access a window with FRAME PARTIAL in effect, if the cursor is on the command line, the accessed window occupies the full screen, the same as for the FRAME FULL setting.

However, with FRAME PARTIAL in effect, if the cursor is somewhere in the Window section when you access a window, that window is displayed starting at the current cursor location. The display of the previous window remains but is terminated where the new window begins. The accessed window extends downward either to the bottom of the Window section, or to the top of the next lower window if one exists. Only the upper boundary of a new window is selected by the cursor; the lower boundary is determined automatically.

Format

```
FRAME [FULL]
      [PARTIAL]
```

Command Class

Set-and-Query

Syntax Rules

1. The operand FULL specifies full display of accessed windows.

2. The operand PARTIAL specifies partial display of accessed windows.

General Rules

1. The status of FRAME affects all windows.

2. When FRAME is assigned to a PF key, operands cannot be changed during the operation.

3. FRAME with no operand displays the current setting.

Related Commands

DATA, DISPLAY, FULL, LISTING, MENU, TRAPS, WINDOW

FREEZE

Description

The FREEZE command is used to specify whether a particular marked entry in the data window is to be updated to its current value. A FREEZE setting exists for every entry in the data window.

Format

```
FREEZE [ON]
       [OFF]
```

Command Class

Set-and-Query

Syntax Rules

1. The operand ON causes the Debugger to maintain the data window entry value. No updating of the value takes place, regardless of the entry's current value.

2. The operand OFF causes the Debugger to update the data window entry with the current program value.

Related Command

MARK

FULL

Description

The FULL command expands the size of the current window to the full size of the Window section (21 lines). If other windows were displayed at the time, they are no longer visible but can still be accessed.

Format

FULL

Command Class

Action

Related Commands

DATA, DISPLAY, FRAME, LISTING, MENU, TRAPS, WINDOW

GOTO

Description

> ***Note:*** *In this command description, the terms "line" and "line number" are used in the context of a window; neither term has any relevance to a program statement number.*

The GOTO command scrolls to the specified line within the current window. If the line is within the current display, the cursor is positioned to that line. If the line is not within the current display, the line is displayed at the top of the window with the cursor positioned at column 1 (except for GOTO LAST).

Format

```
GOTO   [n]
       [FIRST]
       [LAST]
       [MARK]
       [NOTE]
```

Command Class

Action

Syntax Rules

1. The operand n indicates any specified line number.

2. The operand FIRST indicates the first line of text.

3. The operand LAST indicates the last line of text.

4. The operand MARK indicates the first line in a marked range.

5. The operand NOTE is used in conjunction with the NOTE command. NOTE records the location of a particular line of text. GOTO NOTE scrolls the text so that the line recorded by the NOTE command is displayed at that same position.

General Rules

1. GOTO with no operand performs no function.

2. When GOTO is executed by PF key, operands cannot be changed during the operation.

3. GOTO LAST displays the last line of text on the last line of the window.

Related Commands

FIRST, LAST, LEFT, NEXT, NOTE, PREVIOUS, RIGHT

HELP

Description

The HELP command displays on-line information about the Debugger, called Help text. Help text describes the entire debugging environment. To display Help text, execute the HELP command. The Debugger then accesses the VS INFO utility which displays the Help menu. The Help menu lists options for locating desired help information.

Format

HELP

Command Class

Informative

General Rules

1. To navigate through the Help text, position the cursor on the first digit of any desired topic number and press ENTER.

2. Press PF16 to exit from Help text. The Debugger returns to the display from which the HELP command was executed.

Related Command

MENU

HEX

Description

The HEX command changes the display format of a variable in the data window. The display format can be either symbolic or hexadecimal. If the display format is symbolic, execution of HEX changes it to hexadecimal; if the display format is hexadecimal, execution of HEX changes it to symbolic.

Format

HEX

Command Class

Action

General Rule

1. When you execute HEX, the display formats of variables are changed according to the following hierarchy:
   a. If any lines are marked, the display format of the variable for the corresponding entry is changed.
   b. If no lines are marked, the display format of the variable at which the cursor is positioned is changed.
   c. If no lines are marked and the cursor is positioned outside of the data window, the display format of the variable on the last line is changed.
2. If you are executing this command through a PF key, you should first mark the desired entry in the data window or position the cursor at any location within the range of the entry.

HISTORY

Description

The HISTORY command controls the recording of commands entered at the command line, or queries the current setting. This enables you to easily repeat (with the RECALL command) a command you had previously typed.

Format

```
HISTORY [ON]
        [OFF]
```

Command Class

Set-and-Query

Syntax Rules

1. The operand ON provides a record of all commands entered at the command line, including macros.

2. Commands entered by PF keys are not recorded, nor are the following commands: ASSIGN, DEFINE, HISTORY, and RECALL.

Related Command

RECALL

INDICATORS

Description

The INDICATORS command displays in the data window the values of specified RPG II indicators. All indicators of the specified class found in the symbolic section are displayed, seven indicators to a line.

Format 1

INDICATORS [class [selector]]

Format 2

INDICATORS ALL

Command Class

Informative

Syntax Rules

1. The operand "class" indicates one of the following classes.

   - EXTERNAL (U1-U8)
   - GENERAL (01-99)
   - HALT (H1-H9)
   - KEYS (K0-K9, KA-KG)
   - LEVEL (L0-L9)
   - MISCELLANEOUS (LR, MR, 1P)
   - OVERFLOW (OA-OG, OV)
   - SHIFTKEYS (S1-S9, SA-SG)

2. The operand "selector" indicates the valid selection values for a subrange of indicators, which are shown in parenthesis following each class in Syntax Rule 1.

3. The operand ALL indicates that each class listed under Syntax Rule 1 is to be displayed.

General Rules

1. INDICATORS is valid only when the current code section is an RPG II code section.

2. If no indicators are found for the specified class, no display is created and a message is issued.

3. The word OFF is displayed next to all OFF indicators (internal value 00).

4. The word ON is displayed (brightly) next to all ON indicators (internal value F0).

5. The 2-character hexadecimal values of incorrectly defined indicator values (not 00 or F0) are displayed in blinking mode.

Examples

1. Command: INDICATORS

   Displays:

```
General Indicators 01 through 99 in section RPGOBJ
   02 = OFF   03 = OFF   04 = 3A    06 = OFF   29 = OFF   30 = OFF
   49 = OFF   50 = OFF   51 = OFF   76 = OFF   77 = EE    78 = OFF
   79 = OFF
Halt Indicators in section RPGOBJ
   H0 = ON
Key Indicators in section RPGOBJ
   K0 = OFF   K1 = OFF   K2 = OFF   K3 = OFF   K4 = OFF   K5 = OFF
   K6 = OFF   K7 = OFF   K8 = OFF   K9 = OFF   KA = OFF   KB = OFF
   KC = OFF   KD = OFF   KE = OFF   KF = OFF   KG = OFF
Miscellaneous Indicators in section RPGOBJ
   LR = OFF   1P = ON
```

   In this example, the display of indicators H0 and 1P (underlined) would appear highlighted since they have the value ON. The indicators 04 and 77 would appear blinking since they have illegal values.

2. Command: INDICATORS GENERAL 44 66

   Displays:

```
General Indicators 44 through 66 in section RPGOBJ
   49 = OFF   50 = OFF   51 = OFF
```

INSIDE

Description

The INSIDE command sets a trap at all points inside a specified range or a marked range. When this trap is active, the trap is taken if execution of the subject program is resumed and control reaches any point within the marked range.

Note: *The INSIDE command functions as an instruction level trap, not as a statement level trap. For example, the Debugger may interrupt the subject program a number of times at the same statement, because the statement contains multiple instructions.*

Format

```
INSIDE [statement-id-1 statement-id-2 [program-name/]section-name]]
       [ADDRESS address-1 address-2]
       [OFFSET hex-offset-1 hex-offset-2 [program-name/[section-name]]
```

Command Class

Informative

Syntax Rules

1. The keywords and operands associated with INSIDE specify the range to trap. Their forms and definitions are described in Section 1.13.

2. The operands address-1 and address-2 can be either an absolute address or the base-index-displacement form of the address. If the base-index-displacement form is used, address-1 and address-2 are not re-evaluated unless the trap is deleted and then recreated.

General Rules

1. INSIDE includes the range boundaries; that is, the trap is taken on either boundary statement and on any statement within the boundaries.

2. INSIDE requires either operands or marked lines to be executed.

3. If no operands are specified, INSIDE applies to the range of lines in the current section that are marked.

4. If marks are used, the range covers all statements included within the marks, including the marks that establish the boundaries. There must be statement number information for both the first and last marked lines.

5. If the statement-id form is used, the range includes every instruction from the first instruction of statement-id-1 to the last instruction of statement-id-2, inclusive.

6. If the ADDRESS form is used, the range includes every instruction from the instruction at address-1 to the instruction at address-2, inclusive.

7. If the OFFSET form is used, the range includes every statement from the hex-offset-1 (within the section-name if specified) to the hex-offset-2 (within the section-name if specified), inclusive.

8. If the operand section-name is not specified, the current section name is used.

9. Statement-id-1 must be less than statement-id-2; address-1 must be less than address-2 and both addresses must be halfword aligned; hex-offset-1 must be less than hex-offset-2 and both offsets must be halfword aligned.

Example

1. INSIDE 20 30

   The Debugger traps at all statements within the range of 20 to 30 inclusive.

LAST

Description

The LAST command scrolls the display so that the last display line appears on the last line of the current window.

Format

```
LAST
```

Command Class

Action

Related Commands

FIRST, GOTO, LEFT, NEXT, PREVIOUS, RIGHT

LEFT

Description

The LEFT command moves the entire current window display the specified number of columns to the left. LEFT is used alternately with the RIGHT command to view a window display that is wider than the physical screen. You use RIGHT to display columns to the right of the limit, and LEFT to scroll back in the opposite direction.

Format

```
LEFT [n]
```

Command Class

Action

Syntax Rule

1. The operand n indicates the number of columns to move the display to the left. Any value is accepted; values less than 1 or greater than the screen width are truncated accordingly.

General Rules

1. LEFT with no operand displays the text beginning from the left margin.

2. When LEFT is executed by PF key, operands cannot be changed during the operation.

3. When the leftmost margin is displayed on the window, LEFT no longer has any effect when executed.

Related Command

RIGHT

LINKLEVELS

Description

The LINKLEVELS command sets a trap on all LINK SVC instructions and UNLINK SVC instructions in the subject program. If you specify LINKLEVELS ON, control is transferred to the Debugger each time the link level changes, and you are informed on the message line.

Format

```
LINKLEVELS ON
           OFF
```

Command Class

Action

Syntax Rules

1. The operand ON causes control to be transferred to the Debugger each time the link level changes, and a message to be displayed on the message line.

2. The operand OFF suppresses the transfer of control to the Debugger each time the link level changes.

General Rules

1. The default operand for LINKLEVELS is OFF.

LISTFILE

Description

The LISTFILE command allows you to access and display an alternate copy of the listing file during a debugging session. For example, if the volume containing a listing file is not mounted, you can issue the LISTFILE command to access a copy of the listing file on a volume that is currently mounted.

