WANG

VS

Program Development Tools Reference

VS
Program Development Tools Reference

3rd Edition — August 1983

800-1307-03

WANG LABORATORIES, INC., ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 • TEL: 617/459-5000, TWX 710-343-6769, TELEX 94-7421

Disclaimer of Warranties and Limitation of Liabilities

The staff of Wang Laboratories, Inc., has taken due care in preparing this manual; however, nothing contained herein modifies or alters in any way the standard terms and conditions of the Wang purchase, lease, or license agreement by which this software package was acquired, nor increases in any way Wang's liability to the customer. In no event shall Wang Laboratories, Inc., or its subsidiaries be liable for incidental or consequential damages in connection with or arising from the use of the software package, the accompanying manual, or any related materials.

NOTICE:

All Wang Program Products are licensed to customers in accordance with the terms and conditions of the Wang Laboratories, Inc. Standard Program Products License; no ownership of Wang Software is transferred and any use beyond the terms of the aforesaid License, without the written authorization of Wang Laboratories, Inc., is prohibited.

This manual replaces and obsoletes the second edition of the *VS Program Development Tools* (800-1307PT-02). For a list of changes made to this manual since the previous edition, see the "Summary of Changes".

WANG

WANG LABORATORIES, INC., ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 • TEL: 617/459-5000, TWX 710-343-6769, TELEX 94-7421

PREFACE

This manual introduces the programmer to the VS program development tools. It is written for programmers familiar with basic data processing and programming concepts.

After reading this manual and the VS Programmer's Introduction, a user can work with the VS menus and Command Processor, and create, compile (or assemble), and run a program on the VS.

Chapter 1 provides an overview of the program development process and a brief introduction to the EDITOR, LINKER, and Symbolic Debugger.

Chapter 2 discusses the EDITOR. This chapter explains program development through the EDITOR and describes all EDITOR functions.

Chapter 3 describes the LINKER. This chapter discusses the primary functions of this program and provides detailed information about all LINKER screens. A description of the Link and External Reference maps and a table of LINKER return codes are also included.

Chapter 4 includes a brief discussion of program compilation and describes the sections of the program file that apply to program linking and debugging.

Chapter 5 discusses the VS Symbolic Debugger. After a general introduction to debugging, this chapter describes how to use the Symbolic Debugger and provides information about each Debugger option.

Appendix A provides a sample EDITOR session. Appendix B provides an example for each LINKER function. Appendix C describes the use of the Symbolic Debugger through a programming example. Appendix D provides a sample Link map; Appendix E provides a sample External Reference map. Appendix F lists compiler, Assembler, LINKER, and EDITOR GETPARM information. A glossary appears after Appendix F.

For a detailed discussion of VS system features and programming languages, consult the following manuals:

VS Assembler Language Reference Manual	800-1200AS
VS BASIC Language Reference Manual	800-1202BA
VS COBOL Language Reference Manual	800-1201CB
VS FORTRAN Reference	800-1208FR
VS Operating System Services	800-11070S
VS PL/I Language Reference Manual	800-1209PL
VS Principles of Operation	800-1100PO
VS Procedure Language Reference	800-1205PR
VS Programmer's Introduction	800-1101PI
VS RPG II Language Reference Manual	800-1203RP

SUMMARY OF CHANGES
FOR THE VS PROGRAM DEVELOPMENT TOOLS REFERENCE (800-1307PT-03)

Type	Affected Features	Affected Pages
Technical Changes	EDITOR	Chapter 2
	Line numbers for Procedure Language	2-2
	Language parameters	2-4
	DLIBRARY and DVOLUME	2-4
	PF8 (FIND)	2-7
	PF9 (MOD)	2-8
	PF10 (CHNG)	2-8
	PF13 (MOVE)	2-10
	PF14 (COPY)	2-10
	Special Menu: Modification Code for Procedure Language	2-12
	PF10 (COMPILE)	2-15
	PF11 (ERRORS)	2-15
	PF14 (LINK MAP)	2-16
	Program Linking	2-16
	Linker Options	2-17, 3-5, F-4, F-7, F-9, F-13
Editorial Changes	Person changed throughout text	
	Miscellaneous editorial changes	2-5, 2-6, 2-10 2-16, 3-7, 5-9

CONTENTS

FIGURES

CHAPTER 1
INTRODUCTION

The Wang VS offers an easy-to-use integrated programming environment. Wang provides a set of system programs to make programming on the VS easier. This manual provides a detailed discussion of those programs: EDITOR, LINKER, and Symbolic Debugger.

The program development cycle involves six steps:

1. Entering and editing source text
2. Using a language compiler or assembler to translate a source program into a program file,
3. Linking program files, if necessary
4. Running the linked program file
5. Testing
6. Debugging

The EDITOR maintains an integrated programming environment in which you can interactively create, edit, compile or assemble, link, and run programs. Thus, using the EDITOR frees you from invoking each of the system programs that accomplish these functions.

You can use the EDITOR to enter and modify source code in any of the available programming languages. Once you have entered the source code, program development functions are available to display errors, modify incorrect lines, and rerun the program. You can repeat each process as often as necessary within the EDITOR.

The LINKER enables you to combine any number of separately compiled programs to form a single executable program file.

On the Wang VS, you can perform linking through the EDITOR or through the LINKER. By running the LINKER, you can combine any number of program files into one executable program. Thus, the LINKER makes it possible to construct a large and complex program by simply linking a number of smaller modules. Also, if you modify one of the linked modules, you can replace that module without repeating the entire linking process.

The Symbolic Debugger helps you interactively locate errors in an executing program. You can use the Symbolic Debugger to examine a running program during each step of execution, thereby simplifying the debugging process.

In addition, the Symbolic Debugger enables you to gather information about the operation of a program and to make changes to data during program execution. It is useful because data names and program control structures can be referenced with the same names and formats used in the program. For example, to find the value of a particular variable, you reference the variable by using the name the programmer assigned; you do not have to search memory by using data addresses.

CHAPTER 2
EDITOR

2.1 INTRODUCTION

The EDITOR is an integral part of the program development process. You can perform the following program development functions from within the EDITOR:

- Enter a source program or procedure
- Display and correct errors in a source program
- Compile, link, and run a program; or run a procedure
- Debug a program
- Run System Utilities or other programs

You can perform the following editing functions within the EDITOR:

- Delete, insert, and replace text within the file
- Search for text within the file
- Insert lines into the file
- Delet lines from the file
- Move lines from one location to another within the file
- Copy lines from one location to another within the file
- Copy lines into the file from another file

2.2 EDITOR CONCEPTS

This section explains the work file, modification codes, line numbers, EDITOR files and distinguishes between mode and function.

2.2.1 Work File

Whenever you invoke the EDITOR, it creates a temporary work file into which the edited text is copied. When you edit an existing file, the EDITOR copies the contents of that file into a work file. All actions modify the edited text, leaving the permanent file unchanged. You can save the edited text as a new, permanent file, or you can replace an existing, permanent file.

Since the work file is automatically deleted when the EDITOR terminates processing, you must save the file before you exit the EDITOR. If you attempt to exit without saving the file, a message displays to warn you. You then have the option to save it before exiting.

2.2.2 Modification Codes

The modification code is an optional, user defined and supplied, sequence of characters which the EDITOR adds to each source line that you insert or change during an EDITOR session. These codes allow you to keep a record of changes made to your programs. Modification codes do not affect program compilation or assembly. You can view the modification code through the display functions of the EDITOR.

You can define a modification code during any EDITOR session. The EDITOR includes that modification code in any source lines subsequently inserted or changed. You can use different codes to identify programming changes made at different times. For example, the initial modification code could be VERSION1, and codes for lines that you added or changed in later versions could be VERSION2, VERSION3, etc. This clearly shows which lines were changed at each stage of program development.

Modification codes apply only to lines that are added or changed during the current EDITOR session. The modification codes for lines that are deleted from the file are lost. If no modification code is in use during the current editing session, modification codes already in the file are left unchanged.

The special menu Set function (PF2), described in Section 2.5, enables you to define a modification code. Section 2.4 explains how to examine the modification codes.

2.2.3 EDITOR Line Numbers

When the EDITOR generates a source listing, line numbers occur in the following columns on each source line:

Language	Columns
Assembler	75 through 80
BASIC	1 through 6
COBOL	1 through 6
FORTRAN	75 through 80
PL/I	75 through 80
PROCEDURE	73 through 80
RPG II	1 through 5

2.2.4 Files Created by the EDITOR

The EDITOR creates and uses a number of files during program development. When creating a file, the EDITOR assigns the name ##TEST to the edited text. This name remains until you save or compile the file. When you save the file, you assign a name to it. If you compile the file without assigning a name, the EDITOR assigns a temporary file name. Once the edited file compiles successfully, the object file name is the same as the name assigned to the source file.

2.2.5 Definitions: Mode and Function

There is a subtle difference between the terms "mode" and "function" within this chapter. There are nine modes within the EDITOR program: Display, Find, Modify, Change, Insert, Delete, Move, Copy, and Error. Within each mode, you can perform various functions. For example, within Display mode, you can move the display back 15 lines. Within each editing mode, the name of the current mode, and a list of the available functions within that mode always appears at the top of the screen. The system always returns to Display mode when you exit from any other mode.

2.3 RUNNING THE EDITOR

To run the EDITOR, press PF1 (RUN) from the Command Processor. The system accepts the program, library, and volume names of the program to be run. Enter EDITOR as the PROGRAM name, and press ENTER. It is not necessary to enter a library or volume name because the EDITOR resides in the System Library on the System Volume.

The EDITOR first displays the Input screen. This screen accepts the programming language and the name, library, and volume of the file which you will edit. Definitions for each Input screen parameter follow:

Parameter	Description
LANGUAGE	The programming language you selected. You can use any unique abbreviation. The EDITOR automatically sets default tab settings and line numbers for the selected language.
FILE	The name of the file you want to edit. When creating a new file, leave this field blank. When editing an existing file, enter the name of that file.
LIBRARY and VOLUME	The library and volume names for the source file. The EDITOR uses this library and volume location when creating and saving the source file during this editing session. When you edit an existing source file, specify its library and volume names. By default, LIBRARY and VOLUME correspond to INLIB and INVOL, respectively, as defined by the SET command (PF2) of the Command Processor.

PLIBRARY and PVOLUME — The library and volume names for object files created during this session. The default values for PLIBRARY and PVOLUME correspond to OUTLIB and OUTVOL, respectively, as defined by the SET command (PF2) of the Command Processor.

LLIBRARY and LVOLUME — The library and volume names for print files generated by a compiler, the assembler, and/or the LINKER. This program listing contains the source file, error messages, and compiler or assembler options, and LINKER options. Default values for LLIBRARY and LVOLUME correspond to SPOOLIB and SPOOLVOL, respectively, as defined by the SET command (PF2) of the Command Processor.

DLIBRARY and DVOLUME — The library and volume names for assembler symbolic debug information files. The EDITOR determines the default value for DLIBRARY by combining the user ID with DEBUG, i.e., KDKDEBUG. The default for DVOLUME corresponds to OUTVOL, as defined by the SET command (PF2) of the Command Processor. These names are ignored for languages other than assembler language.

SCRATCH — This parameter determines whether to delete or save a file having the same name and location as a second file created by the EDITOR during compilation or assembly. If SCRATCH=YES, any object, listing, or debug files replace any existing files of the same name. If SCRATCH=NO, the EDITOR always requests permission to delete the existing file.

You can examine and change values of Input screen parameters after the values are set. Refer to Section 2.5 for more details.

Once you have supplied all parameters, press ENTER to access the EDITOR main menu.

2.4 EDITOR MAIN MENU

After you specify parameters for the Input screen, the EDITOR displays the main menu, which offers 16 functions, associated with PF keys 1 through 16. These options fall into three functional groups:

- Display functions (PF1 through PF8) enable you to examine the edited text

- Edit functions (PF9 through PF14) enable you to enter and alter the contents of the edited text

- Special functions (PF15 and PF16) indicate the position of the cursor and provide access to the EDITOR special menu

2.4.1 DISPLAY Functions

Display functions permit you to examine portions of the edited text. You can select a display function by pressing any one of PF keys 1 through 8. The available main menu functions appear at the top of the screen when the edited text is being displayed. Display functions include the following:

PF Key	Function	Description
PF1	DISP	Displays source text on the screen
PF2	FIRST	Displays the first 20 lines of the file
PF3	LAST	Displays the last five lines of the file
PF4	PREV	Displays the previous 15 lines of the file
PF5	NEXT	Displays the next 15 lines of the file
PF6	DOWN	Moves the display back one line
PF7	UP	Moves the display forward one line
PF8	FIND	Locates specified text within the file

2.4.2 EDIT Functions

Edit functions allow you to make changes to the contents of the file. Within a particular edit function, all PF keys are disabled except PF1 (Display), PF2 (Set), PF10 (Change RPG II form type), and PF15 (Show Column).

If you make a mistake within an edit function, press PF1 to return to Display mode without making the indicated changes.

To simplify changing the tabs and search settings, the SET function (PF2 on the special menu) is also available in FIND, MODIFY, CHANGE and INSERT modes (refer to section 2.5).

Within edit functions that accept RANGE parameters, the START of the range always defaults to the line number of the line where the cursor is currently located.

Edit functions include the following:

PF Key	Function	Description
PF9	MOD	Modifies contents of the current screen
PF10	CHNG	Replaces text within the file
PF11	INS	Inserts lines within the file
PF12	DEL	Deletes lines from the file
PF13	MOVE	Moves lines within the file
PF14	COPY	Copies lines within the file

If a modification code is in effect for the editing session, it is stored in the modification code field of all lines that are changed, inserted, or copied by the External Copy (PF8 of the special menu), Modify (PF9), Change (PF10), Insert (PF11), and Copy (PF14) functions. Refer to section 2.2.2.

