Prime Computer, Inc.

Source Level Debugger

Rev. 19.2

Debugger Rel. 1.0-19.1

DOC8916-1XA

SOURCE LEVEL DEBUGGER PROGRAMMER'S COMPANION

Revision 19.2
and Debugger Release 1.0–19.1
DOC8916–1XA

This document reflects the software as of Master Disk Revision 19.2 and the Debugger's Independent Product Release (IPR) 1.0–19.1.

by

David A. Kaye

assisted by

Marion Shepp

Prime Computer, Inc.
Prime Park
Natick, Massachusetts 01760

The Programmer's Companion is a series of pocket size, quick reference guides to Prime software products

Published by Prime Computer Inc
Technical Publications Department
500 Old Connecticut Path
Framingham, Massachusetts 01701

The Programmer's Companion and PRIMOS are registered trademarks of Prime Computer Inc

The information contained in this document is subject to change without notice and should not be construed as a commitment by Prime Computer Prime Computer, Inc assumes no responsibility for errors that may appear in this document

Note

For complete information on the Source Level Debugger, see *Source Level Debugger User's Guide*

CREDITS

Editing	Pamela I Pierson
Technical Support	Paul Cioto
	Larry Epstein
	Debra Minard
Design	Susan Windheim
Production	Leo Maldonado
Typesetter	American Stratford–Graphic Services, Inc
Printer	Winthrop Printing Company

TABLE OF CONTENTS

Printing History
First Printing, January 1985

DOCUMENTATION CONVENTIONS

Abbreviations: Indicated by rust-colored letters.

-NOCOMPATIBILITY

Uppercase: Identifies command words, compiler options, and other words that must be entered literally. Enter them in either uppercase or lowercase.

RESTART

Lowercase: Identifies arguments. Substitute an appropriate numerical or text value.

LOADSTATE **filename**

Square Brackets [ ]. Indicate an optional keyword or argument.

-LISTING [**pathname** | YES | NO]

Braces { }: Indicate a choice of arguments and/or key words. At least one must be selected.

ETRACE { ON | ARGSS | OFF }

Ellipsis . . . : Indicates that the preceding parameter may be repeated.

DBG program-name [option-1 [option-2 . . .]]

Angle Brackets <>: Used literally to separate elements of a pathname

<FOREST>BEECH>LEAF

Parentheses ( ): Must be entered exactly as shown

[(argument-list)]

Hyphen –: Identifies a PRIMOS command line option Must be entered literally

–CHECKOUT

GLOSSARY OF PRIME TERMS

64V Mode: See V Mode

Master File Directory (MFD)
A special directory that contains the names of all User File Directories (UFDs) on a particular disk or partition. In PRIMOS, there is one MFD for each logical disk. *See* User File Directory

PRIMOS
Prime Computer's operating system

Pathname
A multi-part name that specifies a particular PRIMOS file system object. A full pathname consists of the names of a disk volume, a UFD, a chain of subdirectories, and a target file system object

SEG Utility
SEG is the utility used to load and execute V-mode programs

Source File
A file containing programming language statements in the format required by the appropriate compiler or assembler

User File Directory (UFD)
A directory listed in the MFD of a particular disk volume or partition. *See* Master File Directory

V Mode
The addressing mode used for multi-segmented programs under PRIMOS.

SUMMARY OF DEBUGGER FEATURES

This section summarizes the debugging command features that help you use the Debugger to solve problems in your program execution.

PROGRAM CONTROL

BREAKPOINT
Suspends program execution to examine program data strategically.

CALL
Calls a program block from Debugger command level.

CLEAR
Deletes a breakpoint or a tracepoint.

CLEARALL
Deletes all breakpoints and tracepoints in either the debugging environment or in a specific program block.

CONTINUE
Continues program execution following a breakpoint, single-step operation, or an error condition.

GOTO
Moves the execution environment pointer to a statement in an active program block.

IF
Executes an action list conditionally, depending on the result of an expression.

IN
Continues execution until the current program block calls another program block

LIST
Displays the attributes of one breakpoint or tracepoint.

LISTALL
Displays the attributes of all the breakpoints or tracepoints you have set.

MAIN
Displays the current main program or designates a program block to be recognized as main program.

OUT
Continues execution until the current program block returns.

RESTART
Starts or restarts your program's execution.