Format

```
LISTFILE [[[volume.] library.] file]
```

Command Class

Set-and-Query

General Rule

1. When LISTFILE is executed by PF key, you cannot change the operands during the operation.

LISTING

Description

The LISTING command accesses the listing window.

Format

```
LISTING
```

Command Class

Action

Related Commands

DATA, DISPLAY, FRAME, MENU, TRAPS, WINDOW

LOAD

Description

The LOAD command loads and executes a specified command file. A command file contains a series of commands that are executed by the Debugger in their order of appearance. You can execute a command file as many times as desired. A command file can itself execute LOAD to load other command files.

Format

```
LOAD [[volume.] library.] file
```

Command Class

Action

Syntax Rule

1. The volume, library, and file operands specify the file to be loaded.

General Rules

1. When LOAD is assigned to a PF key, operands cannot be changed during the operation.

2. The volume, library, and file names must be separated by periods; no embedded spaces are allowed.

3. If the volume and/or library specifications are omitted, the input volume, and the input library are used.

LOCATE

Description

The LOCATE command searches the text for an instance of a specified string or pattern. If text is specified it becomes the new find-string, otherwise the previous find-string is used. LOCATE searches from the current cursor position forward until an instance of the find-string is located. If the end of the file is reached without locating a find-string, the search is then reversed, until the find-string is located or the beginning of the file is reached.

When an instance is found, the cursor is positioned on its first character. If no instance is found, a message is displayed, and the file is not scrolled.

If the CASE EXACT command is in effect, an instance of the string must also match the find-string with regard to the capitalization of each letter. If CASE ANY is in effect the letters are compared without regard to capitalization.

If the MATCH ON command is in effect, the find-string is interpreted as a pattern. It is compared to strings in the text according to pattern matching rules. If MATCH OFF is in effect, strings in the text are compared to the find-string just as it is.

Format

```
LOCATE [text]
```

Command Class

Action

General Rules

1. When LOCATE is assigned to a PF key, operands cannot be changed during the operation.

2. If the search string is not located, no action is taken.

Related Commands

CASE, FIND, MATCH, SEARCH

MARK

Description

The MARK command marks a line or a range of lines to be used by another command. Each line that is marked is identified by a triangle in the left margin.

Each window can have a marked range of lines; however, only one range of lines can exist within a window.

After you mark the desired lines, you can execute subsequent commands that use the marked lines. Some commands automatically clear marks after they are executed, but most commands do not. You can clear marks by executing the CLEAR command.

You can modify commands assigned to PF keys that use marks, to automatically remove the marks following command execution. This is done by adding the CLEAR ALL command to the PF key assignments. For example, the command PRINT; CLEAR ALL creates a print file from the marked lines, and then clears all marks from the associated window.

Format

```
MARK [n]
     [ALL]
     [FIRST]
     [LAST]
```

Command Class

Action

Syntax Rules

1. The operand n indicates the nth line to be marked.

2. The operand ALL indicates all lines in the window. This operand is not allowed in the data window.

3. The operand FIRST indicates the first line in the window.

4. The operand LAST indicates the last line in the window.

General Rules

1. MARK with no operand marks the line at which the cursor is positioned. If the cursor is positioned outside of the window, the top line of the current window is marked.

2. If a range is marked and you execute MARK with the cursor positioned outside of the existing range but within the window, the Debugger marks all lines from the boundary of the existing range to the current cursor line, including that line.

3. Only one entry can be marked at a time in the data window. If an entry is marked and MARK is executed on another entry, the marks are cleared from the previous entry.

Related Command

CLEAR

MATCH

Description

The MATCH command enables pattern matching, a method of representing the find-string with some literal characters and with some symbolic characters. This method of representation enables you to locate strings with variable components. For more information about pattern matching and the symbolic characters, refer to Section 4.4.3.

You specify the find-string with the FIND, SEARCH, or LOCATE command. You can make the search case sensitive with the CASE command.

When MATCH is enabled, any symbolic characters within the find-string are interpreted according to their particular meaning within the pattern matching rules. When MATCH is set to OFF, any symbolic characters are interpreted according to their literal meaning.

Format

```
MATCH [ON]
      [OFF]
```

Command Class

Set-and-Query

Related Commands

FIND, LOCATE, SEARCH

MEMORY

Description

The MEMORY command displays in the data window, the value of a specified range of the subject program's memory.

Format

```
MEMORY    [address [length]]
          [OFFSET hex-offset [base] [length]]
where:

base = [program-name/]section-name or (absolute address) or (Rx)

where:

R = a general register and x denotes a general register number
```

Command Class

Informative

Syntax Rules

1. The keywords and operands (aside from "length") associated with MEMORY, specify the location in the subject program from which to display memory. Their forms and definitions are described in Section 1.13.

2. The operand "length" specifies the decimal number of bytes to be displayed. The maximum value is 4096 bytes; if a larger value is specified, a warning message is displayed. The default number of bytes is 4.

General Rules

1. The default for the operand "section" is the current section.

2. If MEMORY is assigned to a PF key, and is executed through that key, it has the following special behavior:

   If the command line is not blank, the text within the line is executed as the MEMORY command.

   Thus, an alternate way to display a portion of memory is to enter its address (or general register) and length on the command line, and then press the PF key assigned to the MEMORY command.

3. If MEMORY is executed with no operands, but the cursor is positioned on a displayed value that represents an address in the subject program, the four bytes starting at that address are displayed.

Examples

1. Command: MEMORY 8FD010

   Displays:

```
MEMORY 8FD010 =
8FD010   00   002FF200                                              " / "
```

   Since no length was specified, the default length of 4 was used.

2. Command: MEMORY 4(RF) 64

   Displays:

```
MEMORY 4(RF) 64 =
2E4DDC  0000  0000000C 00000054 2A010402 002E4EA0   "      TÜ    N "
2E4DEC  0010  002E4E20 002F2010 0013DC28 40038816   "  Nz     (@   "
2E4DFC  0020  002E5020 802E4E5C 7013B212 002E4E28   "  P   N p   N("
2E4E0C  0030  002F16FC C02F183E 00000000 00000000   " /    / >     "
```

Related **Command**

SET

MENU

Description

The MENU command displays the menu window, which contains a menu of the current PF key assignments. The menu window is displayed either in a full window if FRAME FULL is in effect, or in a partial window if FRAME PARTIAL is in effect (assuming the cursor is positioned within the Window section).

When you execute MENU with FRAME FULL in effect, the menu window is displayed as a full window with both unshifted and shifted PF keys visible. When you execute MENU with FRAME PARTIAL in effect, the menu window may be displayed as a partial window, depending on the position of the cursor. If displayed as a partial window, the entire set of PF key assignments may not be visible in the menu window. You can scroll the window display to view all assignments.

The menu window remains in view until it is closed or replaced by another window. To remove the menu window, execute the CLOSE command with the menu window as the current window.

Format

```
MENU
```

Command Class

Action

General Rule

1. MENU displays the PF key assignments for examination only; to modify the assignments you must execute the ASSIGN command.

Related Commands

ASSIGN, CLOSE, DATA, DISPLAY, FRAME, LISTING, TRAPS, WINDOW

MODTRAP

Description

The MODTRAP command sets a trap that monitors a specified program variable, program address, or a general register of the subject program. When this trap is active and the subject program is resumed, the trap is taken each time the value of the specified variable, address, or general register (subject to the relop clause if included) changes.

Format

```
MODTRAP    [VARIABLE var-name]
           [ADDRESS address [length]]
           [OFFSET hex-offset [program-name/]section-name [length]]
           [REGISTER greg [relop hexval]]
```

Command Class

Action

Syntax Rules

1. The operand var-name indicates the name of a subject program variable.

2. The operand address indicates the address on which the trap is set. The address operand can be either an absolute address or the base-index-displacement form of the address (as shown in Section 1.13). If the base-index-displacement form is used, the address is not re-evaluated unless the trap is deleted and then recreated.

3. The operand length indicates the length of the specified address. Any operand greater than 256 is replaced by 256 bytes. If the length operand is not specified, the default length is 4 bytes.

4. The operand section must refer to a static section, not a code section.

5. The operand greg identifies the name of the 32-bit general register to be monitored by the Debugger. Valid names are R0 through R15, RA through RF, and 0 through 15.

6. The operand relop is described in Section 1.13.2.

7. The operand hexval indicates a hexadecimal value of eight characters or less.

8. The relop greg-value expression specifies the condition under which the trap is to be taken. For example, in the following format, the trap is taken on R1 when the value of R1 becomes greater than 10.

   ```
   MODTRAP R1 > 10
   ```

9. If relop greg-value is not specified, the trap is taken on any attempt to put data in the specified general register that is different from the current contents of that general register.

10. Any variable with a length greater than 256 bytes is partially monitored; only the first 256 bytes are monitored.

Examples

1. MODTRAP VARIABLE SUB1

   The trap is taken each time the value of SUB1 changes.

2. MODTRAP REGISTER R4 = 3

   The trap is taken whenever the value for general register 4 is equal to 3.

NEXT

Description

The NEXT command scrolls the current window display forward a specified number of lines.

Format

```
NEXT [n]
```

Command Class

Action

Syntax Rule

1. The operand n indicates the number of lines to scroll forward.

General Rules

1. If no operand is specified, the default for n is the number of lines minus 2 if five or more lines are visible in the window, or the number of lines if less than five lines are visible in the window.

2. When NEXT is executed by PF key, operands cannot be changed during the operation.

3. If the operand n specifies a scroll that reaches the end of the window display, the last line of the window display is shown at the top of the window.

Related Commands

FIRST, GOTO, LAST, LEFT, PREVIOUS, RIGHT

NOTE

Description

The NOTE command saves the window position and cursor location so you can restore them later with the GOTO command.

NOTE records the location of a line in the current window. You can later invoke the GOTO NOTE command in that window. GOTO NOTE causes the text that includes that line to be displayed in the window and the cursor to be restored to where it was when NOTE was invoked.

Format

NOTE

Command Class

Action

Related Command

GOTO

OPCODE

Description

The OPCODE command sets a trap that monitors all instances of a specified machine instruction in the entire subject program. When this trap is active, the trap is taken if execution of the subject program is resumed and reaches any instance of the specified instruction.

Format

```
OPCODE [inst-mnemonic]
       [HEX hexval]
```

Command Class

Action

Syntax Rules

1. Inst-mnemonic indicates the mnemonic name of a valid VS instruction.

2. Hexval indicates a sequence of two hexadecimal characters.

3. OPCODE traps operate only on the opcode byte of VS instructions. OPCODE traps do not recognize extended mnemonics such as BL, BO, and RTZ.

Examples

In the following examples, the mnemonic name AH is used. This is the instruction to Add Halfword. Its hexval is 4A. Hexadecimal values of operation codes have a 2-character length. The two examples are equivalent.

1. OPCODE AH

   The Debugger sets a trap on all instances of the AH machine instruction.