2.4.3 Range Specification

Several EDITOR functions accept a range of lines within which the function is to operate. The range parameters are START of range, END of range, and TARGET line. The START and END specifications are inclusive. You can specify the range parameters in one of the following ways:

- A line number of one to six digits. Leading zeros are not required. Strings of digits can be differentiated from line numbers by enclosing them in quotes.

- A text string, optionally enclosed within matching single or double quotes, may be used for any or all of the START, END, or TARGET values.

- Depending on the value of the CASE field on the DEFAULTS screen, all text string searches are either case-sensitive (CASE = EXACT), or case-insensitive (CASE = ANY). This means that text strings may be compared for equality regardless of their case (upper or lower). Enclosing text within quotes forces case-sensitivity for that particular operation.

- The special word FIRST (without quotes). This is equivalent to the first line number in the edited text.
- The special word LAST (without quotes). This is equivalent to the last line number in the edited text.

- The special word ALL (without quotes), specified in the START parameter. This indicates the entire edited text.

If you do not specify a value for END, the range is the single line indicated by START (unless the special word ALL is used).

2.4.3 EDITOR Main Menu Functions

The following are the EDITOR main menu functions:

PF1 (DISP) Returns to Display mode. This function displays the current screen of the edited text.

PF2 (FIRST) Displays the first screen of the edited text.

PF3 (LAST) Displays the last five lines of the edited text.

PF4 (PREV) Moves the display back 15 lines. This function displays the previous 15 lines of the edited text, followed by the first five lines of the current screen.

PF5 (NEXT) Advances the display 15 lines. This function displays the next 15 lines of the edited text, preceded by the last five lines of the current screen.

PF6 (DOWN) Moves the display back one line.

PF7 (UP) Moves the display forward one line.

PF8 (FIND) Finds specified text within the edited text. You can supply either a line number or a text string. Find searches both forward and backward from the cursor's current position. You can enter text with or without quotes, but when searching for a character string that is all digits, it must be enclosed in matching quotes.

Press PF8 to enter Find mode. Type the line number or text string to be located into the highlighted entry line and press ENTER. The EDITOR locates the next occurrence and positions the cursor at the text string found during a text Find, and in column one after a line number Find. Press ENTER repeatedly to find subsequent matching lines.

To find a previous specification, press PF8 after entering the line number or text string. The EDITOR locates the previous occurrence and positions the cursor at the text string found during a text Find, and in column one after a line number Find. Press PF8 repeatedly to find each preceding occurrence of your text.

Find will accept FIRST and LAST to locate and display the first and last lines of the edited text. Upon completion of the command, the EDITOR returns to Display mode.

You can interrupt a Find function at any time by pressing PF1. The search is interrupted, and you remain in Find mode. Press PF1 again to exit Find and return to Display mode.

PF9 (MOD) Modifies displayed lines or adds lines to the end of the edited text. In the latter case, the word "ADD" is displayed on the left side of the screen.

Press PF9 to enter Modify mode. The EDITOR highlights the current screen of text. Move the cursor to the characters you want to modify. You can modify in the following ways:

Character	Replace the character at the cursor with your keyed character.
DELETE key	Deletes the character at the cursor.
INSERT key	Inserts a blank before the character at the cursor.
ERASE key	Deletes the rest of the line, including the character at the cursor.

When you complete the modifications, press ENTER to save the changes in the edited text and to return to Display mode. Press PF1 to return to Display mode without making the changes.

If you press ENTER after adding new lines to the end of the text on the screen, the new lines are added to the edited text, the text scrolls up, and you can continue adding lines to the file. Modify mode cannot be used to insert lines between existing lines of the edited text. Use the Insert function (PF11) to perform this operation.

RPG II programmers can change the form type by pressing PF10. The new form type remains in effect for all lines that are subsequently added.

PF10 (CHNG) Changes all occurrences of a text string within a specified range.

Press PF10 to enter Change mode. The Change parameters are as follows:

OLD	The text to be replaced.
NEW	The new text. You can enter a null string as a completely blank line or as two consecutive, matching quotes.
START and END	The range of lines to be changed.

Press ENTER to perform the Change. If the specified RANGE contains exactly one line, all occurrences of the OLD text are changed to the NEW text in the specified line, and the EDITOR returns to Display mode. If the specified RANGE contains more than one line, the cursor is positioned at the first occurrence of the OLD text string. Press ENTER to change that text and find the next occurrence, PF8 to skip that particular change and find the next occurrence, PF10 to change all remaining occurrences without further intervention, or PF1 to terminate the Change command.

If the change causes the line to exceed the maximum line length, the EDITOR displays a warning message. Press ENTER to proceed with the change, PF8 to skip that particular change, or PF1 to terminate the Change command.

You can interrupt the Change function at any point by pressing PF1. If changes have already occurred, the cursor is positioned at the last line changed. You can also press PF1 to exit from the Change mode without making any changes.

PF11 (INS) Inserts one or more new lines at the position of the cursor. Press PF11 to enter Insert mode. The EDITOR displays column numbers at the top of the screen, and a new line number is displayed at the cursor position. Type the new line and press ENTER to add the line to the edited text.

To insert lines before the first line on the screen, position the cursor above the first line and press PF11.

You can change the line number by backspacing and retyping the number; subsequent line numbers are then computed from the new line number.

RPG II programmers can change the default form type by pressing PF10 while in Insert mode. The new form type remains in effect for inserted lines until the form type is changed again.

PF12 (DEL) Deletes a specified range of lines. Press PF12 to enter Delete mode. Specify the START and END range of the lines you wish to delete, and press ENTER.

PF13 (MOVE) Moves a specified line or range of lines from one location in the file to another. Press PF13 to enter Move mode. Type in the START and END of the range of lines you wish to move. To move a single line, enter only the START parameter. The target line can be changed, if desired. Press ENTER to perform the Move. The moved lines will be inserted immediately following the target line.

The moved lines are automatically assigned line numbers in proportion to the line numbers of the target line and the following line. If there are not enough line numbers between these lines, the EDITOR displays a warning to renumber your program before the specified lines can be moved. Refer to section 2.5 for details about renumbering.

To move lines above the first line on the screen, position the cursor above the first line before pressing PF13, or specify a target line of 0 (zero).

To move lines to the end of the file, specify LAST as the target line number.

PF14 (COPY) Copies a specified line or range of lines from one location within the edited text to another. Press PF14 to enter Copy mode. Type in the START and END of the range of lines you wish to copy. To copy a single line, enter only the START parameter. The target line can be changed, if desired. Press ENTER to perform the Copy. The copied lines will be inserted immediately following the target line, but are not deleted from their original position.

To copy lines above the first line of the file, position the cursor above the first line before pressing PF14, or specify a target line of 0 (zero).

To copy lines to the end of the file, specify LAST as the target line number.

PF15 (COL) Shows the cursor column position in the upper right corner of the screen. This function is available from within Display, Modify and Insert mode.

PF16 (MENU) Displays the special menu. Refer to Section 2.5 for a description of all special menu functions. This function is available from Display mode only.

2.5 EDITOR SPECIAL MENU

The EDITOR special menu provides file management and compilation functions. To select the special menu, press PF16 from the EDITOR main menu.

PF1 (DISPLAY)

Resumes displaying the edited text in Display mode.

PF2 (SET)

Sets the tabs, upper/lower case and search modes, and EDITOR/compiler options. The SET command is used to set the following options:

PF2 Sets the following Editor defaults:

TABS The EDITOR automatically sets tabs for the placement of source lines for each programming language. To set new tabs, enter the column number of each desired tab; to remove an existing tab setting, delete the column number of the tab.

MODE MODE enables you to restrict all input to uppercase (UPPER) or to accept uppercase and lowercase (UPLOW) mode. The default is UPPER.

CASE Allows case-insensitive searching for text (ANY), or requires exact case matching (EXACT).

COLUMN Specifies the search columns within which all searching operations are performed (FIND, and all textual RANGE specifications).
Entering MC in either COLUMN field sets the search columns and the Display to the modification code columns (refer to section 2.2.2). When displaying the modification codes, the Change command (PF10) is disabled.

FORMTYPE For RPG II programmers, FORMTYPE sets the default forms type and tab settings. The default initial forms type is F.

PF3 Sets automatic Scratch, Renumber, and Replace modes, and defines the modification code. When you press PF3, the EDITOR displays the Options screen. This screen accepts values for the following parameters:

SCRATCH Automatic Scratch mode causes the EDITOR to delete existing program and listing files that have the same name as those files created during the current operation. The acceptable values are YES and NO. The default is YES.

REPLACE Automatic Replace mode replaces the input file with the edited text before every compilation or assembly. For Procedure language files, it replaces the input file before every run. The acceptable values are YES and NO. The default is NO.

RENUMBER Automatic Renumber mode recomputes line numbers in the input file before the file is saved or replaced. The acceptable values are YES and NO. The default is NO.

NUMBER NUMBER is the starting line number for the RENUMBER option. This is the new line number for the first line in the resequenced file. The default starting line number is not the same for all programming languages.

INCR INCR is the increment between line numbers for the RENUMBER option. The default increment is not the same for all programming languages.

MODCODE MODCODE is the modification code associated with each line that is added or changed during the current editing session. Modification codes are described in detail in section 2.2.2. The default modification code is blank. The field locations and lengths are as follows:

Language	Starting Column	Ending Column	Length
Assembler	73	74	2
BASIC	73	80	8
COBOL	73	80	8
FORTRAN	73	74	2
PL/I	73	74	2
Procedure	72	74	3
RPG II			
Source	75	80	6
Table/Array	70	75	6

PF4 Sets compiler or assembler and LINKER options. The Options screen provides the current compiler or assembler and LINKER options. The parameters for the Options screen are described in detail in Appendix F.

PF5 Changes default file, library, and volume names for program, print, and other files that are created during the EDITOR session.

The Names screen provides current default file, library, and volume names (defined on the EDITOR Input screen) for the following files: compiled program (OBJECT), source listing (LISTING), error listing (ERRORS), linked program (LINKOBJ), and LINKER listing (LINKLIST).

Once you have set these parameters, they remain set until you change them, or terminate EDITOR processing.

PF3 (MENU)

Selects the EDITOR main menu, described in section 2.4.3.

PF4 (RESTART)

Allows you to terminate editing the current file and begin editing another file. If you select this option before you save (PF5) or replace (PF6) the edited file, the EDITOR displays a warning message and permits you to return to the special menu.

PF5 (CREATE)

Saves the edited text as a permanent file. The Output screen accepts values for the following parameters:

FILE — The file name for the edited text.

LIBRARY — The library in which the file resides. The default is INLIB, as defined by the SET command (PF2) of the Command Processor.

VOLUME — The volume on which the file resides. The default is INVOL, as defined by the SET command (PF2) of the Command Processor.

RETAIN — The retention period for the new file. Only a file's owner or a security administrator can delete or rename the file during this period.

FILECLAS — The file protection class associated with the file. Refer to the VS Programmer's Introduction for more information.

START — The first line of the edited text you want to copy into the new permanent file. The default (ALL) saves the entire file.

END — The last line of the edited text you want to copy into the new permanent file.

COMPRESS — Creates compressed records when the file is saved. Compression can reduce the number of bytes required for storage of repetitive characters. The acceptable values are YES and NO. The default value is YES.

NUMBER — Includes line numbers with the saved file. The acceptable values are YES and NO. The default value is YES.

PF6 (REPLACE)

Replaces the input file with the edited text. This function replaces the permanent copy of the edited text (the input file) with the current edited text. The file, library, and volume names are the same as those of the input file. You can perform this function only when you are editing an existing file.

When you press PF6, the EDITOR displays the Replace screen and accepts values for the following parameters:

COMPRESS Creates compressed records to save disk space. The default value is YES.

NUMBER Includes the line numbers in the saved file. The default is YES.

PF7 (RENUMBER)

Generates new line numbers for all or part of the edited text. The Renumber screen accepts values for the following parameters:

NUMBER The new starting line number for the file. The default starting line number depends on the programming language.

INCR The increment between line numbers. The default starting line number depends on the programming language.

START The first line to be renumbered. The default (ALL) renumbers the entire file.

END The last line to be renumbered. If the START parameter is ALL, leave this parameter blank.

PF8 (EXTERNAL COPY)

Copies a range of lines to the edited text from another file or from the file currently being edited. Move the cursor to the line immediately before the position where you want the copied lines to appear (the target line) and press PF8. The EDITOR displays the Xcopy screen, which accepts values for the following parameters:

FILE,
LIBRARY and
VOLUME The file, library, and volume names of the external file from which you want to copy the lines. If the file refers to the current file being edited, the RANGE specifications refer to the original contents of the input file. This is helpful for recovering lines that were accidentally deleted from the edited text.

START The first line or text string you want to copy into the edited text. The default (ALL) copies the entire external file into the edited text.

END — The last line or text string you want to copy to the edited text. If the START parameter is ALL, leave this parameter blank.

TARGET — The line in the edited text after which the lines are to be copied. The default target line is the current cursor position. To copy lines before the first line in the edited text, enter a target line of 0 (zero).

MODCODE — Indicates whether to retain the modification codes of the copied lines or to insert the current modification code into the copied lines. The default (NO) causes the EDITOR to use the current modification code.

PF9 (RUN)

Compiles/assembles and runs the program, or runs the procedure.

After the compilation, the Editor runs the compiled program if there were no errors. If there are compilation errors, the EDITOR displays a highlighted message at the top of the screen, and PF11 (ERRORS) is made available to examine the diagnostic messages.