STEP
Executes a given number of statements at a time stepping across any called program block.

STEPIN
Executes a given number of statements at a time stepping into any called program block.

UNWIND
Erases the call/return stack. (The execution environment pointer becomes undefined.)

DATA MANIPULATION

:
Evaluates a variable or expression

ARGUMENTS
Displays the values of all arguments passed to a program block.

LANGUAGE
Displays or changes the name of the language the Debugger uses to evaluate expressions.

LET
Assigns a new value to any variable defined by the program.

PMODE
Sets the print mode used to evaluate a variable.

TYPE
Examines the data type and other attributes of a variable or expression.

TRACING

ETRACE
Displays a message each time the execution calls or returns from a program block (entry/exit tracing)

STRACE
Displays a trace message before every program statement or labelled statement is executed (statement tracing).

TRACEBACK
Looks at the contents of the call/return stack, which is a list of program block calls currently active in program execution.

TRACEPOINT
Displays a message each time the Debugger encounters a statement, label, or entry/exit of a program block.

UNWATCH
Removes one or more variables from the watch list.

VTRACE
Temporarily limits value tracing to the entry or exit of a program block or turns off value tracing without disturbing the watch list.

WATCH
Displays a message whenever the value of one or more variables changes during program execution (value tracing) by adding one or more variables to the watch list.

WATCHLIST
Displays the names of variables currently in the watch list.

DEBUGGER CONTROL

ACTIONLIST
Displays a breakpoint action list or macro command list immediately before it is executed.

ENVIRONMENT
Changes or verifies the evaluation environment

ENVLIST
Displays the current evaluation environment and the contents of the evaluation environment stack

PSYMBOL
Displays a list of Debugger special symbols (ERASE, KILL, ESCAPE, SEPARATOR, WILD, and BLANKS) and their current character values

RESUBMIT
Invokes the Debugger's command line editor to edit the most recent command

SYMBOL
Changes the character value of a special symbol

INFORMATION REQUEST

INFO
Displays attributes of a program block or statement

SEGMENTS
Displays a list of the segments in memory currently in use

STATUS
Displays information on the state of your current debugging environment

WHERE
Verifies the location of the execution environment pointer or finds the program location that corresponds to a given memory address

MISCELLANEOUS

!
Enters and executes certain PRIMOS commands from Debugger command level

*
Executes the current command line a specific number of times or indefinitely until the Debugger finds an error (or you press the break key or CONTROL-P)

AGAIN
Repeats the command line just executed

CMDLINE
Enters your program's command line arguments from Debugger command level

HELP
Gets help with command syntax definitions

LOADSTATE
Loads the contents of SAVESTATE file into a debugging session

MACRO
Creates a macro to take the place of one or more Debugger commands

MACROLIST
Displays one or all of your currently defined macros and their command lists

PAUSE
Temporarily suspends debugging session and returns to PRIMOS command level

QUIT
Returns to PRIMOS command level

SAVESTATE
Saves all of your breakpoints, tracepoints, and macros in PRIMOS files for future use

SOURCE
Looks at, but does not change, source files using Editor subcommands

VPSD
Invokes the 64V mode Prime Symbolic Debugger (VPSD), (machine level debugger) from Debugger command level

COMMAND LINE EDITOR

Use the Debugger's command line editor to modify the most recent command line as well as any breakpoint action list or macro command list. Invoke the editor with the RESUBMIT, BREAKPOINT -EDIT, or MACRO -EDIT command. Table 1 summarizes the command line editor subcommands, which are entered at the editor's colon ( ) prompt.

TABLE 1
COMMAND LINE EDITOR SUBCOMMANDS

Subcommand	*Function*
A	Appends the text that follows A to the end of the line
D	Deletes the characters under which the D is positioned
F	Makes the character under which the F is positioned the first character of the command line
I	Inserts the text that follows I into the line, starting after the character under which the I is positioned
L	Makes the character under which the L is positioned the last character of the command line
O	Overlays the characters that follow O onto the line, starting at the character under which the O appears
Q	Aborts the editing session and returns to Debugger command level, but does not replace original command line
carriage return	Finishes the editing session and replaces original command line
space	Moves the cursor to the right one position

INVOKING THE DEBUGGER

The Source Level Debugger is Prime's interactive debugging tool for its seven supported high-level languages. In order to use the Debugger, you must compile and load your program successfully. Enter the Debugger during the execution phase. The following steps show the procedure for invoking the Debugger.