2. OPCODE HEX 4A

   The Debugger sets a trap on all instances of the hexadecimal 4A instruction.

OUTSIDE

Description

The OUTSIDE command sets a trap at all points outside a specified range. When this trap is active, the trap is taken if execution of the subject program is resumed and control reaches any point outside the specified range.

Note: *The OUTSIDE command functions as an instruction level trap, not as a statement level trap. For example, the Debugger may interrupt the subject program a number of times at the same statement, because the statement contains multiple instructions.*

Format

```
OUTSIDE [statement-id-1 statement-id-2 [program-name/]section-name]]
        [ADDRESS address-1 address-2]
        [OFFSET hex-offset-1 hex-offset-2 [program-name/]section-name]]
```

Command Class

Action

Syntax Rules

1. The keywords and operands associated with OUTSIDE specify the range to exclude from traps. Their forms and definitions are described in Section 1.13.

2. The operands address-1 and address-2 can be either an absolute address or the base-index-displacement form of the address. If the base-index-displacement form is used, address-1 and address-2 are not re-evaluated unless the trap is deleted and then recreated.

General Rules

1. OUTSIDE does not include the range boundaries; that is, the trap is not taken on either marked boundary statement, but is taken on any statement outside of the boundaries.

2. OUTSIDE requires either operands or marked lines to be executed.

3. If marks are used, the range covers all statements excluded by the marks. There must be statement number information for both the first and last marked lines.

4. If no operands are specified, OUTSIDE applies to the range of statements in the current section that are not marked. If no statements are marked, OUTSIDE with no operands applies to the statement at which the cursor is positioned.

5. If the statement-id form is used, the range includes every instruction from the first instruction of statement-id-1 to the last instruction of statement-id-2, inclusive.

6. If the ADDRESS form is used, the range includes every instruction from the instruction at address-1 to the instruction at address-2, inclusive.

7. If the OFFSET form is used, the range includes every statement from the hex-offset-1 to the hex-offset-2, inclusive.

8. If the operand section-name is not specified, the current section name is used.

9. Statement-id-1 must be less than statement-id-2; address-1 must be less than address-2 and both addresses must be halfword aligned; hex-offset-1 must be less than hex-offset-2 and both offsets must be halfword aligned.

Example

1. OUTSIDE 20 30

   The Debugger traps all statements outside the range of 20 to 30.

POSITION

Description

The POSITION command displays the current cursor position on the message line. The current cursor position consists of the line number and column number of the cursor within the current window. That information remains displayed until the next command is issued.

Format

```
POSITION
```

Command Class

Informative

Related Command

STATUS

PREVIOUS

Description

The PREVIOUS command scrolls the current window display backward a specified number of lines.

Format

```
PREVIOUS [n]
```

Command Class

Action

Syntax Rule

1. The operand n indicates the number of lines to scroll backward.

General Rules

1. If no operand is specified, the default for n is the number of lines minus 2 if five or more lines are visible in the window, or the number of lines if less than five lines are visible in the window.

2. When PREVIOUS is assigned to a PF key, operands cannot be changed during the operation.

Related Commands

FIRST, GOTO, LAST, LEFT, NEXT, RIGHT

PRINT

Description

The PRINT command creates a print file from a marked range of lines, or from the entire contents of the current window, and prints the file. To create a print file from a portion of the current window, first mark the range of lines to print by executing the MARK command. Then execute PRINT with the appropriate file specification. The resultant print file contains the marked range.

To create a print file that contains the entire contents of the current window, execute the PRINT command either with no lines marked or with all lines marked. The resultant print file contains the entire contents of the window.

Format

```
PRINT [[[volume.] library.] file]
```

Command Class

Action

Syntax Rule

1. The volume, library, and file operands specify the file to be printed.

General Rules

1. If the library and/or volume specifications are omitted, the spool volume, and the spool library are used.

2. If the file specification is omitted, a system file name is assigned.

3. The volume, library, and file names must be separated by periods; no embedded spaces are allowed.

4. If no lines are marked, PRINT creates a print file that contains the entire contents of the current window.

Related Command

MARK

PROCEDURES

Description

The PROCEDURES command displays in the data window a list of all subprogram names in all code sections of the subject program. The list does not contain subprograms from code sections that were not included in the Symbolic section of the subject program by the VS translators. For a programming language such as PL/I, which allows statically nested subprograms, any subprograms that are internal to another subprogram are indented by one blank character.

Because not all translators provide symbolic support for subprogram names, some sections will not have subprogram information. However, each section at the minimum contains a line of text indicating the source language and runtime code range for that section.

Format

```
PROCEDURES  [program-name]
```

Command Class

Informative

General Rule

1. If no subprograms exist, the Debugger creates a display conveying that information.

Example

In the following example, the subject program consists of a PL/I object module and a BASIC object module linked together. Subprogram INTINTPROC in the PL/I section, is contained in subprogram INTPROC, which is in turn contained in subprogram OADDRSTAT. The BASIC section has no subprogram information.

Command: PROCEDURES

Displays:

```
PROCEDURES =
Section #ADDRESS
Source language is PL/I code range is 100008 to 1012C0
 OADDRCODE
 OADDRSTAT
  INTPROC
   INTINTPROC
 OADDRACON
 OADDRIMM
 OADDRAUTO
 OADDRPARM
Section FDTOCH
Source language is BASIC code range is 1012C0 to 1020E8
Procedure information is not available for this section
```

PROGRAMS

Description

The PROGRAMS command displays in the data window a list of all the SSLs currently mapped with the main program. This list also displays all files mapped in via the MSMAP system routine.

The PROGRAMS command is supported in Version 1.05 (or greater) of the debugger.

The PROGRAMS command is described in detail in Section 6.5.

Format

```
PROGRAMS
```

Command Class

Informative

Example

Command: PROGRAMS

Displays:

```
Programs =
Main program:
   WORK.TSTRUN.DEBTEST
SSL aliases:
   STRINGPACKAGE
MSMAPed programs:
   LIBPAK.MATHRTM.MATHLIB

CodeSections for STRINGPACKAGE =
#APPEND
```

RECALL

Description

The RECALL command recalls a command that was entered at the command line and displays it in the command line.

When the HISTORY command is set to ON, RECALL displays the nth command (from the beginning), or the most recent command containing "text," or, if neither n nor text is specified, the most recently entered command. When executed without an operand several times in a row, the RECALL command moves backward through the history list, displaying each prior command until the first command is displayed.

RECALL pertains to

- Only commands entered at the command line when HISTORY ON is in effect
- All commands except ASSIGN, DEFINE, HISTORY, and RECALL

When the command is displayed, you can then execute it, with or without modifications, or erase it from the command line.

Format

```
RECALL [n]
       [text]
```

Command Class

Action

General Rule

1. RECALL does not recall commands executed by PF keys.

Related Command

HISTORY

REGISTERS

Description

The REGISTERS command displays, in the data window, the values of all general registers of the subject program.

Format

REGISTERS

Command Class

Informative

Example

1. Command: REGISTERS

   Displays:

```
Registers =
(R0)  002E4D9C 002E3F24 00000045 30000000
(R4)  802E4B6C 00160DA2 00248F1C 00000044
(R8)  00000044 00000110 002E3F28 002EE708
(RC)  00130F68 40038816 002E5020 002E3F24
```

RIGHT

Description

The RIGHT command moves the current window display the specified number of columns to the right. RIGHT is used alternately with the LEFT command to view a window display that is wider than the physical width of the screen. You use RIGHT to display columns to the right of the limit, and LEFT to scroll back in the opposite direction.

Format

```
RIGHT [n]
```

Command Class

Action

Syntax Rule

1. The operand n indicates the number of columns to move the display to the right. Any value is accepted; values less than 1 or greater than the screen width are truncated accordingly.

General Rules

1. RIGHT with no operand displays the text so that the rightmost margin is visible.

2. When RIGHT is executed by PF key, operands cannot be changed during the operation.

3. When the rightmost margin is displayed on the window, RIGHT no longer has any effect when executed.

Related Command

LEFT

ROW

Description

The ROW command positions the cursor on a specified row within the current window. Typically, ROW is used in conjunction with the COLUMN command to move the cursor to a specified row within a designated column.

Format

```
ROW [n]
```

Command Class

Action

Syntax Rule

1. The operand n indicates the row at which you want to position the cursor. Any value for n that is less than 1 defaults to 1; any value for n that is greater than the width of the window defaults to the window width limit.

General Rules

1. ROW operates with the cursor positioned in the current column.

2. If ROW is executed with no operands, the default for n is 1.

Related Commands

COLUMN, CURSOR

SCREEN

Description

The SCREEN command redisplays the subject program screen until you press ENTER or any PF key.

Format

```
SCREEN
```

Command Class

Action

General Rules

1. When SCREEN is executed, the entire Debugger screen, including the top three lines, is replaced by the subject program's screen. The subject program's screen appears exactly as it is presented by the subject program.

2. The Debugger Workstation screen is redisplayed by pressing ENTER or any PF key (no commands assigned to a PF key are executed).

SEARCH

Description

The SEARCH command searches for an instance of a specified string or pattern. If text is specified it becomes the new find-string, otherwise the previous find-string is used. SEARCH searches from the current cursor position forward, or backward, for an instance of the find-string. When an instance is found, the cursor is positioned on the first character of it. If no instance is found, a message to that effect is displayed and the file is not scrolled.

If the CASE EXACT command is in effect, an instance of the string must exactly match the find-string with regard to the capitalization of the letters. If CASE ANY is in effect, letters are compared without regard to capitalization.

If the MATCH ON command is in effect, the find-string is interpreted as a pattern. It is compared to strings in the text according to pattern matching rules (refer to Section 4.4.3). If MATCH OFF is in effect, strings in the text are compared to the find-string just as it is.

Format

```
SEARCH [FORWARD]  [text]
       [BACKWARD] [text]
```

Command Class

Action

Syntax Rules

1. The operand FORWARD specifies a search forward from the current cursor position.

2. The operand BACKWARD specifies a search backward from the current cursor position.

General Rules

1. SEARCH with no operand performs the same function as SEARCH FORWARD.

2. When SEARCH is assigned to a PF key, operands cannot be changed during the operation.

3. If the search string is not found, no action is taken.

Related Commands

CASE, FIND, LOCATE, MATCH

SECTION

Description

The current section is the code section at which control is currently paused. The SECTION command changes the current code section to the named code section of the program listing, making that section the current section. Note that section names are eight characters long.

Format

```
SECTION [program-name/]section-name [statement-id]
```

Command Class

Set-and-Query

Syntax Rules

1. The operand section-name indicates the name of a code section in the subject program.

2. The operand statement-id indicates the ordinal number of a program statement.

General Rules

1. An active section is a section that is currently on the call chain.

2. If statement-id is specified and the section is not active, the listing is positioned such that the specified statement is at the top of the window.

3. If statement-id is not specified, and the specified section is active, the most recent active statement in the section appears at the top of the window.

4. If statement-id is not specified, and the specified section is not active, the first statement of the section appears at the top of the window.