PF10 (COMPILE or ASSEMBLE)

Compiles or assembles the program.

After the compilation or assembly has been completed, if there are compilation errors, the EDITOR displays a highlighted message at the top of the special menu screen and PF11 (ERRORS) is made available to examine the diagnostic messages. If there are no compilation errors, an object file is created, which you can run by pressing PF9.

PF11 (ERRORS)

Displays compilation or assembly diagnostic errors, if any. The EDITOR displays and highlights the incorrect line, followed by the diagnostic message for that line.

The functions available in the Errors display enable you to display the error listing, to locate erroneous lines in the edited text, or to return to the EDITOR special menu. The FIRST, LAST, PREVIOUS, NEXT, UP, DOWN, and FIND functions in the Error display apply to the error listing, not to the edited text.

To edit an erroneous text line, press PF11 from the special menu to display the errors. Move the cursor to a source line in the listing and press PF1. The EDITOR then displays the edited text, with the cursor positioned at the line in error. The line may then be corrected. This process may be repeated if there are more errors.

PF12 (LISTING)

Displays the source listing. This function appears after the edited text is compiled or assembled. The listing contains the source program statements, the diagnostic error messages, and other language-dependent information.

PF13 (UTILITIES)

Runs a VS utility or other program. When you press PF13, the EDITOR displays the Utility Processing menu. This option enables you to run any of the following utility programs by pressing the appropriate PF key: COPY, DISKINIT, DISPLAY, EZFORMAT, LISTVTOC, SORT, TAPECOPY, TAPEINIT, VERIFY, and WP. In addition, you can run any other program by specifying the program file, library, and volume names and pressing ENTER.

PF14 (Link Map)

Displays the Link Map following a compile and link operation. This operation is available following a Compile and Link operation. If there were any linkage errors, PF14 is highlighted.

PF15 (PRINT)

Prints the edited text. Press PF15 to display the Print Command screen. Enter the line number range to be printed. The default (ALL) causes the entire file to be printed. You can set the number of lines per page (default is 60) and can specify whether line numbers are to be included in the listing (the default is YES). You can interrupt this function at any time by pressing PF1.

PRINT generates a print file. Your default print mode determines whether the print file is immediately printed, stored, or placed in the print queue.

PF16 (EXIT)

Terminates EDITOR processing and returns to the Command Processor or controlling program. If the file has been modified since it was last saved, the EDITOR issues a warning message and allows you to return to the EDITOR special menu to create (PF5) or replace (PF6) the file. You can also exit without saving the edited text.

2.6 PROGRAM LINKING

Linking resolves references to programs that are external to the program being compiled or assembled. You can perform linking from within the EDITOR or by running the LINKER directly. Chapter 3 discusses running the LINKER directly. This section describes the process of compiling and linking a program within the EDITOR.

If the current object file refers to other object files, the EDITOR will use the Linker to resolve these references (External References). You can compile a main program and link subroutines, or you can compile a subroutine and link the main program with other subroutines.

The LINKER resolves external references in this order: first, from specific files named on the Linkfile screen, in the order named; and second, from files located in libraries named on the Llibrary screen, in the order in which the libraries are named. If two files have the same file name, unless a specific file is identified on the Linkfile screen, the LINKER uses the first file found in a library named on the Llibrary screen.

2.6.1 Link Screen

When you select PF9 (RUN) or PF10 (COMPILE) from the special menu, the EDITOR displays the Link screen after the compilation options. The Link screen accepts the following information:

Field	Description
LINK	Indicates whether subroutine linkage will occur. The default is NO.
FILES	Indicates whether specific files will be named. Use this field when you compile or assemble a subroutine and must link the main program. Also, use this field when the order in which files are normally linked is not the order you require. If the FILES parameter is YES, the EDITOR displays the Linkfile screen, described in section 2.6.2.
LIBRARY VOLUME	Identifies one library and volume location for a program module you want to link. The default library and volume are INLIB and INVOL, as set by the SET command (PF2) of the Command Processor. You can link program modules stored in up to eight separate libraries (refer to the MORE field).

MORE — Indicates whether or not program modules to be linked are stored in more than one library. The default is NO. If the MORE parameter is YES, the EDITOR displays the Llibrary screen, discussed in section 2.6.3.

MAP — Indicates whether or not a Link map will be printed. A Link map identifies all external symbols and indicates the length of each code section. If the MAP parameter is YES, the EDITOR prints a Link map. The Link map is described in section 3.6.1. A sample map appears in Appendix D.

XREF — Indicates whether a cross-reference listing will be printed. This listing cross-indexes all references to external symbols. The default (NO) does not generate a cross-reference listing. The cross-reference listing is described in section 3.6.2. A sample listing appears in Appendix E.

EXSEC — Allows you to include a listing of those sections that were excluded because of duplicate names. The default is NO. EXSEC lists the excluded sections and the names of the files in which they occur. The names are listed in the order found.

LINES — Indicates the number of lines per page for the Link map and the External Reference map. The default is 55.

SYMB — Indicates whether or not to retain symbolic debug information in an input file. If the SYMB parameter is YES, the EDITOR retains symbolic debug information in the linked output file.

2.6.2 Linkfile Screen

The Linkfile screen appears when you enter FILES=YES on the Link screen. The Linkfile screen enables you to provide names and locations of specific files to link or to define the sequence in which those files will be linked.

On this screen, each FILE, LIBRARY, and VOLUME set defines a specific file that will be linked to the current object file.

The ENTRY field defines the entry point for the linked program. The value for the field should be the name of the main program (the program that executes first).

2.6.3 Llibrary Screen

The EDITOR displays the Llibrary screen when you enter MORE=YES on the Link screen. The Llibrary screen enables you to link subroutines located in more than one subroutine library. You can define up to eight subroutine libraries. Each LIBRARY and VOLUME pair indicates the name and location of a library containing subroutine files that you wish to

link. The order in which you specify the libraries determines the order in which the LINKER searches for external references.

CHAPTER 3
LINKER

3.1 INTRODUCTION

Linking combines separately compiled or assembled object program modules into a single executable program. You must link programs that call subroutines since the subroutines are not a physical part of the program. Linking enables you to:

- Develop program modules independently, then link them together
- Maintain libraries of frequently used subprograms, then link individual subprograms with programs when necessary
- Access subprograms written by other programmers
- Use program modules written in different programming languages

On the VS, the LINKER performs the linking process by enabling you to do the following:

- Construct a single executable linked program, usually consisting of a main program and subprograms
- Replace a program module within a linked program. Therefore, you can modify a particular program module without relinking or recompiling the entire program.
- Remove symbolic debug information from a program, thereby reducing the size of the program.[However, removal of this information precludes the use of symbolic debugger features in debugging the program.]

NOTE

You can only link compiled or assembled program modules.

3.2 LINKER CONCEPTS

This section discusses the principal LINKER functions. Each discussion references LINKER screens, which section 3.4 explains.

3.2.1 Creating a Linked Program

A linked program consists of separately compiled program modules. To create a linked program, you must provide the names of the program modules you will link, and the starting point for execution of the linked program. You must also provide the name and location for the linked program.

First, specify the libraries and volumes for the program modules you want to link. You can name eight separate subroutine libraries. You identify the first library on the Options screen and any others on the Library screen.

Once you specify the subroutine libraries containing the program modules, specify the files you want to link by using the Input screens. Each Input screen requests the name and location of one program module. Specify the main program name and location. It is not necessary to specify more than one file if all linked modules reside in subroutine libraries which the Options and Library screens identify.

The LINKER attempts to resolve external references first from files named on Input screens. If there are unresolved external references, the LINKER searches named subroutine libraries in the order you specify.

Finally, the LINKER requests a name for the output file and the entry point name from the main program module. The result of this procedure is an executable linked program that consists of all linked and referenced program modules.

> **NOTE**
>
> For the linked program to load correctly, the size of the output program (in number of 1024-byte records) cannot exceed 512K records.

The following figure illustrates the creation of a linked program:

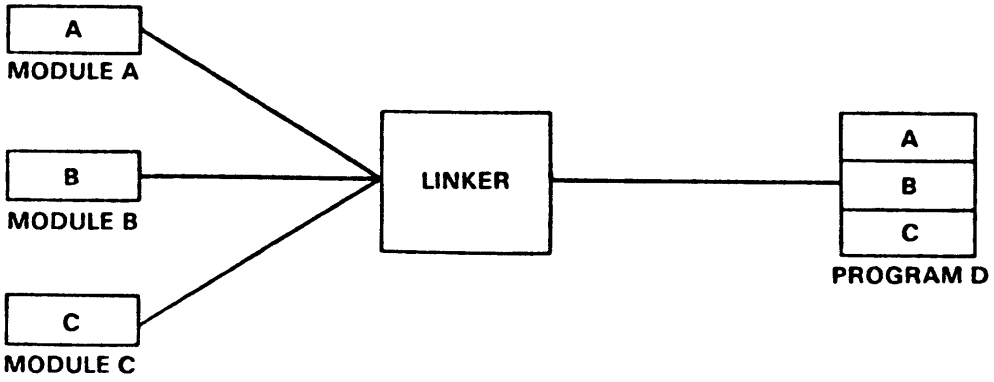

Figure 3-1. Creation of a Linked Program

3.2.2 Replacing a Program Module in a Linked Program

When you recompile one or more object modules of a linked program, you can use the LINKER to replace those modules in the linked program. Once you have linked the modules that make up the linked program, you do not need to link them again.

On each Input screen, you identify a program module that will replace a linked module. After you identify all replaced modules, provide the name and location of the linked program. You must specify the replaced modules individually before providing the name and location of the linked program.

On the Output screen, enter the name and location of the linked program. Specify the entry point of the linked program in the ENTRY field, then indicate whether or not the file named on the Output screen should be replaced. To replace the linked program, specify REPLACE=YES.

The following figure illustrates the replacement of a module in a linked program.

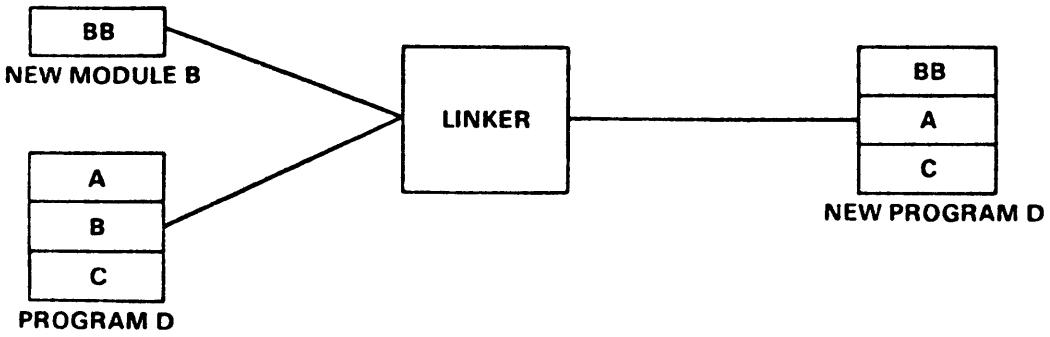

Figure 3-2. Replacement of a Module in a Linked Program

3.2.3 Removing Symbolic Debug Information from a Program File

When a program has been fully debugged, you can remove symbolic debug information to reduce the disk storage associated with the program and, possibly, to accommodate the linked program in Segment 1 address space.

This feature allows you to remove symbolic debug information from the file named on the Output screen. Few fields on other screens need to have values provided. From the Options screen, indicate that the debug information is not to be retained by specifying SYMB=NO.

From the Output screen, provide the name and location of the file whose symbolic debug information you want to remove. If you specify REPLACE=YES, the LINKER replaces the file named on the Output screen with the file whose symbolic debug information is removed. The LINKER creates an output file that contains no symbolic debug information.

3.3 INITIATING THE LINKER

You can run the LINKER through the Command Processor or through the EDITOR. There are minor differences between these two methods. The formats of the screens differ slightly, but all options are provided in both.

From the Command Processor, access the LINKER by pressing PF1 (RUN). From the Run screen, enter LINKER in the PROGRAM field. You do not need to enter the library and volume names because the LINKER is stored in the System Program Library on the System Program Volume.

From the EDITOR, the LINKER Options screen appears when you compile the work file with the special menu PF9 (RUN) or PF10 (COMPILE) function. If the compilation is successful, the EDITOR then calls the LINKER. For details on how to run the LINKER from the EDITOR, refer to Section 2.6.

3.4 LINKER SCREENS

The LINKER displays several screens during its execution. This section describes each parameter of the Options screen, the optional Library screen, the Input screen, and the Output screen.

3.4.1 Specifying LINKER Options

The LINKER first displays the Options screen. This screen requests one library and volume where program modules to be linked may reside, and offers other linking options. Descriptions of the fields on the Options screen follow:

Field	Description
LIBRARY VOLUME	The location of a subroutine library that contains one or more program modules to be linked. The default library and volume are INLIB and INVOL as defined with the SET command (PF2) of the Command Processor.
MORE	Indicates whether or not program modules to be linked are stored in more than one subroutine library. By specifying MORE=YES, you can identify up to eight subroutine libraries. These libraries are specified on the Library screen, discussed in section 3.4.2. The default is NO.
SYMB	Indicates whether or not symbolic debug information will remain in the output file. The default (YES) retains symbolic debug information in the output file.
MAP	Indicates whether or not to print a Link map. A Link map identifies the names of all code and static sections in the linked program, identifies entry points in each, and also identifies the length of each section. The default (YES) prints a Link map. The Link map is described in section 3.5.1. A sample map is provided in Appendix D.
XREF	Indicates whether to print an External Reference map. This map indicates all references to external symbols. The default (NO) does not print an External Reference map. The External Reference map is described in section 3.5.2. A sample map is provided in Appendix E.