1. COMPILING WITH THE -DEBUG OPTION

You must tell the compiler that you intend to use the Debugger by entering the -DEBUG compile-time option on the command line.

compile-command program-name -DEBUG

The **compile-command** is one of the language-specific commands listed in Table 2. **program-name** is the name of the program.

TABLE 2
COMPILE COMMANDS, SOURCE FILE SUFFIXES, AND LIBRARIES

Language	*Compile Command*	*Compiler Source File Suffix*	*Language Library*
FORTRAN IV	FTN	.FTN	None
FORTRAN 77	F77	.F77	None
Pascal	PASCAL	.PASCAL	PASLIB
PL1 Subset G	PL1G	.PL1G	PL1GLB
COBOL 74	CBL	.CBL	CBLLIB
RPG II V-mode	VRPG	RPG	VRPGLB
C	CC	.CC	CCLIB

Compiling FORTRAN IV Programs: When debugging in FORTRAN IV, you must compile your program with the **-64V** option along with the -DEBUG option:

FTN program-name -64V -DEBUG

2. LOADING WITH THE SEG LOADER

The next step is to load your program as follows.

```
OK, SEG -LOAD
[SEG rev 19.2]
$ LOAD program-name
$ LIBRARY language-library
$ LIBRARY
LOAD COMPLETE
$ QUIT
OK,
```

The **language-library** is the appropriate language library listed in the preceding table

3. ENTERING DEBUGGER WITH THE DBG COMMAND

Enter the Debugger by issuing the DBG command

DBG program-name [option-1 [option-2 . . .]]

The **program-name** here is an executable (SEG) file **option-1** and **option-2 . . .** are optional command line parameters DBG command line options are described below

-LOADSTATE pathname
-LOADSTATE allows you to restore the contents of a SAVESTATE file—your saved breakpoints, tracepoints, and macros—upon invoking the Debugger The **pathname** is the pathname of the file you want to restore

-VERIFY_SYMBOLS
Checks all external symbol declarations for consistency in all program blocks within the executable file The Debugger displays a warning message during initialization if it finds inconsistent external symbol declarations

-NO_VERIFY_SYMBOLS
Suppresses external symbol checking, speeding up initialization (-VERIFY_SYMBOLS is the default)

-VERIFY_PROC
Checks the procedure text so you can set statement breakpoints The Debugger gives a warning message if it finds a statement for which the procedure text is unsuitable for placing a breakpoint

-NO_VERIFY_PROC

Specifies that the procedure text is not to be inspected for improper format regarding placement of break points (-VERIFY_PROC is the default)

-COMINPUT

Specifies that the Debugger accept input from a command input file or CPL program

-NO_COMINPUT

Specifies that the Debugger accept input from only the terminal, not from a command input file (-NO_COMINPUT is the default)

-FULL_INIT

Tells the Debugger to read and process the entire symbol table from the specified executable file before entering command mode Normally the debugger reads from the symbol table only when it needs the information Use this option to obtain a complete external symbol mismatch summary at initialization time -FULL_INIT approximately triples initialization time

-QUICK_INIT

Tells the Debugger to load, at initialization time, only the information it needs to identify each program block The debugger loads the rest of the symbol table as required during the debugging session (-QUICK_INIT is the default)

COMMAND FORMAT CONVENTIONS

GENERAL FORMAT

Enter Debugger commands at the prompt character > The general format is

```
>COMMAND-NAME [argument-1
               argument-2 . . . ]
```

argument-1 and **argument-2 . . .** are one or more command arguments that may be

- Options that appear in uppercase in the command format and that you enter literally as shown
- Variables, expressions, symbol names, activation numbers, or statement identifiers that appear in lowercase in the command format You substitute a suitable value

ARGUMENT FORMATS

Variables: The rules for identifying variables within the Debugger are the same as the rules of host language The syntax is expanded so you can reference any variable in the debugging environment There are three possible formats

- variable name

  Specifies a variable in the current program block **variable name** can possibly be qualified and/or subscripted according to the rules of the host language

- program block name\variable name

  Specifies a variable in the most recent (or only) activation of a named program block

- program block name\activation number\variable name

  Specifies a variable in a named program block and activation

Statements· There are six possible ways to identify a statement, three use source file line numbers and three use statement labels