5. If SECTION is executed with no operand, the name of the current section is displayed on the command line.

Examples

1. SECTION COBDEMO

   If COBDEMO is active, the most recent active statement in the section appears at the top of the listing window.

   If COBDEMO is not active, the first statement of the section appears at the top of the listing window.

2. SECTION COBDEMO 40

   The Debugger moves statement 40 in COBDEMO to the top of the Listing Window and displays a portion of the program listing from that point.

SET

Description

The SET command allows you to query and reset the default values for MEMORYLENGTH and the VARECHO flag during a debugging session.

Format

```
SET MEMORYLENGTH [set value]
    VARECHO [set value]
```

Command Class

Set-and-Query

General Rules

1. The default value for MEMORYLENGTH, initially set to 4, can be reset to any integer between 1 and 4096.

2. The default for the VARECHO flag is initially set to OFF and can be reset to ON to echo the variable commands on the command line.

3. The current values for MEMORYLENGTH or VARECHO can be displayed on the command line by issuing the SET command without specifying a new default value.

4. To reset a default value, enter SET followed by either MEMORYLENGTH or VARECHO and the corresponding value and press ENTER.

SNAPSHOT

Description

The SNAPSHOT command creates a print file that contains the image of the Debugger Workstation screen.

Format

```
SNAPSHOT [file path]
```

Command Class

Action

General Rule

1. You can specify only the file name or the library and file names. The Debugger uses the library and volume provided by your usage constants for spoolib and spoolvol.

   You can also omit the file name altogether. In that case, a default system name, spoolvol.spoolib.@DEBnnnn, is automatically generated. The nnnn of @DEBnnnn represents decimal digits.

STACKTRACE

Description

The STACKTRACE command displays in the data window the addresses of the currently active JSI, SVC, and LINK instructions. For the current link level, STACKTRACE displays the absolute addresses, code sections and offsets, and line numbers when possible.

Format

```
STACKTRACE  [ALL]
            [ n ]
```

Command Class

Informative

Syntax Rules

1. The operand ALL indicates a list of all active calls, including all link levels above the current link level.

2. The operand n indicates the number of stack frames that you want to display. This feature is useful when debugging large programs where the number of active stack frames can be very high, requiring you to page through several displays to view specific stack frames. The most recent stack frames are always displayed.

General Rule

1. If ALL is not specified, only the active calls at the current link level are shown.

Example

```
Command:  STACKTRACE

Displays:

STACKTRACE =
Procedure called from 127D92
Statement 423 Offset 00168A in section #EDITOR

 SaveArea at 3E6BA0 contains:
(R0)    803E5230 003E6CC4 00000FF3 FFFFFFF3
(R4)    C03E6D1C 003E6D08 00000001 003F0D83
(R8)    00000000 003E6C5B 00000000 003E6C04
(RC)    003F0AD0 0012BFE4 003FF188

Procedure called from 126836
Statement 92 Offset 00012E in section #EDITOR

 SaveArea at 3E6CC4 contains:
(R0)    803E5230 003EA5AC 003E6D08 00000000
(R4)    003E9DB0 003FD4C4 003FD4D0 003FD529
(R8)    003FD512 003FD4B4 803FD52B 003E6D14
(RC)    003F0AD0 0012BFE4 003FF188

Program NEWDEBUG linked from 10EA12

 SaveArea at 3E6DD4 contains:
(R0)    803E5230 00000000 00000000 003FD508
(R4)    003FD52C 003FD4C4 003FD4D0 003FD529
(R8)    003FD512 003FD4B4 803FD52B 003FD498
(RC)    003FF1C0 0010E980 003FF188

Procedure called from 100B2A

 SaveArea at 3FD7E0 contains:
(R0)    003E5230 00201011 003FD824 003FDF30
(R4)    003FDF38 00100B20 003FDF40 003FDFE8
(R8)    00000002 00000054 003FDFE0 003FD828
(RC)    003FDC78 00100D14 B0100280

Procedure called from 100140

 SaveArea at 3FDB9C contains:
(R0)    003E5230 00100040 003FDC60 00100008
(R4)    00000110 0002FBD0 000262B0 00011030
(R8)    00000001 0001E258 003FDBE0 003FDC34
(RC)    003FFF80 00100040 003FDC30
```

STATE

Description

The STATE command displays in the data window the value of the components of the subject program's state (other than register values).

Format

STATE

Command Class

Informative

Example

Command: STATE

Displays:

```
State =
PCW = 7012A3B8 00000000

Program Address =  12A3B8 Interrupt Code = 70

Condition Code                          =  00
Fixed Point Overflow Interrupt Mask     = 0
Decimal Overflow Interrupt Mask         = 0
Exponent Underflow Interrupt Mask       = 0
Significance Loss Interrupt Mask        = 0

Process Level          = 0
Previous Address       = 000000
Stack Backlink Pointer = 3FE884
Program Static Base    = 3FF1AC
```

STATICSECTIONS

Description

The STATICSECTIONS command displays in the data window a list of all static sections in the subject program.

Format

```
STATICSECTIONS [program-name]
```

Command Class

Informative

Example

```
Command:  STATICSECTIONS

Displays:

COBDEMO
```

STATUS

Description

The STATUS command displays user-requested information about the current debugging session. That information refers to the current window only. STATUS can take several operands that indicate the desired contents of the status line.

Format 1

```
STATUS [CODESECTION] [COLUMNS] [CURSOR] [LINES] [MARK] [MEMORY]
       [NAME] [PCW] [PROGRAM] [STATEMENT] [TASKID] [TIME]
       [USERID] [WINDOW]
```

Format 2

```
STATUS OFF
```

Command Class

Set-and-Query

Syntax Rules

1. The operand CODESECTION indicates the name of the current code section.
2. The operand COLUMNS indicates the visible column numbers of the file.
3. The operand CURSOR indicates the cursor position after either the SEARCH or LOCATE command has been successfully executed.
4. The operand LINES indicates the lines of the current window that are currently visible.
5. The operand MARK indicates the lines that are currently marked.
6. The operand MEMORY indicates the amount of memory available for debugging.
7. The operand NAME indicates the name of the current window.
8. The operand OFF suppresses the status display for the current window.
9. The operand PCW provides Program Control Word information.
10. The operand PROGRAM indicates the program name.
11. The operand STATEMENT indicates the number of the statement at which control is paused.
12. The operand TASKID indicates the task identification number of the subject program.

13. The operand TIME indicates the current time.

14. The operand USERID indicates the user ID.

15. The operand WINDOW indicates the current window number.

General Rules

1. The default operands for STATUS are CODESECTION, STATEMENT, and PCW.

2. If STATUS is entered with no operands, the current operands are displayed.

3. When STATUS is assigned to a PF key, operands cannot be changed during the operation.

4. You can specify, in any order, as many operands as will fit on the command line.

5. Status items are not displayed if their contents are not meaningful or have undefined values.

6. If you are displaying more than one window, the status display for the window nearest the top of the Window section refers to that window. Status lines for lower windows are displayed as the first line of the window.

Example

When STATUS NAME CODESECTION STATEMENT is in effect for the listing window, the following information is a sample of what could appear on the status line of that window:

```
Listing  CodeSection COBDEMO  Statement# 49
```

STEP

Description

The STEP command executes the subject program in increments. Incremental execution means that a certain number of program statements (or instructions) are executed and then the program is automatically paused, with control returned to the Debugger.

Format

```
STEP [INSTRUCTION] [n]
```

Command Class

Action

Syntax Rule

1. The INSTRUCTION operand specifies that the operand n represents the number of machine instructions in the subject program to be executed before execution is paused. If INSTRUCTION is not specified, n represents the number of source statements to be executed before execution is paused.

General Rules

1. The specified number of statements or instructions is executed immediately upon execution of STEP. When those statements or instructions have run, execution halts and control returns to the Debugger.

2. When STEP n is executed, and another trap is taken before n statements (or instructions) have been executed, control returns to the Debugger. A message is displayed that the trap has been taken and the STEP n command is terminated.

Examples

1. STEP 2

   The subject program resumes execution for two source program statements and pauses execution following the second statement. Control is then passed to the Debugger.

2. STEP INSTRUCTION 3

   The subject program resumes execution for three machine instructions and pauses execution following the third instruction.

TRAINING

Description

The TRAINING command assists you in the creation of minimally valid command strings. When you execute the TRAINING command with no operands, a selection screen appears listing each of the Debugger commands. A command is chosen by tabbing the cursor to the select field located to the left of the field and pressing the ENTER key.

A fill-in-the-blank template is displayed for the command, including a description of the command and its operands. Once the desired operands have been filled in, you press ENTER and the Debugger creates a command line with the entered information.

The Training Facility is described in detail in Section 1.9.

Format

```
TRAINING [command-name]
```

Command Class

Action

Syntax Rules

1. The command-name operand enables you to skip past the Debugger Command screen directly to the desired command.

2. The command-name operand can be abbreviated to the shortest unique abbreviation. If any ambiguous abbreviations are encountered, the Debugger will select the first command containing the abbreviation. Invalid abbreviations cause the Debugger to display the Debugger Command screen.

Related Command

HELP

TRAPS

Description

The TRAPS command accesses the trap window.

Format

TRAPS

Command Class

Action

Related Commands

DATA, DISPLAY, FRAME, LISTING, MENU, WINDOW

VARIABLE

Description

The VARIABLE command displays specified program variables in the data window, and monitors the value of each variable so that the current value is always displayed.

Format

```
VARIABLE [identifier]
```

Command Class

Informative

Syntax Rule

1. The operand identifier can be any character string that comprises a valid data name in the source language of the current code section. Identifiers include pointers, subscripts, containing structure names, containing subprogram names, and containing block names.

General Rules

1. The identifier operand must be defined in the current section.

2. Use of VARIABLE can be simplified by positioning the cursor at the first character of an identifier. If VARIABLE is executed with no operand but the cursor is positioned at that location, that identifier is used as the operand. The identifier selected from the cursor position includes all characters up to the first blank encountered.

3. If VARIABLE is assigned to a PF key, and is executed through that key, it has the following special behavior:

   If the command line is not blank, the text within the line is the operand for the VARIABLE command.

   Thus, an alternate way to display a variable is to enter its name on the command line (or position the cursor at the variable), then press the PF key to which the VARIABLE command is assigned.

4. If the value displayed by VARIABLE is too long to fit on one line, it is formatted on successive lines.

5. Automatic variables require that the procedure or function name precede the variable name. For example, to display automatic variable K in procedure #MAIN, the command would be

   ```
   VARIABLE MAIN.K
   ```

Examples

1. Command: `VARIABLE SUB2`

   Displays:

   ```
   SUB2 = +0
   ```

2. Command: `VARIABLE FORECAST`

   Displays:

   ```
   FORECAST = SUNLIGHT AT DAWN
   ```

3. Command: `VARIABLE MAIN.K`

   Displays:

   ```
   MAIN.K = 2
   ```

VERSION

Description

The VERSION command displays on the message line, the version number of the currently installed Debugger.