EXSEC — Allows you to include a listing if those sections that were excluded because of duplicate names when you enter YES. The default is NO. The excluded sections and the names of the files in which they occur are listed in the order they are found.

LINES — The number of lines per page for the Link map and the External Reference map. The default is 55.

3.4.2 Specifying Additional Subroutine Libraries

If you link program modules from more than one subroutine library (indicated by specifying MORE=YES on the Options screen), the LINKER displays the Library screen after the Options screen.

This screen provides eight pairs of LIBRARY and VOLUME fields. Use these fields to enter the appropriate subroutine library and volume names. The first library and volume are the same as those specified on the Options screen. These values can be overridden.

The LINKER attempts to satisfy external references from files located in libraries specified on this screen in the order named.

3.4.3 Specifying Program Modules to be Linked

After you complete the Options screen and, if appropriate, the Library screen, the LINKER displays an Input screen.

Each Input screen requests the file, library, and volume names of a program module to be linked. The LINKER continues to display Input screens until you press only the ENTER key when a new Input screen appears.

The LINKER satisfies external references from input files first, then searches subroutine libraries for any unsatisfied external references. Therefore, Input screens should provide file names and locations of the main program, of files that are not in named subroutine libraries, and of files that would not be linked because of the order of subroutine libraries specified. It is not necessary to name every file you want to link.

3.4.4 Specifying the Linked Output Program

The Output screen requests the name of the output linked program and the entry point that initiates processing of the linked program or module. A description of each field on the Output screen follows:

Field	Description
FILE LIBRARY VOLUME	The file, library, and volume names for the output (linked) program, which is the executable program that contains all the linked program modules. The default library and volume are OUTLIB and OUTVOL, as defined by the SET command (PF2) of the Command Processor.
RETAIN	The retention period for the linked program. Only the file owner or a security administrator can delete or rename a file during this period.
FILECLAS	The file protection class for the linked program file. Refer to the VS Programmer's Introduction for further information about file classes.
ENTRY	The name or address of the point at which execution of the output file begins.
REPLACE	Indicates whether or not a file specified on an Input screen with the same name as the output file is to be replaced by the output file. The default is NO.

3.5 MAPS PRODUCED BY THE LINKER

The LINKER Options screen allows you to print the Link map and the External Reference map. Each map is described in detail in the following sections. Additionaly, the following discussions refer to the four sections of a program file (code, static, symbolic, and linkage). Refer to Chapter 4 for a complete discussion of these topics.

Regardless of the map options you select, the LINKER provides the name, location, and size of each input file; all link error messages; and the name, location, and size of the output file on a print file whose disposition depends on your default print mode.

3.5.1 Link Map

Specify MAP=YES on the Options screen to produce a copy of the Link map. The copy is stored on a print file with the prefix LINK in your print library. (If the LINKER is run from the EDITOR, the print file name starts with EDIT.) Its disposition depends on your print option. A sample Link map is included in Appendix D.

Code Section Information

The LINKER provides the following information about each code section:

Field	Description
NAME	The name of the code section. Names are constructed as follows: Assembler — CODE statement label BASIC — Main program - MAIN Subroutine - the subroutine name COBOL — PROGRAM-ID FORTRAN — Main program - MAIN Subroutine - the subroutine name PL/I — #xxx where xxx is the name of the first external procedure in the program RPG II — Columns 75 - 80 of the Header (H) spec, or RPGOBJ, if no value is specified in columns 75 - 80, or if no Header spec is included
ORIGIN	The virtual address in hexadecimal at the beginning of the code section
LENGTH	The hexadecimal length, in bytes, of the code section
ENTRY POINTS NAME/LOCATION	The names and locations of the entry points in the code section
MADE BY	The compiler or Assembler that produced the code section. Abbreviations are as follows: AS Assembler BS BASIC CB COBOL FT FORTRAN PL PL/I RP RPG II
VERSION	The version of the compiler or Assembler that produced the code section
DATE	The date of compilation or assembly of the code section

Static Section Information

The LINKER provides the following information about each static section:

Field	Description
NAME	The name of the static section is constructed as follows:
	ASSEMBLER — STATIC statement label
	BASIC — Main program - $MAIN Subroutine - $ + first 7 characters of name
	COBOL — $ + first 7 characters of PROGRAM-ID
	FORTRAN — $ + first 6 characters of module name
	PL/I — $xxx where xxx is the name of the first external procedure in the program
	RPG II — $ + first 7 characters of module name
ORIGIN	The displacement of the static section in hexadecimal bytes, from the beginning of the static area of the program
LENGTH	The hexadecimal length of the static section
ENTRY POINTS NAME/LOCATION	The names and locations of the entry points in the static section
MADE BY	The compiler or Assembler that produced the static section. The abbreviations are as follows:
	AS Assembler BS BASIC CB COBOL FT FORTRAN PL PL/I RP RPG II
VERSION	The version of the compiler or Assembler that produced the static section
DATE	The date of compilation or assembly of the static section

Undefined Symbols

The map provides a list of all program modules that were referenced but not linked. This section might indicate misspelled names or missing code.

Link Statistics

The following link statistics are provided:

- The total length (hexadecimal) of the code and static sections
- The amount (in kilo bytes) of Segment 1 and Segment 2 address space used by the code and static sections
- The virtual address (hexadecimal) of the entry point to the linked program
- The name and location of the linked program, the output of the LINKER
- The size (decimal) of the linked program in records (1 record = 1024 bytes)

3.5.2 External Reference Map

The External Reference map also provides link information. A sample External Reference map is included in Appendix E. The map contains the following information:

Linked Program Modules

The LINKER lists the file, library, and volume names for each input file that is linked. This section is generated even if a Link map is not printed.

External References

The External References table provides the following information for the code and static sections of the linked program:

Field	Description
LOCATION	The virtual address of the external reference. An address prefixed with an asterisk (*) indicates an address in the static area relative to the start of the static area.
SYMBOL	The name of the externally referenced entry point. A name followed by the # character indicates a system routine.
SECTION	The section name in which the SYMBOL is an entry point. An asterisk before the name indicates a symbol in the static section.
FLAGS	The flag associated with each relocation item in the relocation reference block. Each flag consists of two digits. Together, they describe the relocation reference item. The two flag bytes are described as follows:

First Byte

Value	RCON Flag	Relocation Record Flag
0	F	F
1	F	T
2	T	F
3	T	T

RCON Flag: F (false) signifies that the object of relocation in the RUN block is an absolute address constant; T (true) indicates that it is an RCON (relative address constant).

Relocation Record Flag: F (false) indicates that the object of relocation in the RUN block is not a relocation record; T (true) indicates that it is a relocation record.

Second Byte

Value	Length	Direction	Resolved
0	3	P	T
2	3	P	F
4	3	N	T
6	3	N	F
8	4	P	T
A	4	P	F
C	4	N	T
E	4	N	F

Length: Indicates the length in bytes of the object of relocation.

Direction: Indicates the direction of relocation: P indicates positive, N indicates negative.

Resolved: Indicates whether the referenced name is resolved. (T) indicates that it is resolved; (F) indicates that it is not.

Link Statistics

The link statistics are the same as those provided by the Link map. Refer to section 3.5.1.

3.6 SUPPRESSION OF PRINT OUTPUT

You can suppress the print output (including the Link map and the External Reference map) only if you run the LINKER with a procedure. Include the following statement in the procedure that is running the LINKER: ENTER PRNTFILE PRINT=NO. This statement follows the output specification.

For example, the following steps link programs X and Y into program Z and suppress the print output:

```
PROCEDURE SUPPRESS
RUN LINKER
ENTER INPUT  FILE=X
ENTER INPUT  FILE=Y
ENTER INPUT
ENTER OUTPUT FILE=Z
ENTER PRNTFILE  PRINT=NO
RETURN
```

3.7 LINKER RETURN CODES

The following are the error return codes associated with the LINKER:

Return Code	Meaning
0	Successful link
4	Warning message: a. Unsatisfied External Reference(s). b. Labeled COMMON sections of unequal lengths encountered (FORTRAN only).
8	Error message: a. Multiple entry points with the same name encountered. b. Static section in Segment 0 address space encountered. c. Initialized blank COMMON static section encountered (FORTRAN only). d. Illegal replacement attempted (possible only in program files generated in FORTRAN). This error occurs when the compiler encounters a code or static section with the same name as a labeled or blank COMMON section.

The print output of the LINKER contains warning messages or error messages indicating the section name(s) and the file(s) in which the section(s) were encountered.

CHAPTER 4
VS COMPILERS AND THE PROGRAM FILE

An essential step in program development is compilation or assembly. This chapter provides you with information about translating a source file into a program file and constructing the program file.

4.1 PROGRAM FILE

The programmer writes a source program, which the system stores as a source file. The system cannot execute a source program directly. First, it must translate that program into "machine code," or object code. The object code is stored as a program file. This process is called compilation when the program is written in a higher-level language (such as COBOL or BASIC). It is called assembly when the program is written in Assembly language.

The program file contains four sections that are important to the program development tools: the code section, the static section, the symbolic section, and the linkage section. These sections are important to the programmer during program execution, debugging, and linking. They are discussed later in this chapter.

On the VS, the code section occupies Segment 1, and the static section occupies Segment 2. The operating system occupies Segment 0, which you cannot access.

4.2 PROGRAM COMPILATION

You can compile or assemble a source program from within the EDITOR or by running the compiler or Assembler directly from the Command Processor. Compilation from within the EDITOR is discussed in Chapter 2. You can run the compiler or the Assembler from the Command Processor by selecting PF1 (RUN) and entering the name (ASSEMBLE, BASIC, COBOL, FORTRAN4, PLI, or RPG II) in the PROGRAM field of the Run screen. The compiler or Assembler then displays the Options screen (refer to the appropriate programming language reference manual). The Input screen identifies the source program you want to compile or assemble. The Output screen identifies the name of the program file, which contains the object code.

Compiler output (source listing and maps) is discussed in the programming language reference manuals. These manuals also contain the compilation or assembly return codes.

Compilation or assembly of an individual source program module creates a program file that may contain all four sections mentioned in Section 4.1. The code section is the object code for the module; the static section contains the data.

4.2.1 Contents of Code and Static Sections

For COBOL programs, the code section is the Procedure Division, and the static section is the Working Storage Area. For BASIC, FORTRAN, and PL/I, the code section contains program statements, and the static section contains the program variables. For RPG II the code section contains the calculation statements (C and W specs), and the static section contains the data items.

4.2.2 Naming Conventions for Code and Static Sections

Each code section and static section is assigned a name that is derived from the program module name. The rules for forming names are as follows:

Language	Section	Naming Rule
Assembler	Code section	CODE statement label
	Static section	STATIC statement label
BASIC	Code section	MAIN for main program subroutine name for subroutines
	Static section	$MAIN for main program $ plus first 7 characters of subroutine name for subroutines
COBOL	Code section	PROGRAM-ID
	Static section	$ plus first 7 characters of PROGRAM-ID
FORTRAN	Code section	MAIN for main program subroutine name for subroutines
	Static section	$ plus first 6 characters of module name
PL/I	Code section	# plus the name of the first external procedure in the program
	Static section	$ plus the name of the first external procedure in the program
RPG II	Code section	Columns 75 - 80 of the Header (H) spec or, RPGOBJ, if no value is specified in those columns, or if no Header spec is included
	Static section	$ plus first 7 characters of module name

You may encounter unpredictable problems if you attempt to link modules with similar section names. For example, the static section names for two BASIC subroutines are named MODULE15 and MODULE16 are not unique because the system uses only the first seven characters (MODULE1) to determine the name. In this case, both are named $MODULE1.

If two programs have the same static section name when linked, the LINKER retains the contents of the section it encounters first, and the length of the longer of the two sections. For code sections, the LINKER retains contents of the section encountered first, and the length of that section, even if it is shorter.

4.3 PROGRAM LINKING

The LINKER combines two or more program modules to form a linked program. The LINKER produces a code section for the linked program that consists of code sections for all linked modules that are arranged in the linked order. The static section for the linked program consists of static sections for all linked modules, also arranged in the order linked. The code section for the linked program precedes its static section.

The linked program file also contains a linkage section, which is used when the LINKER replaces a module in a linked program. The linkage section contains the following information:

- Names of all entry points in the program file
- Offsets into the code section of the program file for each individual module's code section
- Offsets into the static section of the program file for each code section
- Names of all external references in the file

4.4 SYMBOLIC DEBUGGING

You can cause a compiler to create a symbolic section for a program module by specifying SYMB=YES on the compiler Options screen. Including Symbolic Debug information does not affect the program's execution; instead it enables you to use the Symbolic Debugger to debug the program. The symbolic section contains a pointer to the program listing and a description of attributes for program variables.

When you remove this section from the program file, you cannot view the program listing while debugging. Also, you cannot refer to variables by name. You must reference them by memory location (if they can be found).

CHAPTER 5
SYMBOLIC DEBUGGER

5.1 INTRODUCTION

Debugging is the process of locating and correcting errors in a program. The VS Symbolic 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 Symbolic Debugger allows you to reference variables by the names used in the program instead of using data addresses to search through memory.

The Symbolic Debugger permits you to monitor the behavior of a program while it is executing. Also, you can inspect and modify values of variables, set break points (known as traps), and alter the flow of execution of the program.

5.2 HOW THE SYMBOLIC DEBUGGER WORKS

The VS programming languages place symbolic information into the symbolic block of the program file during compilation or assembly. The symbolic block contains information relating to the static section and the code section of the program. In the static section, the block contains the data item names and their offsets into the static section. The code section of the symbolic block contains the verb you want to execute, the line number of the statement that is executing, and the offset of the instruction into the code section.