Statement labels are label numbers or label constants in any of the seven supported languages (Precede labels that begin with a digit with a dollar sign ($) to distinguish them from source line numbers)

You may specify the program block name with a backslash (\) directly before the information in any of the six formats

program-block name

If you do not supply **program-block-name,** the Debugger looks for the statement in the current program block Below are the six formats

- source line

  source line is a source file line number When multiple statements are on a single line, this format specifies the leftmost statement at the source file line number

- source-linestatement-offset

  statement-offset is the number of statements to count from the first statement on a multistatement line such as an IF statement whose THEN clauses are on a different line of text (The first statement has an offset of 0)

- source line(insert line)
  source line(insert line+statement offset)

  insert-line is a line number in a $INSERT or %INCLUDE file This format is for $INSERT or %INCLUDE files that contain executable statements

- statement-label

  Identifies a statement by label Precede references to FORTRAN, Pascal, and COBOL 74 statement numbers with a dollar sign to distinguish numeric labels from source line numbers See Table 3

TABLE 3
CHARACTERISTICS OF STATEMENT LABELS

Language	*Type*	*Characteristics*
FORTRAN IV	Numeric	Referenced with preceding dollar sign ($)
FORTRAN 77	Numeric	Referenced with preceding dollar sign ($)
Pascal	Numeric	Declared in LABEL declaration part of program, referenced with preceding $
PL/I–G	Alphanumeric	First character must be alphabetic, not referenced with $

TABLE 3 (CONTINUED)

Language	*Type*	*Characteristics*
COBOL 74	Alphanumeric	COBOL paragraph name or section name, referenced with preceding $ only when first character is numeric.
RPG II V-mode	Alphanumeric	RPG tag. First character must be alphabetic; not referenced with $
C	Alphanumeric	First character must be alphabetic, not referenced with $.

- statement-label+line-offset

  line-offset is the number of source lines following the line containing **statement-label.** The referenced statement is always the leftmost statement on the line

- statement-label+line-offset+statement-offset

  statement-offset is the number of statements to count from the first statement on a multistatement line (The first statement has an offset of 0)

ALPHABETICAL LIST OF DEBUGGER COMMANDS

► . [**language-name[,print-mode]**] **expression**
[**[print-mode]**]

The . (evaluation) command evaluates a variable or an expression.

The optional **language-name** is the language of evaluation, you are evaluating with the syntax and semantic rules of this language If not specified, the language of evaluation is the host language.

The **print-mode** specifies the format for the result. The print-mode can be ASCII, BIT, DECIMAL, FLOAT, HEX, or OCTAL. (See PMODE command)

The **expression** is the variable or expression you want to evaluate. The expression can be a variable or a more complex expression. Where applicable, you may evaluate.

Simple variables	Arrays
Array elements	Array cross sections
Structures (records)	Structure members (fields)
Pointer-referenced data	All legal expressions involving any of the above

Evaluating Arrays: To reference a portion of an array, specify a star extent or a bound pair.

Star extent To display or operate on the full range of a dimension, substitute a star (*) for the corresponding subscript. For example:

```
> : array-name (*, num)
```

num is the index for the second dimension.

Bound pair To display or operate on a limited range of a dimension, specify bounds in the form:

lower-bound . . . upper-bound

For example:

array-name . . . (3 . . . 7)

Lower-bound and **upper-bound** are any valid expressions that reduce to integer values. Enter the ellipsis (. . .) literally.

Functions: The Debugger supports standard FORTRAN, PL1G, Pascal, and C language functions. You can use the evaluation command with these functions to evaluate expressions. Table 4 on the next page lists supported language functions.

▶ ! primos-command-line

The ! command executes internal PRIMOS commands from Debugger command level.

The **primos-command-line** contains one or more internal PRIMOS commands that you want to execute from the Debugger. (External commands interfere with the memory image of the Debugger or your program.)