Format

```
VERSION
```

Command Class

Informative

WINDOW

Description

The WINDOW command accesses an existing window. The Debugger numbers windows sequentially in the following way:

- The trap window is assigned the number 1.
- The data window is assigned the number 2.
- The listing window is assigned the number 3.
- Additional windows (the display window or the menu window) are numbered sequentially starting at 4, in their order of access.

If an additionally accessed window is closed, the following window (if present) moves up in the sequence to take the place of that window and is assigned that number. The trap, data, and listing windows cannot be closed.

To access a specific window, execute WINDOW followed by the number of the window to be displayed.

Format

```
WINDOW [n]
```

Command Class

Action

Syntax Rule

1. The operand n represents the number of the window.

General Rules

1. WINDOW affects all windows.
2. When WINDOW is assigned to a PF key, operands cannot be changed during the operation.
3. WINDOW with no operand displays the next sequentially numbered window. If the last window is the current window when WINDOW with no operand is executed, the first window (trap window) is displayed.

Related Commands

DATA, DISPLAY, FRAME, FULL, LISTING, MENU, TRAPS

APPENDIX A
TRANSITION FROM THE PREVIOUS DEBUGGER TO THE NEW DEBUGGER

A.1 INTRODUCTION

This appendix is intended for VS users who are familiar with the previous Debugger. It describes how to use the new Debugger to perform the tasks that you used to perform with the previous Debugger. The information is organized on a screen-by-screen basis according to the screens of the previous Debugger.

Note: *If you are a new or infrequent Debugger user, you may want to use Easy mode for simplified Debugger operation. Chapter 3 describes Easy mode in detail.*

A.2 MAIN SCREEN

Upon entry, the previous Debugger always displayed a menu which listed the main debugging functions. This menu is referred to as the Main screen in this appendix. From the Main screen you could perform several debugging functions and access other debugging menus for trap management and the Inspect and Modify functions.

A.2.1 Continue

Previous Debugger

PF1, the Continue key, caused the user program to resume execution.

New Debugger

The CONTINUE command causes the user program to resume execution. It also accepts operands which specify an alternate program restart address (absolute addresses, statement numbers, or offsets within a named section). This command is assigned to PF14.

A.2.2 Scrolling

Previous Debugger

PF2 and PF3, the Previous and Next keys, caused the listing, displayed in the top seven lines of the screen, to scroll five lines backward or forward.

New Debugger

Scrolling the listing window, and other Debugger windows, is accomplished by using the PREVIOUS and NEXT commands. These commands take optional numeric operands which specify the number of lines to scroll. If no operands are specified, the selected window is scrolled n lines, where n is 2 less than the total number of lines displayed. Several scrolling commands are assigned to PF keys, as follow:

PF4	PREVIOUS
PF5	NEXT
PF6	PREVIOUS 1
PF7	NEXT 1

In addition, the new Debugger also provides the following two commands to scroll the listing so that the first line of the listing appears at the top of the window or the last line of the listing appears at the bottom of the window:

PF2	FIRST
PF3	LAST

A.2.3 Traps

Previous Debugger

PF4, the Trap key, put the user in Trap mode, which displayed the Trap screen.

New Debugger

Refer to Section A.3 for a discussion of the Trap screen and the corresponding functions in the new Debugger.

A.2.4 Inspect & Modify

Previous Debugger

PF5, the Inspect & Modify key, put the user in Inspect & Modify mode, which displayed the Inspect & Modify screen.

New Debugger

Refer to Section A.4 for a discussion of the Inspect & Modify screen and the corresponding functions in the new Debugger.

A.2.5 Select Section

Previous Debugger

PF6, the Select Section key, allowed the user to switch the section being examined, which changed the contents of the listing window and the defaults supplied on other screens.

New Debugger

The SECTION command allows the user to switch the default section, changing the contents of the listing window accordingly. The SECTION command also takes an optional operand which indicates the line number at which to position the listing, defaulting to the most recent display of that section, if previously displayed.

A.2.6 Dump

Previous Debugger

PF13, the Dump key, took a task dump of user memory and stored it in a print file in the user's spool library, or in the library #PRT on the system volume, if the spool library was not set. It did not display the name of the file created, which was of the form DUMPxxxx.

New Debugger

The DUMP command creates a Dump just as the previous Debugger did. Similarly, it does not display the name of the Dump file created, and the file name takes the same form, DUMPxxxx.

A.2.7 Print Program Screen

Previous Debugger

PF14, the Print Program Screen key, created a print file of the contents of the user program screen in the user's print library (or #PRT if not set).

New Debugger

This function is not directly available in the new Debugger. It may be invoked by pressing HELP to go to the modified Command Processor menu, and then pressing PF14.

A.2.8 Print Debug Screen

Previous Debugger

PF15, the Print Debug Screen key, created a print file of the contents of the Main screen in the user's print library (or #PRT if not set).

New Debugger

The SNAPSHOT command creates a different type of print file than the previous Debugger created, but it does include all the contents of the Debugger screen.

A.2.9 Cancel Processing

Previous Debugger

PF16, the Cancel Processing key, caused the user program to be cancelled, and to terminate the debugging session.

New Debugger

The CANCEL command, assigned to PF32, behaves exactly as PF16 did in the previous Debugger.

A.3 TRAP SCREEN

When you pressed PF4 on the Main screen, the Debugger entered Trap mode and displayed the Trap screen. This screen offered three different types of traps to set, either singly or jointly. The three types of traps were

- Breakpoint (address) trap
- Single step trap
- Memory modification trap

The new Debugger enables you to set the same types of traps with the commands BREAK, STEP, and MODTRAP.

A.3.1 Breakpoint (Address) Trap

Previous Debugger

A breakpoint trap was specified by filling in one or more fields of the Breakpoint Trap section of the Trap screen. Four fields were associated with breakpoints: Line Number, Section, Offset, and Count.

Line Number	This field was only used when the section specified in the Section field contained symbolic statement information. Specifying a line number within a section caused a breakpoint trap to be set on the address of the first instruction of that statement.
Section	This field specified the section within which the offset of Line Number was to be used. This was restricted to code sections and had to be used with Line Numbers. Offsets were allowed.
Offset	This field was used to specify either an offset within the section specified in the Section field, or, if that field was blank, an absolute address.
Count	This field was used to specify the count of the breakpoint trap, that is, how many times the breakpoint was hit before the trap was taken.

New Debugger

The BREAK command set a breakpoint trap on a symbolic statement, at an absolute address, and at an offset within a section.

To set a ***breakpoint on a symbolic statement,*** the command is

```
BREAK nnn ssssssss
```

where nnn is the line number and ssssssss is the section name. Note the section name need not be specified if it is the current section, just as with the previous Debugger.

To set a ***breakpoint at an absolute address,*** the command is

```
BREAK ADDRESS aaaaaa
```

where aaaaaa is an absolute address specification in either a 6-digit hex value or a base-index-displacement value (as in the Assembler).

To set a ***breakpoint at an offset within a section,*** the command is

```
BREAK OFFSET xxxxxx ssssssss
```

where xxxxxx is the offset and ssssssss is the section name. Note the section name need not be specified if it is the current section, just as with the previous Debugger.

The new Debugger lets you set a breakpoint trap very easily if there is a listing displayed in the listing window. Simply position the cursor on the statement (or instruction, if the source language is Assembler), and press the PF key to which BREAK has been assigned (PF11 is the default).

A.3.2 Single-Step Trap

Previous Debugger

The single-step trap executed either a fixed number of high level language statements or machine level instructions. There were two fields associated with single stepping: Symbolic Statement (S) or Instruction (A), and Count.

S or A	This field was used to select either symbolic statement (S) or instruction level (A) traps. If "S" were specified and the source language were Assembler, the value would be automatically changed to "A" since symbolic statements at the Assembler level are instructions.
Count	This field was used to specify the number of steps, that is, the number of statements or instructions that were to be executed before stopping.

New Debugger

The STEP command not only sets both types of single step traps, but also resumes program execution immediately. With the previous Debugger, you had to return to the Main screen before resuming program execution.

To step n symbolic statements, the command STEP n is used. To step n instructions, the command STEP INSTRUCTION n is used.

Two single-step STEP commands are assigned to PF keys as follow:

```
PF10  STEP 1
PF26  STEP INSTRUCTION 1
```

A.3.3 Memory Modification Trap

Previous Debugger

The memory modification trap caused the program to trap when one specified byte of storage changed value. That byte could be specified either as an offset within a static section or as an absolute address. Two fields were associated with memory modification traps: Section and Offset.

Section	This field specified either the static section to be used as a base for the offset specified by the Offset field, or, if blank, indicated that the Offset field was to be treated as an absolute address.
Offset	This field specified the offset within the section specified by the Section field, or, if the Section field were blank, the absolute address of the byte to be monitored.

New Debugger

The MODTRAP command sets modification traps, both on memory and on general registers. The size of the value is no longer restricted to one byte, and symbolic (data name) traps are also available.

To set a ***modification trap on a specific address,*** the command is

```
MODTRAP ADDRESS aaaaaa !!!!
```

where aaaaaa is an absolute address and !!!! is the number of bytes, which is optional. The default is 4.

To set a ***modification trap on an offset within a section,*** the command is

```
MODTRAP OFFSET xxxxxx ssssssss !!!!
```

where xxxxxx is a hex offset value, ssssssss is a static section name, and !!!! is the number of bytes. The static section name and the number of bytes are both optional. If the static section name is omitted, it defaults to the current section (the same as the previous Debugger). If the number of bytes is omitted, it defaults to 4.

To set a ***modification trap on a program variable,*** the command is

```
MODTRAP VARIABLE vvvvvvv
```

where vvvvvvvv is a variable name in the user program, specified in the same manner as on the Inspect & Modify screen (for more information, refer to the next section).

A.4 INSPECT & MODIFY SCREEN

When you pressed PF5 on the Main screen, the Debugger entered Inspect & Modify mode and displayed the Inspect & Modify screen. This screen enabled the display of program variables. You could specify the variable, which was then displayed on the same screen. However, the menu was modified to provide functions for changing the form of the display and modifying the variable.

The Inspect & Modify screen also provided access to other screens to display the contents of memory, general and floating-point registers, and the PCW.

A.4.1 Displaying Variables

Previous Debugger

After you specified the Name and Section fields, the ENTER key caused these fields to be used to obtain the value of the program variable. Either the value was displayed on the screen with the modified menu (refer to A.5) or an error condition was reported.

New Debugger

(Refer to Section A.5.1.)

A.4.2 Inspecting Memory

Previous Debugger

PF10 displayed the Inspect & Modify Memory screen (refer to Section A.6).

New Debugger

(Refer to Section A.6.)

A.4.3 Inspecting Registers

Previous Debugger

PF11 displayed the Inspect & Modify Registers screen (refer to Section A.7).

New Debugger

(Refer to Section A.7.)

A.4.4 CALL/LINK/SVC Trace

Previous Debugger

PF12 displayed the CALL/LINK/SVC Trace screen which showed the save area associated with the most recent call or SVC issued. From this screen, pressing ENTER displayed the next save area, and so on, until the last save area was displayed. PF1 was used to return to the Inspect & Modify screen. Only machine-level save information was displayed.