When you run the Symbolic Debugger, the source code appears as the program executes. Four possible conditions could prevent the source code from being displayed. First, you might not have selected the SYMB option at compilation or assembly (on the compiler/assembler Options screen). Second, execution of the program might have been interrupted during the processing of a routine for which the source code is not available (such as a system or external subroutine). Third, the Symbolic Debugger might not be able to access the program source code. Fourth, the LINKER might have removed the symbolic debug information from the program file.

5.3 USING THE SYMBOLIC DEBUGGER

Both you and the system can execute the Symbolic Debugger. If you run the program from the Command Processor, you can access the Symbolic Debugger at the beginning of program execution. First, select the RUN command (PF1)

from the Command Processor. Then, specify the program name, and press PF1 (instead of ENTER). You can also access the Symbolic Debugger after you have interrupted execution with the HELP key. To do so, press PF10 from the Modified Command Processor screen.

When a program error is encountered, the system displays the Cancel Processing screen and allows you to run the Symbolic Debugger.

To use all the features of the Symbolic Debugger, you must compile the program with SYMB=YES specified on the compiler or Assembler Options screen. The symbolic debug information must not be removed from the program file by the LINKER. In addition, the source listing must be available. A source listing is available if it (1) is in a file class that is available to you, (2) is on a volume that is currently mounted, (3) is a print file (COBOL, PL/I, and RPG II) and has a status of HOLD or KEEP, (4) is a source file (BASIC and FORTRAN) that is not currently in use, or (5) is a debug file (Assembler) that is accessible.

5.4 DEBUG PROCESSOR SCREEN

When you execute the Symbolic Debugger, the Wang VS Debug Processor screen appears. This screen is the Debug menu, and consists of the following parts:

Window — The window displays seven lines of the program currently being executed and debugged. The center line in the window is the line to be executed next unless the window shows the first or last seven lines in the program. If the source code is not available, a message appears.

Header — The header is the screen label; Wang VS Debug Processor is the header for the Debug menu.

Message — The message describes the last debug operation. When the Symbolic Debugger is run, the message indicates that the user program is loaded. When the Symbolic Debugger returns to the Debug menu because it encounters a trap, the message field identifies the trap type. If an error is encountered, the message field provides the error message and an error description.

Trap Area — The traps area lists the trap types, their locations within the program, and the values of any set counters. This area is blank until you set traps.

Status Area — The status area indicates the current program status by displaying four parameters with their current values. These parameters include the code section currently executing, the Statement # (sequence number) of the source program statement you will execute next, the statement verb, and the current Program Control Word (PCW).

Options Area The options area lists the PF keys and Symbolic Debugger functions available from the screen.

If you attempt to debug a program that is execute-only, the message "Sorry, Current User does not have access to Debug facilities for this program" appears, and PF2 through PF6 are disabled.

5.5 DEBUG MENU OPTIONS

A description of the Debug menu options follows:

PF1 (CONTINUE) Causes program execution to continue.

PF2 (PREVIOUS) Causes the previous seven source lines to be displayed in the window. The contents of the window do not affect program execution; the next program step is identified in the status area.

PF3 (NEXT) Causes the next seven source lines to be displayed in the window. The contents of the window do not affect program execution; the next program step is identified in the status area.

PF4 (TRAP) Selects a trap or allows you to modify or clear a previously set trap. The Symbolic Debugger provides a Statement Breakpoint trap, a Single-Step trap, and a Modification trap. Each trap is discussed in detail in section 5.6.

PF5 (INSPECT/MODIFY) Enables you to inspect and modify memory, program registers, or the PCW. You can use the Inspect/Modify screen to reference data items by the symbol used in the program. The Inspect/Modify function is described in section 5.7.

PF6 (SELECT SECTION) Enables you to display the source code of a different code section. This function also enables you to change the default section name for Inspect/Modify functions.

PF13 (DUMP) Enables you to obtain a full Segment 2 memory dump. This function creates a print file of a program's variables, buffers, and control blocks.

PF14 (PRINT PROGRAM SCREEN)

Enables you to create a print file that consists of the program screen displayed immediately before the interruption of program execution.

PF15 (PRINT DEBUG SCREEN)

Prints the debug screen currently displayed. This option is available for printing any of the debug screens, even when PF15 is not displayed in the current menu.

PF16 (CANCEL PROCESSING)

Cancels the interrupted program and returns control to the Command Processor or the program from which the interrupted program was run.

5.6 TRAP FUNCTION

When you select the Trap function (PF4) from the Debug menu, the Symbolic Debugger displays the Trap screen. Select one or more traps by specifying values for the parameters in the appropriate sections of the screen. When you set a trap, a message identifying the trap type and parameters appears on the Debug menu screen in the traps area. When the Symbolic Debugger encounters a trap during program execution, the debugger interrupts program execution and displays a message identifying the trap. You can then use other Symbolic Debugger features.

You can clear a trap by erasing the values that are displayed for its parameters.

Three different trap types are available. You can use these trap types alone or in combination with others.

5.6.1 Statement Breakpoint Trap

This trap interrupts execution at the specified program statement (or machine instruction in Assembler). When you encounter this trap, execution stops immediately before the statement at which the trap is set. The parameters that apply to this trap are described as follows:

Field	Description
LINE NUMBER	Identifies the program statement at which execution will stop. You cannot use this field if symbolic debug information is not available. You can specify either LINE NUMBER or OFFSET, but not both.

CODE SECTION — Identifies the program module in which the line number or offset is located. The default is the section currently executing. If you interrupt execution during a run-time subroutine, the section name is the system-supplied subroutine.

OFFSET — Instead of specifying a line number at which to interrupt execution, you can specify a hexadecimal address in memory (if CODE SECTION is blank) or a hexadecimal offset in the specified code section. Program execution is interrupted immediately before the machine instruction located at the specified offset. You can specify either LINE NUMBER or OFFSET, but not both.

COUNTER1 — The number of times that the program statement (or instruction) at which the trap is set will be encountered before the trap is taken. The default (COUNTER1=1) interrupts execution the first time the specified line or instruction is encountered.

5.6.2 Single-Step Trap

The Single-Step trap interrupts execution after you have executed a specified number of statements or instructions. The parameters for this trap are described as follows:

Field	Description
COUNTER2	Number of statements or instructions to be executed before interrupting execution.
TYPE	Set to S for program source statements, or A for machine instructions.

5.6.3 Modification Trap

The Modification trap halts execution when a specified byte in memory is changed. The parameters applicable to this trap are described as follows:

STATIC SECTION3 — Static section of a program. The naming convention for static section names is included in section 4.2.2. This parameter can be blank if you use OFFSET3 to specify an address.

OFFSET3 — Offset (in hexadecimal) from the start of the named section that indicates the byte you want to monitor.

You can only modify variable data in the program static section. Also, the trap is taken only if the specified byte is actually changed. For example, if the hexadecimal value of a variable is "104F" and it is changed to "1000", the trap is not taken because the first byte (X'10') has not been

modified.

5.7 INSPECT/MODIFY FUNCTION

The following describes the options available when you select the Inspect/Modify function (PF5) from the Debug menu:

PF Key	Option and Description
ENTER	Inspects the data item whose name and code section have been specified. This option is discussed in detail in section 5.7.1.
1	Returns to the Debug menu.
2	Displays the previous seven source lines in the window.
3	Displays the next seven source lines in the window.
10	Selects the Inspect and Modify Memory option, which enables you to inspect memory and modify unprotected memory. The option is described in detail in section 5.7.2.
11	Selects the Inspect and Modify Registers and the Program Control Word option. This enables you to inspect machine registers and the PCW, and then modify them. The option is discussed in detail in section 5.7.3.
12	Selects the CALL/LINK/SVC Trace option, which enables you to trace through all previous calls to subroutines, other programs, and supervisor calls. The option is discussed in detail in section 5.7.4.
14	Displays the most recent screen of the executing program. Press PF15 to print the screen; any other response returns you to the Inspect and Modify screen.

5.7.1 Inspect Symbolic Data/Character String

You can examine the value of any program variable by entering the variable name and section in the appropriate fields of the Inspect and Modify screen and pressing ENTER. The Symbolic Debugger displays the Inspect Symbolic Data screen if the data is numeric type or the Inspect Character String screen if the data is character type. The screens have the following fields:

Field	Description
DATANAME	Name of the variable as it is referenced in the program.

SECTION Name of the program module containing the variable. The default value of this field is the name of the module whose execution was interrupted.

The value of the variable appears below the variable name in a form appropriate for the type of the variable. Bit string data appears in hexadecimal form.

The options available from this screen are described as follows:

PF Key	Option	Description
ENTER	Inspect	Enables you to display the value of a data variable by entering the variable name and code section and pressing ENTER.
1	I/M Menu	Returns to the Inspect and Modify menu.
2	Prev Source	Displays the previous seven lines of source in the window.
3	Next Source	Displays the next seven lines of source in the window.
4	Prev	Displays the previous screen of data if the amount of data is greater than the capacity of the screen.
5	Next	Displays the next screen of data if the amount of data is greater than the capacity of the screen.
6	HEX/Char Notation	Displays the value of the named variable in either hexadecimal or character form. Selection of this option allows you to display the value in the other form.
7	Modify	Displays the Modify Symbolic Data screen or the Modify Character String screen and permits you to modify the value of the variable. If you attempt to display nondisplayable characters, the following message appears: "Warning - Nondisplayable characters (and Blanks) are displayed as *. To retain their current values, Modify only in 'hex' mode."

5.7.2 Inspect and Modify Memory

The Inspect Memory screen enables you to examine memory and modify unprotected memory. The fields available on this screen are described as follows:

Field	Description
OFFSET	Memory location at which the display of memory begins. The offset is added to the base address to produce a starting location for the memory display. Entering a blank is the same as entering a zero.
SECTION	Code section in which the OFFSET field operates. You can display memory contents by entering the name of the code section, or by entering the base address of a block of memory in the BASEADR field. This field may be left blank.
LENGTH	Number of bytes of memory to be displayed on the screen. LENGTH is used to specify how many bytes will be displayed each time you press the ENTER key. The default is hexadecimal 100 (256 bytes). Each time you press the ENTER key, the next block of bytes whose size is specified by LENGTH is displayed, the OFFSET field is incremented by this amount.
BASEADR	Base address. The offset is added to this address. You can specify the base address in hexadecimal. You can also specify the base address as a register (R) followed by a hexadecimal digit (e.g., R0, R2, RF) to indicate that the contents of that register will be used as the base address. When you press ENTER, the debugger replaces the register by the value contained in that register.

The options available on this screen are described as follows:

PF Key	Option	Description
ENTER	Display Memory	Displays memory defined by the above fields in both hexadecimal and character notation.
1	I/M Menu	Returns to the Inspect and Modify menu.
2	Inspect Regs/PCW	Selects the Inspect Registers and Program Control Word screen, discussed in Subsection 5.7.3.
3	Change Displayed Data	Allows you to modify the displayed memory. When you select this function, you modify the highlighted memory values. You can only modify data in Segment 2 (addresses 200000 and higher). From this screen, ENTER changes the modified memory and returns to the Inspect Memory screen. PF1 returns to the Inspect and Modify screen (section 5.7.2); PF2 returns to the Inspect Memory screen (section 5.7); PF3 selects the Inspect Registers and Program Control Word screen (section 5.7.3).

5.7.3 Inspect and Modify Program Registers and the Program Control Word

The Inspect Registers and Program Control Word screen enables you to examine and modify both the program registers and the PCW. The screen displays the PCW, the General Registers, and the Floating Point Registers, and offers the following options:

PF Key	Option	Description
1	I/M Menu	Returns to the Inspect and Modify menu.
2	Inspect Memory	Displays the Inspect Memory screen and enables you to examine and modify memory. This screen is described in section 5.7.2.
3	Modify Regs	Displays the Modify Registers screen and enables you to modify highlighted values for program registers. This screen is described in section 5.7.5.
4	Modify PCW	Displays the Modify PCW screen and enables you to modify values in the PCW. This screen is described in section 5.7.6.

5.7.4 CALL/LINK/SVC Trace

The Inspect Program, Subroutine, and SVC Linkage screen enables you to display a list of calling programs, subroutines, or SVCs prior to the current program. The last call issued before the program was interrupted appears first. When you press ENTER, the system steps backwards through the previous calls and displays the following:

- Subroutine calls (all high-level language and Assembler calls made through the JSCI instruction)
- Program linkages (through the LINK SVC or Procedure language RUN statement)
- SVCs (Supervisor Calls)

In addition, the address of the call and the calling program's registers at the time of the call are displayed. By convention, the registers in the Save area contain the following information:

Register	Description
0	General usage
1	Argument list pointer for use with CALL (JSCI) and LINK SVC
2 - 13	General usage
14	Pointer to the beginning of the first static area (not enforced)
15	Stack pointer (not displayed by TRACE)

5.7.5 Modify Registers

The Modify Registers screen allows you to examine the PCW, and examine and modify the General Registers and the Floating Point Registers. This screen provides the following options:

PF Key	Option	Description
ENTER	Chng Values in Regs	Changes the register values as you specify.
1	I/M Menu	Returns to the Inspect and Modify menu.
2	Inspect Memory	Returns to the Inspect Memory menu.
3	Inspect Regs/PCW	Returns to the Inspect Registers and Program Control Word menu.
4	Modify PCW	Displays the Modify PCW screen and enables you to modify the PCW. This screen is discussed in section 5.7.6.

5.7.6 Modify Program Control Word

The Modify PCW screen displays, and enables you to modify, values of the PCW. This screen provides the following options:

PF Key	Option	Description
ENTER	Chng Values in PCW	Modifies the PCW which you specify.
1	I/M Menu	Returns to the Inspect and Modify menu without changing the PCW.
2	Inspect without Memory	Returns to the Inspect Memory menu changing the PCW.
3	Inspect Program Regs/PCW	Returns to the Inspect Registers and Control Word menu without changing the PCW.

APPENDIX A
EDITOR EXAMPLE

The COBOL program example in this appendix contains a number of errors and uses several EDITOR functions to correct them. After you read and follow the example on your workstation, you can use these EDITOR functions: entering a new program, editing, compiling, and executing a program, and saving or replacing a file on disk. Chapter 2 discusses all of the EDITOR functions in detail.

```
000100  IDENTIFICATION DIVISION.
000200  PROGRAM-ID.  SAMPLE.
000300  ENVIRONMENT DIVISION.
000400  CONFIGURATION SECTION.
000500  SOURCE-COMPUTER.  WANG-VS.
000600  OBJECT-COMPUTER.  WANG-VS.
000700  IIPUT-OUTPUT SECTION.
000800  FILE-CONTROL.
000900      SELECT LISTING
001000          ASSIGN TO "OUTPUT", "PRINTER",
001100          ORGANIZATION IS SEQUENTIAL,
001200          ACCESS MODE IS SEQUENTIAL.
001300  DATA DIVISION.
001400  FILE SECTION.
001500  FD  LISTING
001600  FD  LISTING
001700      LABEL RECORDS ARE OMITTED.
001800  01  PRINT-RECORD     PIC X(132).
001900  WORKING-STORAGE SECTION.
002000  77  PRINT-LINE       PIC X(132) VALUE SPACES.
002100  PROCEDURE DIVISION.
002200  START-PROGRAM.
002300  PERFORM INITIALIZATION.
002400      PERFORM WRITE-A-PRINT-LINE.
002500      PERFORM TERMINATION.
002600  EXIT-PROGRAM.
002700      STOP RUN.
002800  INITIALIZATION.
002900      OPEN OUTPUT LISTING.
003000      MOVE SPACES TO PRINT-LINE.
003100  WRITE-A-PRINT-LINE.
003200      MOVE "YOUR PROGRAM PRINTED THIS LINE. CONGRATULATIONS!"
003300          TO PRINT-LINE.
003400      WRITE PRINT-RECORD FROM PRINT-LINE
003500          AFTER ADVANCING 1 LINE.
003600  TERMINATION.
003700      CLOSE LISTING.
```

Invoking the EDITOR

From the Command Processor, select the RUN command (PF1). The next screen requests a program name, and the library and volume locations. In the PROGRAM field, type EDITOR and press ENTER. It is not necessary to enter library and volume names since the EDITOR resides in the System Library on the System Volume.

The Input screen is the first screen to appear. On this screen, you specify the programming language and the name of the input file. In the LANGUAGE field, type COBOL. Since you are creating a new file, leave the FILE field blank. Press ENTER.

EDITOR Main Menu

The EDITOR now displays the Main menu. This menu offers 16 editing commands, and displays a highlighted message that the editing work file is being created. Examine the menu to become familiar with the functions offered at this level of the EDITOR.

The functions on the Main menu are divided into three types:

Functions	PFkeys
display	1 - 8
edit	9 -14
special	15-16

The most commonly used functions include the DISP function (PF1), the PREV and NEXT functions (PF4 and PF5), the MOD function (PF9), the INS function (PF11), the DEL function (PF12), and the MENU function (PF16).

Entering the Program

Wait a few seconds until the highlighted message leaves the screen. Press PF11 (INS) to enter the program in Insert mode. Examine the screen that appears. Several of the display functions are listed at the top of this screen. Column headings appear above the program area. The EDITOR produces line or sequence numbers to the left of each new line and positions the cursor at the first usable column in the first line.

The EDITOR automatically sets tabs for each language; the TAB key can be used to align statements as required by the language you are programming. While in Insert mode, type the first line of the program and then press ENTER. As you type, the newly entered characters are highlighted. You can backspace to any incorrect character and retype it. After you enter the line, press ENTER. The EDITOR produces a new line number and moves the cursor to the next line. In Insert mode, the EDITOR continues to produce a new line each time you press ENTER. Enter the remainder of the program. Remember to type the lines as they appear on the previous page, including the incorrect lines, since you will

correct these lines later. Once you enter the last line of the program and the cursor is at the start of the next line, press PF1 to exit Insert mode and return to Display mode.

Compiling the Program

You have entered the program. Now you will compile it and discover any diagnostic errors. From the main menu, press PF16 to select the EDITOR special menu. The special menu offers numerous functions. Examine this menu and the functions it offers.

Select the COMPILE command (PF 10) to compile your COBOL program. The EDITOR displays the compiler Options screen, which contains a number of options. For this sample program, the default options are sufficient. When you are ready to compile the program, press ENTER.

The LINKER Options screen appears next. This screen presents a number of options for the LINKER. The LINKER links a number of compiled program modules into a larger executable program. Since this program does not link with any external subprograms, it is not necessary to modify this screen, so press ENTER. (Refer to Chapter 3 of this manual for more information about the LINKER.)

The compiler now attempts to compile the program. During compilation, the COBOL compiler displays the message COBOL Compilation of ##TEST in Progress. The EDITOR assigns new files a name of ##TEST before they are compiled. Files whose names begin with # are temporary files and are scratched when you terminate EDITOR processing. A message appears: Program EDITOR in progress when compilation is complete.

When the EDITOR returns to the special menu, it highlights a new PF key, PF11 (ERRORS). A message at the top of the screen indicates that the program has compilation errors. (If the program compiles successfully, this PF key and message do not appear.) To examine the error listing, press PF11. Review the contents of this listing. Press PF5 (NEXT) to examine the next screen; PF4 (PREV) to examine the previous screen; or PF2 (FIRST) to return to the first listing screen.

The source listing contains three errors. The compiler may report other errors which have resulted because of the first three. The sample program contains the following three errors. You may have made other errors made while entering the program.

```
Line  700 INPUT misspelled
Line 1600 Line 1500 duplicated
Line 2300 Line should begin at B margin
```

Modifying the Program

While the error listing is on the workstation screen, move the cursor to Line 700 and press PF1. The EDITOR displays the work file beginning with that line.

Use Modify mode to correct the error in Line 700. Press PF9

(MOD). Notice that only PF keys 1 and 15 are available and that the current screen of the work file is highlighted on the workstation. You can correct errors in Modify mode by moving the cursor to the appropriate character and making the modification. Correct Line 700 and press ENTER.

Now edit Line 1600. You will delete this line. Return to the special menu by pressing PF16. Next, press PF11 to return to the error listing. Move the cursor to Line 1600 and press PF1. The program appears on the screen with Line 1600 at the top. Press PF12 to enter Delete mode. Read the top of the screen and note that you must enter the range of lines to delete. Since the cursor is at Line 1600, this is the default start line. Because Line 1600 is the only line you want to delete, it is not necessary to define the end line. Press ENTER.

To correct Line 2300, enter Modify mode again. Move to Line 2300 and position the cursor to the first character in the line (P). Press INSERT four times to insert four blanks before that character. When that character lines up with the following line, press ENTER to make the change and to return to Display. Now, recompile the program.

Recompiling the Program

Return to the special menu (PF16 from the main menu) and select the COMPILE function (PF10). The compiler does not request the compiler options since you defined them on the last run.

Your program has compiled successfully when the message "Compilation of ##TEST completed" appears. Select the RUN command (PF9) to execute the program. Press ENTER to execute your test program. Once the program executes, you return to the special menu and the message "Program ##TEST completed" appears. You should have a print file consisting of the program output stored in your print library (if your print mode is set to Hold or Keep).

Saving the File

Select the CREATE function (PF5) from the special menu to create a permanent disk copy of your corrected source file. Name your file, specify library and volume names if no defaults are displayed, then press ENTER. The special menu displays a message about your new permanent file and displays the REPLACE function (PF6) in the list.

APPENDIX B
LINKER EXAMPLES

The LINKER performs three functions:

1) It links two or more separate program modules into a single executable linked program;

2) It replaces a linked module in a linked program;

3) It removes symbolic debug information from a program module.

A separate example demonstrates each of the LINKER functions.

Linking Separate Program Modules into a Single Executable Linked Program

In this example, the LINKER links four program modules, which are stored in different subroutine libraries, into a linked program. The file names and their libraries and volumes are as follows:

File	Library	Volume	Comments
MYPROG	ABCOBJ	SYSVOL	The main program
FINSUB	FINLIB	SUBVOL	A financial subroutine
LEDGER	FINLIB	SUBVOL	A ledger subroutine
REPTSUB	FORMLIB	OUTPAK	A report generating subroutine

The output linked program, produced by the LINKER will be named LINKPROG, and will be stored in library ABCOBJ on volume SYSVOL.

Select the RUN command (PF1) from the Command Processor to invoke the LINKER. Enter LINKER in the PROGRAM field then press ENTER. It is not necessary to specify a library or volume because the LINKER resides in the System Library on the System Volume.

First, the LINKER displays the Options screen. In the LIBRARY and VOLUME fields enter ABCOBJ and SYSVOL, the library and volume of the main program. Specify that MORE equals YES because you will be linking program modules from more than one library. You can leave the remaining fields as they are. Press ENTER.

Next, the LINKER displays the Library screen. Here, you specify

the locations of other subroutines that will be linked. In the LIBRARY2 and VOLUME2 fields, enter FINLIB and SUBVOL. In the LIBRARY3 and VOLUME3 fields, enter FORMLIB and OUTPAK; then press ENTER.

The LINKER then displays an Input screen. Specify the file and library names of the main program, then press ENTER. The LINKER displays another Input screen. Since it is only necessary to specify one input file name, press ENTER without specifying any additional file information.

The Output screen appears next. Specify the name and location of the linked program on this screen. In this example, FILE equals LINKPROG, LIBRARY equals ABCOBJ, and VOLUME equals SYSVOL. Make certain that ENTRY=MAIN so that the starting point of the linked program is correctly identified. You can leave the remaining fields as they are. If you specify that REPLACE equals YES, the LINKER replaces an existing file named LINKPROG on ABCOBJ in SYSVOL (if this file exists).

The LINKER constructs the output linked program from the entered data and signals its completion. The result is a new file (LINKPROG, stored in ABCOBJ on SYSVOL), which can be run directly from the Command Processor.

Replacing a Program Module in a Linked Program

In this example, the linked program created in the above example is modified because one of the subroutines has been recompiled. The recompiled subroutine is FINSUB, stored in library FINLIB on volume SUBVOL.

On the Options screen, the LIBRARY and VOLUME fields identify the location of the linked program. For this example, LIBRARY equals ABCOBJ and VOLUME equals SYSVOL.

The first Input screen identifies the new version of the linked subroutine. For this example, define the parameters as follows: FILE equals FINSUB, LIBRARY equals FINLIB, and VOLUME equals SUBVOL. The second Input screen identifies the linked program. On this second Input screen, define the parameters as follows: FILE equals LINKPROG, LIBRARY equals ABCOBJ, and VOLUME equals SYSVOL. Since there are no other files to name, press the ENTER key when the third Input screen appears.

The Output screen identifies the linked program. For this example, FILE equals LINKPROG. The screen correctly names the program entry point so that field need not be changed. Finally, because the new linked program will replace LINKPROG, specify REPLACE equals YES.

The result of this procedure is a new linked program with the same name and location as the original LINKPROG, but with the replaced subroutine module.

Removing Symbolic Debug Information From a Program File

Once you are satisfied that LINKPROG has been fully debugged (or if you are certain that a backup copy exists), you can remove the symbolic

debug information.

On the Options screen, specify that SYMB equals NO. You do not have to provide any other information.

On the first Input screen, specify that FILE equals LINKPROG and provide its library and volume names. On the second Input screen, press ENTER without providing any information.

On the Output screen, specify the location for LINKPROG. Also, specify that REPLACE equals YES to replace the previous version of LINKPROG with the version without symbolic debug information. All symbolic debug information for this program is removed.

APPENDIX C
SYMBOLIC DEBUGGER EXAMPLE

A simple BASIC program demonstrates several features of the VS Symbolic Debugger. The program contains a logic error that causes an infinite loop. This problem is quite easy to find with the Symbolic Debugger. The program listing follows:

```
10  PRINT 'THIS IS THE PROGRAM START'
20  SUMTOTAL = 0
30  COUNTER = 0
40  SUMTOTAL = SUMTOTAL + COUNTER
50  COUNTER = COUNTER - 1
60  IF COUNTER=10 THEN 80
70  GO TO 40
80  PRINT 'SUMTOTAL = ', SUMTOTAL
90  END
```

The program logic is incorrect because statement 50 should increment rather than decrement COUNTER. Because of this error, the program continues to add increasingly negative values of COUNTER to SUMTOTAL. Control never passes to statement 80, which displays the value of SUMTOTAL.

First, you must enter the program into the system with the EDITOR. After you enter the program, select the special menu. Select the CREATE function (PF5) to assign a file name and save the file. Then, select the COMPILE function (PF10) from the special menu to compile the program. On the compiler Options screen, ensure that a source listing is printed (although the print file must remain in your print library) and that symbolic debug information is retained. If there are no diagnostic errors, terminate EDITOR processing.

From the Command Processor, press PF1 (RUN) and run the program. The program displays the message "this is the program start" on the workstation, then goes into the loop. Terminate processing by pressing HELP, then PF16 and ENTER. From the Command Processor, press PF1 to run the program again. After you enter the program name, press PF1 to initiate Symbolic Debugger processing.