New Debugger

The STACKTRACE command creates a display in the data window containing all save areas, with symbolic information where available (line numbers in sections rather than absolute addresses, offsets for nonsymbolic or Assembler sections).

A.4.5 Display Program Screen

Previous Debugger

PF14 caused the user program screen to be displayed, exactly as it appeared when the program was last executing. Pressing any PF key returned the display to the Inspect & Modify menu.

New Debugger

The SCREEN command displays the user program screen. You can return to the Debugger by pressing any key. This command is not assigned to a PF key.

A.5 INSPECT & MODIFY SCREEN WITH MODIFIED MENU

When you pressed ENTER from the Inspect & Modify screen and a variable was displayed, it was displayed on the same screen but the menu choices changed, enabling you to manipulate the variable. From this menu, you could

- Display the variable in its data type format or in hexadecimal notation
- Modify it
- Display portions of it, if the variable were too large to display on the screen

A.5.1 Displaying Variables

Previous Debugger

PF6 enabled you to display the variable either in its data type format or in hexadecimal notation.

New Debugger

The HEX command alternately toggles the selected variable display between hex and symbolic format. Since the data window may contain more than one variable display, one specific entry must be selected. This can be done with either the MARK command or by positioning the cursor.

A.5.2 Modifying Variables

Previous Debugger

PF7 made the display modifiable.

New Debugger

Modifying variables is handled differently in the new Debugger. Rather than entering a Modify mode to make changes to variables, the entries in the data window are always modifiable. However, any modifications that you make are not read until you execute the ALTER command.

Only one variable (or other modifiable entry) can be changed at a time. First, you use the cursor or the MARK command to select an entry, then you enter a new value and execute the ALTER command. ALTER is assigned to PF24.

Note that the ALTER command does ***not*** put you in Alter mode, but instead acts on the values that you typed into the selected entry ***before*** you execute the ALTER command (either by PF key or at the command line).

A.5.3 Scrolling Variables

Previous Debugger

PF4 and PF5 enabled you to display portions of a variable that was too large to be displayed at one time.

New Debugger

The new Debugger creates entries in the data window for the whole variable specified. You use the scrolling commands PREVIOUS and NEXT in the data window to view whatever portion of the variable interests you.

A.6 MEMORY SCREEN

The Memory screen enabled you to view program memory, in code, data, or certain protected segments, and to modify the data segment.

A.6.1 Displaying Program Memory

Previous Debugger

After filling in the various fields for the Memory screen, pressing ENTER updated the display to reflect those values. The fields of the Memory screen were

OFFSET	Hex offset from specified base address (SECTION, BASEADDR, or none)
LENGTH	Hex number of bytes displayed (maximum is hex 100 (decimal 256))
SECTION	Section name used as a base address; if specified, the BASEADDR field cannot be used
BASEADDR	The hexadecimal base address; if specified, the SECTION field cannot be used

There were three different methods to specify the starting address to be displayed:

1. Base address only (either absolute address or contents of a register)

2. Base address (either absolute address or contents of a register) plus offset

3. Section name (code or static) plus offset

The first method was achieved by filling in either the absolute address or Rx (where x was 0-9, A-F) in the BASEADDR field and pressing ENTER. The second was achieved the same way, with the additional specification of a hex value in the OFFSET field. The third was done by specifying a section name in the SECTION field rather than using the BASEADDR field.

New Debugger

The MEMORY command allows you to display portions of program memory in the data window. Rather than filling in a SIZE field, the size is the last operand specified in the command. If not specified, the size defaults to 4. The three methods of specification that were available in the previous Debugger are also available in the new Debugger, as follows:

1. To specify a base address only, enter the command

   ```
   MEMORY addr ref [length]
   ```

   where addr ref is either an absolute address or a register (Rx) as before, or a base-index-displacement combination which takes the form ([disp]([ix,]b)). The base-index-displacement specification is a new feature.

2. To specify a base address plus offset, enter the command

   ```
   MEMORY OFFSET hex val (addr ref)[length]
   ```

   where hex val is a hex value and addr ref is as described in method 1.

3. To specify an offset within a section, enter the command

   ```
   MEMORY OFFSET hex val section name [length]
   ```

where hex val is a hex value and section name is the name of either a code or a static section.

A.6.2 Modifying Program Memory

Previous Debugger

After displaying a portion of memory, pressing PF3 caused the mode to change to Modify Memory mode. You could then type the desired values and press ENTER. Pressing ENTER caused the data to be accepted and exited from the Modify Memory mode.

New Debugger

Modifying memory displays is similar to modifying variable displays. All memory displays in the data window are modifiable, but the screen is not read unless the ALTER command is executed. The target entry of the ALTER command is selected either by marking an entry or by using the cursor (if no entry is marked).

Note that the ALTER command does *not* put you in Alter mode, but instead acts on the values typed into the selected entry *before* the ALTER command is executed (either by PF key or at the command line).

A.7 REGISTERS SCREEN

The Registers screen displayed the contents of the general registers, floating-point registers, and the PCW. It also enabled you to modify these entities.

A.7.1 Modifying Register Contents

Previous Debugger

Pressing PF3 caused the mode to change to Modify Register mode where you could alter the values of the general and floating-point registers. You did this by typing the desired values in the respective fields and pressing ENTER.

New Debugger

The REGISTERS and FLOATREGISTERS commands allow you to display the contents of general and floating-point registers in the data window. Modifying registers is similar to modifying variable and memory displays. Register displays (general and floating-point) in the data window are modifiable, but the screen is not read unless the ALTER command is executed.

Note that there are separate entries for the general and floating-point registers. The target entry of the ALTER command is selected either by marking an entry or by using the cursor (if no entry is marked).

Note also, that the ALTER command does *not* put the user in Alter mode, but instead acts on the values typed into the selected entry *before* the ALTER command is executed (either by PF key or at the command line).

A.7.2 Modifying the PCW

Previous Debugger

Pressing PF4 caused the Modify PCW screen to be displayed where the contents of the PCW and certain flags were presented in modifiable fields. Entering the desired values and pressing ENTER changed the values accordingly, and returned the user to the Registers screen.

New Debugger

The STATE command allows you to display the PCW, along with other related information concerning the current program state, in the data window. Modifying the PCW is similar to modifying variable and memory displays and registers. The display of the PCW in the data window is modifiable, but the screen is not read unless the ALTER command is executed. The target entry of the ALTER command is selected either by marking an entry or by using the cursor (if no entry is marked).

Note that the ALTER command does ***not*** put the user in Alter mode, but instead acts on the values typed into the selected entry ***before*** the command is executed (either by PF key or at the command line).

APPENDIX B
GLOSSARY

address

A 6-digit hexadecimal value that describes a program code or data location.

base address

An address that indicates the virtual address of the first byte of a program file.

code section

The section in the program file that contains executable instructions for the program.

command line

The second line of the Control section, which serves as the primary input mechanism to the Debugger. You enter commands on the Command line.

control section

The first three lines of the Debugger Workstation screen. The Control section contains the status, message, and command lines.

count

The number of times that the condition for a trap is to be encountered before the trap is taken.

data value text

Actual data values (not the names).

debugging commands

Commands that enable you to perform debugging actions on program data displayed in windows.

default startup file

A file that contains defaults for PF key assignments and various other options to be used by the Debugger. A system startup file is supplied with the Debugger; you can specify a user startup file to be loaded after the system startup file.

full window format

A format specified by the FRAME command in which each accessed window occupies the entire 21-line Window section.

help text

On-line reference information about the Debugger displayed through the VS INFO utility. To access Help text for the Debugger, execute the HELP command.

hexadecimal notation

A method for expressing numbers in base 16. Digits are 0-9, A-F.

hits

The number of times a trap has been encountered by the subject program. When the Hits value equals the Count value, the trap is taken.

macro

A special command defined by specifying a name of your choice and one or more commands to be assigned to that name. The name and definition together are called a macro; macros are executed in the same way as commands.

menu

A list of the current PF key assignments that is displayed in the menu window.

message line

The third line of the Control section, which serves as the location where the Debugger displays error or informational messages.

partial window

A window that is displayed with a reduced number of lines in order to display one or more additional partial windows.

partial window format

A format specified by the FRAME command in which each accessed window can occupy all or a portion of the Window section, depending on the cursor position when the command to access the window is executed.

prname

The screen label needed in VS Procedure language to qualify references to the fields or PF keys of a particular screen interaction.

procedure

A VS Procedure language program.

program control word (PCW)

A doubleword that contains the address of the next instruction to be executed and various status bits. The PCW is described in more detail in *VS Principles of Operation*.

regular expression

A search string specified by the MATCH command that uses special symbols to indicate the desired type of search.

relocation

The process of adjusting an address referencing value to reflect the movement of the location being referenced due to the context changes between input and output files.

screen management commands

Commands that enable you to perform screen management functions such as scrolling text, searching text, marking lines, and displaying partial windows.

search string

A string that you specify by the FIND or MATCH command that you then look for by the SEARCH or LOCATE command.

section

The basic building block of a program. The Linker composes the output file from sections selected from input files, static subroutine libraries, and shared subroutine libraries. There are two main section types: code sections contain the instructions for the program; static sections contain information required by the operating system to build and initialize memory for the data portion of a program run.

segment

One of two portions which make up a program. The code segment is the instruction portion and the static segment is the data portion. Each segment is composed of zero or more sections of the appropriate section type.

status line

The first line of the Control section, or the header line for windows displayed in partial format, which displays information about the topmost window.

subject program

A program for which you have explicitly requested debugging assistance.

symbolic data

Data that makes symbolic debugging possible. It is inserted in a program file by the compiler.

symbolic name

The program name of a subject program variable.

symbolic reference

Use of a particular symbol in a place other than where it is defined. An internal symbolic reference refers to a symbol defined in the same section. An external symbolic reference refers to a symbol defined in another section.

symbolic support

The Debugger's ability to accept references to program components by their symbolic names.

trap

A stopping mechanism set in the subject program to inspect and modify program variables, and to alter the flow of execution of the program.

trap list

The list of traps displayed in the trap window.

undefined symbol

A symbol that has symbolic references but does not have a corresponding section name or an entry point name in the program.

window

A specified area within the Window section that contains a particular type of program information. For example, the trap window contains trap information.

window context

A collection of information associated with each window. The window context consists of the following attributes that are unique to that window: case setting, current search string, contents of the status display, range of marked lines.

window section

The section of the Debugger Workstation screen in which windows are displayed. The Window section extends from line 4 to the bottom of the Debugger Workstation screen.

APPENDIX C
DEBUGGER ERROR MESSAGES

C.1 DEBUGGER ERROR MESSAGES

This appendix lists the error messages for the Debugger. Each message includes a brief description. In each message, the expression (text) represents the command or text string that the message is about. The command or text string is specified when the message is displayed.

A General Register must be specified.

The MODTRAP REGISTER command was specified without a register operand.