The Debugger displays the Main Debug screen and awaits a command. Notice that the first seven program statements appear in the Debugger window. Also, a message indicates that statement 10 is the next statement that will be executed (that statement number appears in the "Statement #" field on the right center area of the screen).

Set a stopping point in the program after 10 iterations to check the values of COUNTER and SUMTOTAL. From the Debug menu, press PF4 (TRAP) to display the Trap screen. Set a Statement Breakpoint trap at Line 60 and set COUNTER1=10. This causes execution to proceed until Statement 60 is about to execute for the tenth time. Press ENTER to return to the Debug menu, then press PF1 (CONTINUE) to initiate processing.

When statement 60 is reached for the tenth time, the Symbolic Debugger interrupts processing and issues a message indicating that the trap is taken (which means that the trap condition is met).

Check the values of SUMTOTAL and COUNTER to verify that an infinite loop exists and to discover the cause. Select PF5 (Inspect/Modify) from the Debug menu. The Symbolic Debugger displays the Inspect Symbolic Data screen and is ready to display the value of any variable whose name is entered. Enter SUMTOTAL in the DATANAME field and press ENTER. The value -45.0 appears on the screen. The fact that this value is negative indicates a logic error in the program. Now, enter COUNTER in the DATANAME field and press ENTER to display its value. The Symbolic Debugger displays the value -10.0 and indicates that COUNTER is not incrementing in the program.

If you suspect that Statement 50 is incorrect, you can change the value of COUNTER to 11 and see how the program runs. Press PF7 (MODIFY) from the Inspect Symbolic Data screen. Enter 11.0 in the value field, and press ENTER.

Now, return to the Debug menu by pressing PF1 twice, then press PF4 to set a new trap. On the Trap screen, select the Single Step trap and set COUNTER2 to equal 1 and TYPE to equal S. This procedure causes the program to interrupt execution after each statement executes. Return to the Debug menu and press PF1 to continue processing. Notice how the program handles the new data. The program statements in the window change each time the Symbolic Debugger interrupts program execution and that the Statement # field also changes. Press PF1 to continue processing each time the Debugger interrupts execution after you have examined the screen that appears. The program should step through Statements 70, 40, 50, and 60, then transfer to Statements 80 and 90. Finally, end execution.

Correct Statement 50, then compile and run the program to verify that the problem is solved.

APPENDIX D
LINK MAP

This sample Link map illustrates LINKER output that describes the sample link performed in Appendix B. The main program is MYPROG, stored in library ABCOBJ on volume SYSVOL. The subroutines include FINSUB, in FINLIB on SUBVOL; LEDGER, in FINLIB on SUBVOL; and REPTSUB, in FORMLIB on OUTPAK. The Link map is described in detail in section 3.5.1.

```
WANG VS LINKER                                                    PAGE 1

        INPUT FILE MYPROG    IN LIBRARY ABCOBJ    ON VOLUME SYSVOL

        INPUT FILE FINSUB    IN LIBRARY FINLIB    ON VOLUME SUBVOL

        INPUT FILE LEDGER    IN LIBRARY FINLIB    ON VOLUME SUBVOL

        INPUT FILE REPTSUB   IN LIBRARY FORMLIB   ON VOLUME OUTPAK
```

CODE SECTION			ENTRY POINTS				
NAME	ORIGIN	LENGTH	NAME	LOCATION	MADE BY	VERSION	DATE
MAIN	100008	0000E0			CB	03.01.01	10/26/81
FEXIT	100198	000008			CB	03.01.01	7/28/81
FRDWF#	1002B0	000A00			CB	03.01.01	7/28/81
			FWRWF#	1002BA			
			FIOLF#	1002C4			
FCVZI#	1031B8	0001C0			CB	03.01.01	7/28/81
			FCVZO#	1032EA			
FINSUB	103378	000090			CB	03.01.01	7/28/81
LEDGER	103408	000125			CB	03.01.01	7/28/81
FIXPI#	10352D	0000F8			CB	03.01.01	7/28/81

STATIC SECTION			ENTRY POINTS				
NAME	ORIGIN	LENGTH	NAME	LOCATION	MADE BY	VERSION	DATE
$MAIN	*000000	000068			CB	03.01.01	7/28/81
$FCVF	*002110	0000F0			CB	03.01.01	7/28/81
$FINSUB		0022D0	000040		CB	03.01.01	7/28/81
$LEDGER		002310	000050		CB	03.01.01	7/28/81
$REPTSUB		002360	000110		CB	03.01.01	7/28/81
$FIXPI#		002470	000000		CB	03.01.01	7/28/81

```
TOTAL LENGTH:  CODE SECTIONS  STATIC SECTIONS
            013590    007360

THIS PROGRAM OCCUPIES    54K BYTES OF SEGMENT 1 ADDRESS SPACE
        AND      16K BYTES OF SEGMENT 2 ADDRESS SPACE

ENTRY POINT:   NAME LOCATION
     MAIN   100008

PROGRAM STORED IN
 FILE MYPROG   IN LIBRARY ABCOBJ   ON VOLUME SYSVOL
LENGTH =  78 RECORDS
```

APPENDIX E
EXTERNAL REFERENCE MAP

This sample External Reference map illustrates LINKER output that describes the sample link performed in Appendix B. The main program is MYPROG, which is stored in library ABCOBJ on volume SYSVOL. The subroutines include FINSUB, in FINLIB on SUBVOL; LEDGER, in FINLIB on SUBVOL; and REPTSUB, in FORMLIB on OUTPAK. The External Reference map is described in detail in section 3.5.2.

```
WANG VS LINKER                                                PAGE 1

        INPUT FILE MYPROG    IN LIBRARY ABCOBJ     ON VOLUME SYSVOL

        INPUT FILE FINSUB    IN LIBRARY FINLIB     ON VOLUME SUBVOL

        INPUT FILE LEDGER    IN LIBRARY FINLIB     ON VOLUME SUBVOL

        INPUT FILE REPTSUB   IN LIBRARY FORMLIB    ON VOLUME OUTPAK
```

LOCATION	REFERS TO SYMBOL	IN SECTION	FLAGS
100028	$MAIN	*$MAIN	28
100060	$MAIN	*$MAIN	28
10018C	FIOCS#	FIOCS#	08
1002A8	$FSTOP#	*$FSTOP#	28
103398	$FINSUB	*$FINSUB	28
*00000C	MAIN	MAIN	18
*000010	LEDGER	LEDGER	18
*00001C	FENDF#	FRDWF#	18
*000020	REPTSUB	REPTSUB	18
*002324	IBEXI#	IBEXI#	18

```
TOTAL LENGTH:  CODE SECTIONS  STATIC SECTIONS
            013590    007360

THIS PROGRAM OCCUPIES    54K BYTES OF SEGMENT 1 ADDRESS SPACE
        AND    16K BYTES OF SEGMENT 2 ADDRESS SPACE

ENTRY POINT:   NAME LOCATION
     MAIN   100008

PROGRAM STORED IN
 FILE MYPROG   IN LIBRARY ABCOBJ    ON VOLUME SYSVOL
LENGTH =  78 RECORDS
```

APPENDIX F
VS COMPILERS, ASSEMBLER, LINKER, AND EDITOR GETPARM REQUESTS

The following tables contain the prnames, keywords, and options used by GETPARM requests for the VS system programs documented in this manual. Please note the following conventions when using these tables:

∅	Denotes a blank. Whenever the last option given in an ENTER statement is a blank, the blank must be enclosed by quotation marks.
PF Keys	Do not use a keyword when entering options listed under "PF Keys" in the KEYWORD column. The PF key number alone is specified in the ENTER statement.
ENTER	When the ENTER key is specified, no keyword is written in the ENTER statement.

PRNAME	KEYWORD	LENGTH	OPTION(S)	DEFAULT
INPUT	FILE	8		
	LIBRARY	8		(INLIB value)
	VOLUME	6		(INVOL value)
	DEVICE		DISK, NONE	DISK
OPTIONS	LIBRARY1	8		(INLIB value)
	VOLUME1	6		(INVOL value)
	LIBRARY2	8		
	VOLUME2	6		
	MORE		YES, NO	NO
	PROGRAM		YES, NO, SYS, IPL	YES
	LIST		YES, NO	YES
	LINES	2	2-99	55
	RENT		YES, NO	YES
	XREF	5	FULL, SHORT, NONE	SHORT
	ERRLIST		YES, NO	NO
	SYMB		YES, NO	YES
	ESD		YES, NO	YES
	RLD		YES, NO	NO
	LIBMAC		YES, NO	NO
	ALOGIC		YES, NO	YES
	MLOGIC		YES, NO	NO
	MCALLS		YES, NO	NO
	WKSIZE	5		(determined by Assembler)
	BUFSIZE	3	STD, MIN	STD
	FLAG	3	0-20	00
	BUFCT	2		12
	STOP	2	1-20	05
	SYSPARM	68		
LIBRARY	LIBRARY1	8		
	VOLUME1	6		
	LIBRARY2	8		
	VOLUME2	6		
	LIBRARY3	8		
	VOLUME3	6		
	LIBRARY4	8		
	VOLUME4	6		
	LIBRARY5	8		
	VOLUME5	6		
	LIBRARY6	8		
	VOLUME6	6		
	LIBRARY7	8		
	VOLUME7	6		
	LIBRARY8	8		
	VOLUME8	6		

PRNAME	KEYWORD	LENGTH	OPTION(S)	DEFAULT
SPECIAL	CHARSET	6	NATIVE, UPPER, EBCDIC	NATIVE
	TEXTOUT		YES, NO	NO
	TRANSLID	2		AS
OUTPUT	FILE	8		
	LIBRARY	8		(OUTLIB value)
	VOLUME	6		(OUTVOL value)
	RECORDS	7		(determined by Assembler)
	RETAIN	3	0-999	0
	RELEASE		YES, NO	YES
	FILECLAS	1	A-Z, #, 0, @, $	(user default)
	DEVICE		DISK	DISK
ERRORS*	(Refer to PRINT under DEFAULT GETPARMS.)			
PRINT*				
WORK1*				
WORK2*	(Refer to WORK under DEFAULT GETPARMS.)			
WORK3*				
DEBUG	(Same as OUTPUT.)			

* Default GETPARM

PRNAME	KEYWORD	LENGTH	OPTION(S)	DEFAULT
OPTIONS	SOURCE		YES, NO	YES
	PMAP		YES, NO	NO
	XREF		YES, NO	NO
	LOAD		YES, NO	YES
	SUBCHK		YES, NO	NO
	DFLOAT		YES, NO	NO
	ERRLIST		YES, NO	YES
	SYMB		YES, NO	YES
	FLAG	2	00-20	00
	STOP	2	01-20	05
	LINES	2	05-99	55
INPUT	FILE	8		
	LIBRARY	8		(INLIB value)
	VOLUME	6		(INVOL value)
	DEVICE		DISK, NONE	DISK
OUTPUT	FILE	8		
	LIBRARY	8		(OUTLIB value)
	VOLUME	6		(OUTVOL value)
	RECORDS	7		(determined by compiler)
	RETAIN	3	0-999	0
	RELEASE		YES, NO	YES
	FILECLAS	1	A-Z, #, 0, @, $	(user default)
	DEVICE		DISK	DISK
PRINT*	(Refer to PRINT under DEFAULT GETPARMS.)			
WORK*	(Refer to WORK under DEFAULT GETPARMS.)			
ERRORS*	(Refer to PRINT under DEFAULT GETPARMS.)			
XREF*	(Refer to PRINT under DEFAULT GETPARMS.)			

* Default GETPARM

PRNAME	KEYWORD	LENGTH	OPTION(S)	DEFAULT
OPTIONS	SOURCE		YES, NO	YES
	LOAD		YES, NO	YES
	PMAP		YES, NO	NO
	SEQ		YES, NO	YES
	TRUNC		YES, NO	NO
	SEPSGN		YES, NO	YES
	XREF		YES, NO	NO
	SYMB		YES, NO	YES
	DMAP		YES, NO	NO
	SUBCHK		YES, NO	NO
	BIGPGT		YES, NO	NO
	LOWER		YES, NO	NO
	FLAG	2	00-20	00
	STOP	2	01-20	05
	LINES	2	2-99	55
	SPACE	1	1, 2	1
	FIPS		NO, LOW	NO
INPUT	FILE	8		
	LIBRARY	8		(INLIB value)
	VOLUME	6		(INVOL value)
	DEVICE		DISK, NONE	DISK
OUTPUT	FILE	8		
	LIBRARY	8		(OUTLIB value)
	VOLUME	6		(OUTVOL value)
	RETAIN	3	0-999	0
	RELEASE		YES, NO	YES
	FILECLAS	1	A-Z, #, 0, @, $	(user default)
	DEVICE		DISK	DISK
	RECORDS	7		(determined by compiler)
PRINT*	(Refer to PRINT under DEFAULT GETPARMS.)			
WORKTOK*				
WORK*	(Refer to WORK under DEFAULT GETPARMS.)			
ERRORWRK*				

* Default GETPARM

PRNAME	KEYWORD	LENGTH	OPTION(S)	DEFAULT
INPUT	LANGUAGE	9	COBOL, ASSEMBLER, BASIC, FORTRAN4, PLI, PROCEDURE, RPGII	COBOL
	FILE	8		
	LIBRARY	8		(INLIB value)
	VOLUME	6		(INVOL value)
	PLIBRARY	8		(OUTLIB value)
	PVOLUME	6		(OUTVOL value)
	LLIBRARY	8		(SPOOLIB value)
	LVOLUME	6		(SPOOLVOL value)
	DLIBRARY	8		(WORKLIB value)
	DVOLUME	6		(WORKVOL value)
	SCRATCH		YES, NO	YES
OUTPUT	PF Keys*		1 = Don't create ENTER = Create	
	FILE	8		
	LIBRARY	8		(INLIB value)
	VOLUME	6		(INVOL value)
	RETAIN	3	0-999	Ø
	FILECLAS	1	A-Z, #, Ø, @, $	(user default)
	START	16		ALL
	END	16		Ø
	COMPRESS		YES, NO	YES (or same as file being replaced)
	NUMBER		YES, NO	YES (or same as file being replaced)
	ID	8 (COBOL, BASIC) 6 (RPG II) None for others		
REPLACE	PF Keys*		1 = Don't replace ENTER = Replace	
	COMPRESS		YES, NO	YES
	NUMBER		YES, NO	YES
RENUMBER	PF Keys*		1 = Don't renumber ENTER = Renumber	
	NUMBER	6		(language-dependent)
	INCR	6		(language-dependent)
	START	16		ALL
	END	16		

PRNAME	KEYWORD	LENGTH	OPTION(S)	DEFAULT
DEFAULTS	TABS	20	1-71 Procedure 7-72 BASIC, COBOL 1-72 ASSEMBLER, PLI, FORTRAN None RPG II	Language dependent
	MODE	5	UPPER, UPLOW	Language dependent
	CASE	5	ANY, EXACT	Language dependent
	COLUMN	5	1-71 Procedure 7-72 BASIC, COBOL 1-72 ASSEMBLER, PLI, FORTRAN None RPG II	Language dependent
	PF Keys*		1 = Return to SET menu ENTER = Make changes	

* Description only. No keyword is used. Refer to note on page F-1.

PRNAME	KEYWORD	LENGTH	OPTION(S)	DEFAULT
DEFAULTS (for RPG II)	FORMTYPE	1	A, C, E, F, H, I, L, O, W, b	F
	MODE		UPPER, UPLOW	UPPER
	TABS	29	6-74	
	ATABS	29	6-74	
	CTABS	29	6-74	
	ETABS	29	6-74	
	FTABS	29	6-74	
	HTABS	29	6-74	
	ITABS	29	6-74	
	LTABS	29	6-74	
	OTABS	29	6-74	
	WTABS	29	6-74	
COMPILE	(Refer to compiler Options screens. Note that defaults may differ.)			
LINK	(Refer to OPTIONS under LINKER program.)			
COUTPUT	FILE	8		
	LIBRARY	8		
	VOLUME	6		
	RETAIN	3	0-999	Ø
	FILECLAS	1	A-Z, #, Ø, $, @	(user default)
LOUTPUT	(Refer to OUTPUT under LINKER program.)			
CPRINT	(Refer to PRINT under DEFAULT GETPARMS.)			
LPRINT	(Refer to PRINT under DEFAULT GETPARMS.)			
PRINT*	(Refer to PRINT under DEFAULT GETPARMS.)			
WORK*	(Refer to WORK under DEFAULT GETPARMS.)			
CDEBUG (for Assemble only)	FILE	8		
	LIBRARY	8		
	VOLUME	6		
	RETAIN	3	0-999	Ø
	FILECLAS	1	A-Z, #, Ø, $, @	(user default)

* Default GETPARM

PRNAME	KEYWORD	LENGTH	OPTION(S)	DEFAULT
OPTIONS	LIBRARY	8		
	VOLUME	6		
	NAME	6		MAIN
	LOAD		YES, NO	YES
	LIST		YES, NO	YES
	DMAP		YES, NO	NO
	SYMB		YES, NO	YES
	PMAP		YES, NO	NO
	ERRLIST		YES, NO	YES
	LINES	2	10-99	55
	STOP**	2	01-20	05
	LINKLIB		YES, NO	YES
	RUNLIB	8		@F4LIB@
	RUNVOL	6		(user default)
	LMAP		YES, NO	NO
INPUT	FILE	8		
	LIBRARY	8		(INLIB value)
	VOLUME	6		(INVOL value)
	DEVICE		DISK, NONE	DISK
OUTPUT	FILE	8		
	LIBRARY	8		(OUTLIB value)
	VOLUME	6		(OUTVOL value)
	RECORDS	7		(determined by compiler)
	RETAIN	3	0-999	0
	RELEASE		YES, NO	YES
	FILECLAS	1	A-Z, #, @, $, @	(user default)
	DEVICE		DISK	DISK
PRINT*	(Refer to PRINT under DEFAULT GETPARMS.)			
ERRORS*	(Refer to PRINT under DEFAULT GETPARMS.)			
LINK*	(Refer to OPTIONS* under LINKER.) (Footnote ** also applies.)			

* Default GETPARM
** Available with compilation from the EDITOR

PRNAME	KEYWORD	LENGTH	OPTION(S)	DEFAULT
OPTIONS	PF Keys*		16=End LINKER processing	
	LIBRARY	8		(INLIB value)
	VOLUME	6		(INVOL value)
	MORE		YES, NO	NO
	SYMB		YES, NO	YES
	MAP		YES, NO	YES
	XREF		YES, NO	NO
	EXSEC		YES, NO	NO
	LINES	2		55
LIBRARY**	PF Keys*		1=Return to Options screen	
	LIBRARY1	8		
	VOLUME1	6		
	LIBRARY2	8		
	VOLUME2	6		
	LIBRARY3	8		
	VOLUME3	6		
	LIBRARY4	8		
	VOLUME4	6		
	LIBRARY5	8		
	VOLUME5	6		
	LIBRARY6	8		
	VOLUME6	6		
	LIBRARY7	8		
	VOLUME7	6		
	LIBRARY8	8		
	VOLUME8	6		
INPUT	FILE	8		
	LIBRARY	8		(INLIB value)
	VOLUME	6		(INVOL value)
OUTPUT	FILE	8		
	LIBRARY	8		(OUTLIB value)
	VOLUME	6		(OUTVOL value)
	RETAIN	3	0-999	0
	FILECLAS	1	A-Z, #, 0, $, @	(user default)
	ENTRY	8		
	REPLACE		YES, NO	NO

* Description only. No keyword is used. Refer to note on page F-1.
** Activated if MORE=YES specified on Options screen.

PRNAME	KEYWORD	LENGTH	OPTION(S)	DEFAULT
PRNTFILE	PRINT		YES, NO	YES
	(Screen does not appear when LINKER is run interactively.)			
PRINT*	(Refer to PRINT under DEFAULT GETPARMS.)			
WORK*	(Refer to WORK under DEFAULT GETPARMS.)			
OPTIONS**	LINK		YES, NO	NO
	MAP		YES, NO	NO
	XREF		YES, NO	NO
	SYMB		YES, NO	YES
	LINES	2	20-99	55
	LIBRARY	8		(INLIB value)
	VOLUME	6		(INVOL value)
	MORE		YES, NO	NO
	FILES		YES, NO	NO

* Default GETPARM
** Available when LINKER is run from the EDITOR

PRNAME	KEYWORD	LENGTH	OPTION(S)	DEFAULT
OPTIONS	SOURCE		YES, NO	YES
	DMAP		YES, NO	YES
	PMAP		YES, NO	NO
	SYMB		YES, NO	NO
	FLAG	2	00-99	00
	STOP	2	01-99	05
	OPT		YES, NO, 1-4	NO
	CHECK		YES, NO, 1-4	NO
	SFLBIN		YES, NO	*
	FLDEC		YES, NO	*
	PREP	8	blank=none	
	LINKLIB	8	blank=none	@PLIRTM@
	LINKVOL	6		(system volume)
INPUT	FILE	8		
	LIBRARY	8		(INLIB value)
	VOLUME	6		(INVOL value)
	DEVICE	8	DISK, NONE	DISK
	FILESEQ (for tape only)	4		1
LIBRARY	LIBRARY1	8	blank=none	
	LIBRARY2	8		
	LIBRARY3	8		
	LIBRARY4	8		
	LIBRARY5	8		
	LIBRARY6	8		
	LIBRARY7	8		
	LIBRARY8	8		
	VOLUME1	6		
	VOLUME2	6		
	VOLUME3	6		
	VOLUME4	6		
	VOLUME5	6		
	VOLUME6	6		
	VOLUME7	6		
	VOLUME8	6		

* Default depends on the features supported by the Wang computer compiling
the program

PRNAME	KEYWORD	LENGTH	OPTION(S)	DEFAULT
ERRORS	FILE	8		
	LIBRARY	8		*
	VOLUME	6		(SPOOLVOL value)
	RECORDS	8		500
	FILECLAS	1		(user default)
	DEVICE	8	DISK, TAPE, PRINTER, WS	DISK
	FILESEQ (for tape only)	4		1
PRINT	FILE	8		
	LIBRARY	8		(SPOOLIB value)
	VOLUME	6		(SPOOLVOL value)
	RECORDS	8		
	FILECLAS	1		(user default)
	DEVICE	8	DISK, TAPE, PRINTER, WS	DISK
	FILESEQ (for tape only)	4		1
OUTPUT	FILE	8		
	LIBRARY	8		(OUTLIB value)
	VOLUME	6		(OUTVOL value)
	RETAIN	3	0-999	0
	FILECLAS	1	A-Z, #, @, $, @	(user default)

* Library name is xxxERR, where xxx is the user's ID.

PRNAME	KEYWORD	LENGTH	OPTION(S)	DEFAULT
OPTIONS	LIST		YES, NO	YES
	LOAD		YES, NO	YES
	DEBUG		YES, NO	NO
	SYMB		YES, NO	YES
	XREF		YES, NO	NO
	NATIVE		YES, NO	YES
	LINES	2	5-99	55
	LENGTH	2	1-99	66
INPUT	FILE	8		
	LIBRARY	8		(INLIB value)
	VOLUME	6		(INVOL value)
	DEVICE		DISK, NONE	DISK
OUTPUT	FILE	8		
	LIBRARY	8		(OUTLIB value)
	VOLUME	6		(OUTVOL value)
	RECORDS	7		(determined by compiler)
	RETAIN	3	0-999	0
	RELEASE		YES, NO	YES
	DEVICE		DISK	DISK
	FILECLAS	1	A-Z, #, @, $, @	(user default)
PRINT*	(Refer to PRINT under DEFAULT GETPARMS.)			

* Default GETPARM

DEFAULT GETPARMS

A default GETPARM is not normally displayed when a program is run because the needed information is already available. The following is a list of default GETPARMs for the user who wishes to change these values.

PRNAME	KEYWORD	LENGTH	OPTION(S)	DEFAULT
PRINT	FILE	8		
	LIBRARY	8		(SPOOLIB value)
	VOLUME	6		(SPOOLVOL value)
	RECORDS	7		
	RETAIN	3	0-999	Ø
	RELEASE		YES, NO	YES
	FILECLAS	1	A-Z, #, Ø, $, @	(user default)
	PRTCLASS	1	A-Z	(user default)
	FORM#	3	0-254	(user default)
	COPIES	5	1-32767	1
	DEVICE		DISK, PRINTER	DISK (unless PRNTMODE=Online) PRINTER (if PRNTMODE=Online)
WORK	FILE	8		
	LIBRARY	8		
	VOLUME	6		
	RETAIN		#	(temporary files)
	RELEASE		YES, NO	YES
	FILECLAS	1	A-Z, #, Ø, $, @	(user default)
	RECORDS	7		(determined by compiler)

GLOSSARY

Assembler	A system program that translates an Assembly language program into machine code.
Base Address	A starting address in memory.
Code Section	The section in the program file that contains the linked object code for the program.
Compiler	A system program that translates a source program into machine code, which the computer can execute directly.
Entry Point	A named location in a program module referenced outside the module. Also, the location in a main program at which execution begins.
External Reference	A reference within one program module to an entry point outside that module. An external reference is a reference to a subprogram.
Input File	A program file that is being linked. Input files include the main program and the subprograms.
Interpreter	A system program that processes each line of a Procedure language file individually. The Procedure Interpreter does not create a program file from the procedure file.
Main Program	The program unit that begins execution of the linked program.
Object Program	The file created by a compiler or the Assembler that can be executed or linked. On the Wang VS, this file is called the program file.
Offset	A value which, when added to a base address, indicates another address.
Output File	The executable linked program. The output file consists of all linked object program modules.

Program Control Word (PCW)	A doubleword in memory that contains the address of the next instruction that will be executed and various status bits. The PCW is described more fully in the VS Principles of Operation.
Program File	The compiled or assembled source program. This file can be executed or linked.
Program Module	A separate program unit, in either source or object form. The main program and subprograms are program modules.
Screen	That portion of the work file that fits on the workstation screen.
Source Line	An individual line of program code written in a programming language.
Source Listing	An output file created by each compiler and the Assembler that consists of the source program, errors detected during compilation or assembly, and compilation or Assembler options.
Source Program	A collection of source lines.
Static Section	The section in the program file that contains the data that the program will use.
Subprogram	A separate program unit that is called by a main program or another subprogram.
Symbolic Block	The block in the object file that contains symbolic debug information.
Trap	A location within a program module at which execution is directed to be interrupted.
Verb	Identifies the action part of a source statement (e.g., MOVE, LET).

INDEX

R

S

T

U

W

X

WANG

Customer Comment Form

Publication Number 800-1307-03

Title VS PROGRAM DEVELOPMENT TOOLS REFERENCE

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.

Printed in U.S.A. 14-3140 10-82-5C

WANG

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 16 LOWELL, MA

POSTAGE WILL BE PAID BY ADDRESSEE

WANG LABORATORIES, INC.
CHARLES T. PEERS, JR., MAIL STOP 1226A
ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851

Fold

Cut along dotted line.

WANG

ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851
TEL. (617) 459-5000
TWX 710-343-6769, TELEX 94-7421

Printed in U.S.A.
800-1307-03
10-83