A range of lines must be marked for this command.

The INSIDE or OUTSIDE command was specified without operands, and no lines were marked in the listing window.

A stack frame error has occurred.

An illegal back chain has been detected in the user's call chain.

A subscript value specified is too big.

The integer value specified for a subscript is greater than the upper bound for the array.

A subscript value specified is too small.

The integer value specified for a subscript is smaller than the lower bound for the array.

A variable must be specified.

The MODTRAP VARIABLE command was specified without a variable operand.

Address of indicator <indname> is invalid. Contact your system administrator at once.

This indicator cannot be addressed correctly due to either an internal error, compiler error, or system error.

Address specified is not in user memory.

The address of the variable specified for MODTRAP VARIABLE, or the address specified for MODTRAP ADDRESS is not in user memory.

Address2 may not be less than Address1.

For the INSIDE and OUTSIDE commands, the first address specified must be less than the second.

ALTER is not permitted when paused in privileged code.

The user program is currently executing in privileged code (SVC, DMS, etc.) and therefore cannot ALTER any memory values or components of the program state.

An error occurred while attempting to allocate trap table space.

There is insufficient memory to create further trap entries.

An incorrect number of operands have been used.

The number of operands specified is incorrect for any of the known combinations of operands for this command. An example would be the MODTRAP REGISTER command (which takes either 2 or 4 operands) when specified with 3 operands.

An integer value was expected as a subscript.

A non-integer value was found in a subscript field.

An internal trap error has occurred.

An internal error was detected.

An invalid address was specified.

The specified value for an address contains too many digits.

An invalid base register was specified.

The specified value is not a valid general register specification. Valid values are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, A, B, C, D, E, F, all optionally preceded by the letter R or the letter r.

An invalid character is present in Float Register <n> <text>.

Invalid characters were entered in the specified field of a FLOATREGISTERS display for the ALTER command.

An invalid character is present in General Register <n> <text>.

Invalid characters were entered in the specified field of a REGISTERS display for the ALTER command.

An invalid character is present in <text> at location <text>.

Invalid characters were entered in the specified field of a MEMORY display for the ALTER command.

An invalid character is present in the Condition Code field <text>.

A character specified for this attempted ALTER is illegal for the Condition Code field of the STATE display.

An invalid character is present in the Decimal Overflow field <text>.

A character specified for this attempted ALTER is illegal for the Decimal Overflow field of the STATE display.

An invalid character is present in the Exponent Underflow field <text>.

A character specified for this attempted ALTER is illegal for the Exponent Underflow field of the STATE display.

An invalid character is present in the Fixed Point Overflow field <text>.

A character specified for this attempted ALTER is illegal for the Fixed Point Overflow field of the STATE display.

An invalid character is present in the Program Address field <text>.

A character specified for this attempted ALTER is illegal for the Program Address field of the STATE display.

An invalid character is present in the Significance Loss field <text>.

A character specified for this attempted ALTER is illegal for the Significance Loss field of the STATE display.

An invalid displacement was specified.

The specified value for a displacement value contains an invalid character.

An invalid index register was specified.

The specified value is not a valid general register specification. Valid values are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, A, B, C, D, E, F, all optionally preceded by the letter R or the letter r.

An invalid register was specified.

This message is self-explanatory.

An invalid relational operator was specified.

The specified value is not a valid relational operator. Valid values are: =, ^=, <, >, <=, >=.

An undefined CONVERT error has occurred.

An internal error was detected.

An undefined file open error has occurred. Error code is <n>.

A file could not be opened and the error code is unknown to the Debugger.

An undefined TRAP related error has occurred.

An internal error was detected.

CLOSE is not legal in the <window-name> window.

CLOSE was executed in the listing, data, trap, or easy mode menu window; CLOSE is not a valid command for those windows.

<CodeSection> has no symbolic information.

The specified code section has no symbolic information available.

Comparisons are not allowed in this command.

The MODTRAP VARIABLE command was specified with additional operands, and it is assumed that the user meant to specify a comparison.

Current section is not an RPGII code section.

The INDICATORS command can be executed only when the current code section is an RPG II code section.

Current window is not the data window.

The HEX command can only be executed with the cursor positioned in the data window.

Default length used instead of <text>

The second operand <text> of the MEMORY command could not be converted to a valid length specification, so the default length of 4 was used instead.

DELETE is only legal in the data and trap windows.

DELETE was executed in either the menu, display, or listing window.

DISPLAY requires a file name.

No file name operand was specified for the DISPLAY command.

Extraneous text found past end of complete variable reference.

A complete variable reference (possibly including subscripts) is followed by additional text.

File already in use.

The specified file is currently in use and could not be opened.

File <filespec> does not exist.

The specified file name was not found.

<File name> is invalid.

Either illegal characters or more than eight characters were specified for the file name.

File name is not a valid Debugger command file.

The specified file does not meet the qualifications required for a valid command file.

File or volume is being used exclusively.

The specified file could not be opened because either the volume is mounted for exclusive use, or the file is open for exclusive use.

File or volume possession conflict.

The specified file is currently open in non-shared use by another user.

File specification is invalid.

The file specification is invalid for some unknown reason.

File specified already exists.

This message is self-explanatory.

File <text> is not a valid private startup file.

The specified private default startup file is invalid.

File <text> is not a valid system startup file.

The specified system default startup file is invalid.

<filetype> files may not be printed.

Only consecutive files and print files can be printed. The specified file is a program, word processing, or indexed file.

Help file <helpfilename> is not available.

The Debugger Help file could not be found.

Illegal character detected.

A character specified for this attempted ALTER is illegal for the data type of this variable.

Illegal character detected in comparand field.

The comparand field for the MODTRAP REGISTER command contains an invalid hexadecimal character. Valid characters are: 0 through 15, A through F.

Illegal command detected in record <n> of file <filespec>: <text>

While loading a command file, an unknown command was encountered.

Illegal value of <text> specified for indicator <indname>.

Invalid characters were entered in the specified field of an INDICATORS display for the ALTER command.

Insufficient access rights to access the file.

The specified file could not be opened because the user lacks the access rights to that file.

Insufficient memory for search pattern.

There is not enough memory to compile the specified search string.

Internal error attempting to modify variable.

An internal error was detected.

Invalid address specified <n>.

An illegal address was specified for the MEMORY command.

Invalid characters in variable name.

Invalid characters are present in the data name operand for the VARIABLE command or the MODTRAP command.

Invalid decimal characters used.

The specified value contains illegal decimal digits. Valid values are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Invalid decimal data specified; possibly two decimal points.

The character string specified contains all legal decimal data characters, but in an incorrect format.

Invalid hex characters used in address specification.

The specified value for an address value contains an invalid character.

Invalid object format.

The object format bits of the program file contain an unknown value. Current legal values are '000'b and '001'b.

Invalid program address specified.

Through either the ALTER command or the CONTINUE address command, an attempt was made to set the program control word (PCW) to an invalid value.

Invalid Run block length.

The Run block length field of the program file contains an invalid value.

Invalid Symbolic block length.

The Symbolic block length field of the program file contains an invalid value.

Left parenthesis expected.

The data name can only be followed by a subscript reference, starting with a left parenthesis. A different character was found.

Library name is invalid.

Either illegal characters were specified or there are more than eight characters specified for the library name.

Library not found on volume.

The specified library was not found on the specified volume.

Memory location is non-modifiable.

A memory display which is not part of the user's data segment was the target of the ALTER command.

No diagnostic present for this program.

There is no diagnostic present for this program. As a result, no DIAGNOSTIC display was created.

No Linkage block is available to display procedure names.

The user program contains no LINKAGE information. As a result, no procedure information is available.

No PF key specified.

The ASSIGN command with no operands was specified and the ENTER key was pressed.

No <text> indicators found.

No indicators of the specified class were found.

Only <n> attributes will be accepted.

A limit of <n> attributes can be displayed, but the user has specified more than <n>. Rather than ignore the command completely, the first <n> attributes have been accepted and the rest ignored.

Only consecutive and print files are acceptable.

The only type of files that are legal for the LISTFILE command are consecutive and print. Program, indexed, or WP files are not accepted.

Only ON or OFF are legal operands for the LINKLEVELS command.

An illegal operand was specified for LINKLEVELS.

Please specify a variable name.

The VARIABLE command was specified without operands.

Recursive macro expansion detected.

A macro definition has been expanded in an illegal circular pattern that creates an infinite loop. For example, the following macro definitions would create an infinite loop:

```
  DEFINE A B
  DEFINE B C
  DEFINE C A
```

Right parenthesis expected.

The data name can only be followed by a subscript reference ending with a right parenthesis. A different character or end-of-text was found.

Section specified is not a static section.

The section name for a MODTRAP OFFSET command must be a static section, since code sections are not modifiable.

Separator in illegal position.

Either the file specification started with a delimiter, or two delimiters were found in a row.

Syntax error in search pattern.

The specified search string contains an invalid sequence of characters.

<text> is not a known attribute.

An illegal operand was specified for the ATTRIBUTES command.

<text> is not a valid FRAME operand.

An illegal operand was specified for the FRAME command.

<text> is not a valid macro name.

The first operand of the DEFINE command is not a valid macro name. Macro names must start with a letter and can be followed by one or more letters, digits, or the underscore.

<text> is not a valid operand for the STEP command.

An illegal operand was specified for the STEP command.

<text> is not a valid operand for the STACKTRACE command.

An illegal operand was specified for STACKTRACE.

<text> is not a valid PF key.

The first operand specified for the ASSIGN command is not an integer value in the range 1-256.

<text> is not defined.

An unknown command or macro name was entered.

The ACTIVATE command requires that at least one trap be set.

ACTIVATE was executed when no traps were set.

The ALTER command is only valid in the data window.

ALTER can be executed only with the cursor positioned in the data window.

The CODESECTIONS command does not accept any operands.

The specified operand was not accepted because the command does not function with operands.

The COUNT command accepts only one operand.

COUNT was specified with too many operands.

The COUNT command requires a count value operand.

COUNT was specified without operands.

The COUNT command requires that at least one trap be set.

COUNT was executed when no traps were set.

The COUNT operand is not a valid unsigned integer value.

This message is self-explanatory.

The COUNT operand must be greater than zero.

COUNT was specified with an operand less than zero.

The cursor must be positioned if an entry is not marked.

The ALTER command requires that either an entry be marked, or the cursor be positioned.

The Debugger window structure cannot be altered while in simplified mode.

You may not issue the following commands while you are using Easy mode: DATA, LISTING, TRAPS, and WINDOW.

The DEACTIVATE command requires that at least one trap be set.

DEACTIVATE was executed when no traps were set.

The DEBUGFILE command only takes 1 operand.

DEBUGFILE was specified with too many operands.

The DEFINE command may not be used with a PF key.

DEFINE was executed and a PF key was pressed.

The exponent given is too large for the target value.

This message is self-explanatory.

The exponent given is too small for the target value.

This message is self-explanatory.

The length has been reduced to the limit of 256.

A length which is greater than 256 bytes was specified for the MODTRAP ADDRESS command. Rather than ignore the command completely, the length has been truncated to 256.

The LISTFILE command only takes 1 operand.

LISTFILE was specified with too many operands.

The LISTING command does not accept any operands.

The specified operand was not accepted because the command does not function with operands.

The LOAD command requires a file name.

No file name operand was specified for LOAD.

The maximum macro expansion depth of 16 has been exceeded.

The maximum number of nested macro levels (macros that invoke macros) is 16. That limit was exceeded.

The MEMORY command requires an address operand.

MEMORY was specified without an operand.

The OPCODE command requires at least one operand.

OPCODE was specified without operands.

The section specified was not found.

The section name specified for the MEMORY OFFSET command was not found.

The trap limit of <n> has been exceeded.

The limit of <n> traps has been exceeded. In order to create more traps, one or more existing traps must be deleted from the list.

The value of the displacement field is too large.

The specified value for a displacement value is greater than the limit of 4095.

The volume table of contents (VTOC) is full.

The specified output file could not be opened because there is insufficient space left in the VTOC of the volume specified.

There is no line number information for this line.

This statement number does not appear in the statement number subblock for this code section.

There is no status display for <text>.

An illegal operand was specified for the STATUS command.

There were too few subscripts specified.

The specified array reference does not contain a sufficient number of subscripts needed for the specified data name.

There were too many subscripts specified.

The specified array reference contains more subscripts than needed for the specified data name.

This entry is not modifiable.

The ALTER command was executed and the target entry was a non-modifiable type of entry (CODESECTIONS, DIAGNOSTIC, PROCEDURES, STACKTRACE, STATICSECTIONS).

This variable is not an array but subscripts were given.

Subscripts were specified for a scalar data name.

This variable name is not present in the current section.

The specified data name was not found in the symbolic section for the currently selected code section.

Too many characters in comparand field.

The limit of eight characters for a comparand field for the MODTRAP REGISTER command has been exceeded.

Too many digits specified.

More digits were specified for this numeric data item than it can hold.

Too many operands specified for the STEP command.

There can be a maximum of two operands for the STEP command, INSTRUCTION, and a count value n, but more were found.

Too many pattern groupings.

There can be a maximum of 32 pattern groupings and more were found.

Trap limit reached. Delete or deactivate other traps.

The limit for the number of microcode traps has been reached. Creation of this trap is not possible without deleting or deactivating an already existing trap entry.

Unable to find Linkage information for this section.

There is no linkage information available for this code section.

Unable to find Symbolic information for this section.

This code section contains no symbolic information.

Unable to open file on an unlabeled volume.

The volume specified contains no VTOC. As a result, the specified file was not found.

Unable to reset program address.

An invalid restart address is specified for the CONTINUE command.

Unknown symbolic subblock found.

An invalid symbolic subblock has been detected in the current symbolic section.

Value given could not be converted to float binary.

Either invalid characters were detected or the specified value cannot be converted to floating-point binary.

Value given could not be converted to float decimal.

Either invalid characters were detected or the specified value cannot be converted to floating-point decimal.

Value given could not be converted to integer.

Either invalid characters were detected or the specified value cannot be converted to an integer.

Value is unsigned, but a sign character was specified.

This data name is specified to be an unsigned value but a sign character was entered.

VARIABLE, ADDRESS, OFFSET, or REGISTER must be specified.

An unknown keyword was found as the first operand of the MODTRAP command. Only VARIABLE, ADDRESS, OFFSET, and REGISTER are accepted.

Volume full.

The specified output file could not be opened because there is insufficient space left on the volume specified.

Volume is not mounted.

The specified volume is not currently mounted.

Volume name is invalid.

Either illegal characters were specified or there are more than six characters specified for the volume name.

APPENDIX D
DEBUGGER COMMAND ABBREVIATIONS

The following list contains the shortest unique abbreviation for each corresponding screen management or debugging command.

Command	**Abbreviation**
ACTIVATE	AC
ALTER	AL
ASSIGN	AS
ATTRIBUTES	AT
BREAK	B
CANCEL	CAN
CASE	CAS
CLEAR	CLE
CLOSE	CLO
CODESECTIONS	COD
COLUMN	COL
CONTINUE	CON
COUNT	COU
CURSOR	CU
DATA	DA
DEACTIVATE	DEA
DEBUGFILE	DEB
DEFINE	DEF
DELETE	DEL
DIAGNOSTIC	DIA
DISPLAY	DIS
DUMP	DU
EASY	E
FIND	FIN
FIRST	FIR
FLOATREGISTERS	FL
FRAME	FRA
FREEZE	FRE
FULL	FU
GOTO	G

Command	**Abbreviation**
HELP	HEL
HEX	HEX
HISTORY	HI
INDICATORS	IND
INSIDE	INS
LAST	LA
LEFT	LE
LINKLEVELS	LIN
LISTFILE	LISTF
LISTING	LISTI
LOAD	LOA
LOCATE	LOC
MARK	MAR
MATCH	MAT
MEMORY	MEM
MENU	MEN
MODTRAP	MO
NEXT	NE
NOTE	NO
OPCODE	OP
OUTSIDE	OU
POSITION	PO
PREVIOUS	PRE
PRINT	PRI
PROCEDURES	PROC
PROGRAMS	PROG
RECALL	REC
REGISTERS	REG
RIGHT	RI
ROW	RO
SCREEN	SC
SEARCH	SEA
SECTION	SEC
SET	SET
SNAPSHOT	SN
STACKTRACE	STAC
STATE	STATE
STATICSECTIONS	STATI
STATUS	STATU
STEP	STE
TRAINING	TRAI
TRAPS	TRAP
VARIABLE	VA
VERSION	VE
WINDOW	W

INDEX

WANG

Customer Comment Form

Publication Number 715-1144

Title VS SYMBOLIC DEBUGGER REFERENCE RELEASE 7 SERIES

Help Us Help You . . .

We've worked hard to make this document useful, readable, and technically accurate. Did we succeed? Only you can tell us! Your comments and suggestions will help us improve our technical communications. Please take a few minutes to let us know how you feel.

How did you receive this publication?

- ☐ Support or Sales Rep
- ☐ Wang Supplies Division
- ☐ From another user
- ☐ Enclosed with equipment
- ☐ Don't know
- ☐ Other ____________

How did you use this Publication?

- ☐ Introduction to the subject
- ☐ Classroom text (student)
- ☐ Classroom text (teacher)
- ☐ Self-study text
- ☐ Aid to advanced knowledge
- ☐ Guide to operating instructions
- ☐ As a reference manual
- ☐ Other ____________

Please rate the quality of this publication in each of the following areas.

	EXCELLENT	GOOD	FAIR	POOR	VERY POOR
Technical Accuracy — Does the system work the way the manual says it does?	☐	☐	☐	☐	☐
Readability — Is the manual easy to read and understand?	☐	☐	☐	☐	☐
Clarity — Are the instructions easy to follow?	☐	☐	☐	☐	☐
Examples — Were they helpful, realistic? Were there enough of them?	☐	☐	☐	☐	☐
Organization — Was it logical? Was it easy to find what you needed to know?	☐	☐	☐	☐	☐
Illustrations — Were they clear and useful?	☐	☐	☐	☐	☐
Physical Attractiveness — What did you think of the printing, binding, etc?	☐	☐	☐	☐	☐

Were there any terms or concepts that were not defined properly? ☐ Y ☐ N If so, what were they? ____________

After reading this document do you feel that you will be able to operate the equipment/software? ☐ Yes ☐ No ☐ Yes, with practice

What errors or faults did you find in the manual? (Please include page numbers) ____________

Do you have any other comments or suggestions? ____________

Name ____________ **Street** ____________

Title ____________ **City** ____________

Dept/Mail Stop ____________ **State/Country** ____________

Company ____________ **Zip Code** ________ **Telephone** ____________

Thank you for your help.

WANG

Cut along dotted line

Fold

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

WANG

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 16 LOWELL, MA

POSTAGE WILL BE PAID BY ADDRESSEE

Wang Laboratories, Inc.
Technical Publications Dept.
M/S 012-260
One Industrial Avenue
Lowell, Massachusetts 01851-9971

Fold

WANG

To Order by Phone, Call:
1-(800)225-0234
Telex 172108

Order Form for Wang Manuals and Documentation

① Customer Number (If Known)

② Bill To: ______ Ship To: ______

③ Customer Contact: ( ) ( ) ______ ④ Date ______ Purchase Order Number ______
Phone Name

⑤ Taxable Yes ☐ No ☐ ⑥Tax Exempt Number ⑦Credit This Order to A Wang Salesperson Please Complete ______ Salesperson's Name ______ Employee No. ______ RDB No.

⑧ Document Number	Description	Quantity	⑨ Unit Price	Total Price

⑩ ______ Authorized Signature ______ Date

☐ Check this box if you would like a free copy of
WangDirect Software & Literature Catalog
(711-0888A)

Sub Total	
Less Any Applicable Discount	
Sub Total	
LocalStateTax	
Total Amount	

Ordering Instructions

1. If you have purchased supplies from Wang before, and know your Customer Number, please write it here.
2. Provide appropriate Billing Address and Shipping Address.
3. Please provide a phone number and name, should it be necessary for WANG to contact you about your order.
4. Your purchase order number and date.
5. Show whether order is taxable or not.
6. If tax exempt, please provide your exemption number.
7. If you wish credit for this order to be given to a WANG salesperson, please complete.
8. Show part numbers, description and quantity for each product ordered.
9. *Pricing extensions and totaling can be completed at your option; Wang will refigure these prices and add freight on your invoice.*
10. Signature of authorized buyer and date.

Wang Terms and Conditions

1. **TAXES** — Prices are exclusive of all sales, use, and like taxes.
2. **DELIVERY** — Delivery will be F.O.B. Wang's plant. Customer will be billed for freight charges; and unless customer specifies otherwise, all shipments will go best way surface as determined by Wang. Wang shall not assume any liability in connection with the shipment nor shall the carrier be construed to be an agent of Wang. If the customer requests that Wang arrange for insurance the customer will be billed for the insurance charges.
3. **PAYMENT** — Terms are net 30 days from date of invoice. Unless otherwise stated by customer, partial shipments will generate partial invoices.
4. **PRICES** — The prices shown are subject to change without notice. Individual document prices may be found in the WangDirect Software & Literature Catalog (711-0888A)
5. **LIMITATION OF LIABILITY** — In no event shall Wang be liable for loss of data or for special, incidental or consequential damages in connection with or arising out of the use of or information contained in any manuals or documentation furnished hereunder.

WANG

Fold

WANG

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 16 LOWELL, MA

POSTAGE WILL BE PAID BY ADDRESSEE

WangDirect
Wang Laboratories, Inc.
M/S 017-110
800 Chelmsford Street
Lowell, Massachusetts 01851-9972

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

Fold

Cut along dotted line

WANG

ONE INDUSTRIAL AVENUE, LOWELL, MA 01851
TEL. (508) 459-5000, TELEX 172108

Printed in U.S.A. 715-1144 10-88