Internal and external commands are summarized in the *PRIMOS Commands Reference Guide.*

TABLE 4
PLUS PASCAL, FORTRAN AND C
SUPPORTED FUNCTIONS

ABS	COMPLEX	EXP	MAXI	QTANH
ACOS	CONJG	FAULT	MIN	QUAD
ADD	COPY	FIXED	MIN0	RANK
ADDR	COS	FLOAT	MINI	REAL
ADDREI	COSD	FLOOR	MOD	RED
AFTER	COSH	HBOUND	MULTIPLY	REVERSE
AIMAG	CSIN	HIGH	NINT	RINC
AINT	CSQRT	IABS	NOT	RND
ALOG	DABS	ICHAR	NULL	ROUND
ALOG10	DACOS	IDIM	ODD	RS
AMAX0	DASIN	IDINT	OFFSET	RT
AMAX1	DATAN	IDNINT	ONCODE	SEARCH
AMIN0	DATAN	IFIX	OR	SEGNO
AMINI	DATA	IMAG	ORD	SHFT
AMOD	DBLE	INDEX	POINTER	SIGN
AND	DBLEQ	INSERT	PRED	SIN
ANINT	DCMPLX	INT	PTR	SIND
ARCTAN	DCOS	INTR	QABS	SINH
ASIN	DCOSH	INTS	QACOS	SIZEOF
ATAN	DDIM	IQINT	QASIN	SNGL
ATAN2	DEC	IQNINT	QATAN	SQR
ATAND	DECAT	IRND	QATAN2	SQRT
ATANH	DECIMAL	ISIGN	QCOS	STACKBASE
BASEPTR	DFLOAT	LBOUND	QCOSH	STACKPTR
BASLREL	DINT	LEN	QDIM	STR
BEFORE	DIM	LENGTH	QEXP	STRING
BIN	DIMENSION	LGE	QEXT	SUBSTR
BINARY	DINT	LGT	QEXTD	SUBTRACT
BIT	DIVIDE	LINKPTR	QINT	SUCC
BOOL	DLOG	LLE	QLOG	TAN
BYTE	DLOG10	LLT	QLOG10	TAND
CABS	DMAX1	LN	QMAX1	TANH
CCOS	DMIN1	LOG	QMIN1	TIME
CEIL	DMOD	LOG10	QMIN1	TRANSLATE
CEXP	DNINT	LOGTO	QMOD	TRIM
CHAR	DPROD	LOG	QNINT	TRUNC
CHARACTER	DSIGN	LOWW	QPROD	UNSPEC
CHR	DSIN	LS	QSIGN	UNSTR
CLOG	DSINH	LT	QSIN	UTILITY
CMPLX	DSQRT	LTRIM	QSINH	VOK
CMPX	DTAN	MAX	QSQRT	
COLLATE	DTANH	MAX0	QTAN	

► *** [value]**

The * command executes the current command line a specific number of times or indefinitely

The **value** is the optional number of times you want to repeat the command line. If you do not specify a value, the command line repeats until the Debugger finds an error or until you press the break key or CONTROL-P.

The * must be the last command on the command line and separated from the preceding commands by a semicolon (command separator).

▶ ACTIONLIST [SUPPRESS | PRINT]

ACTIONLIST PRINT displays the commands in a breakpoint action list or in a macro command list. ACTIONLIST SUPPRESS deactivates ACTIONLIST PRINT so no lists are displayed. The default is SUPPRESS.

▶ AGAIN

AGAIN repeats the most recently executed Debugger command line. Enter the AGAIN command by itself after the > prompt.

▶ ARGUMENTS [program-block name[\activation number]]

ARGUMENTS displays the values of the arguments passed to the program block defined by the evaluation environment pointer.

program-block-name is the name of the program block whose arguments you want to display. **activation-number** is a particular activation of a specified program block.

► **BREAKPOINT [breakpoint-identifier] [action-list] [options]**
[-AFTER value] [-BEFORE value]
[-EVERY value] [-COUNT value] [-EDIT]

[-IGNORE]
[-NIGNORE]

BREAKPOINT suspends the execution of your program. The **breakpoint-identifier** identifies where you want to suspend execution, which can be an executable statement, statement label, or an entry to or exit from a program block. If you don't specify a breakpoint-identifier, the Debugger uses the value of the execution environment pointer. See the list of options and functions that follows.

The **action-list** specifies one or more Debugger commands to be executed at the breakpoint. To create an action list, enclose the list of Debugger commands within a pair of square brackets ([ ]) and separate the commands with semicolons

-AFTER
Causes the breakpoint to occur only when the value of the breakpoint counter exceeds the value of the specified **value** following -AFTER

-BEFORE
Causes the breakpoint to occur only when the value of the breakpoint counter is less than the **value** following -BEFORE

-EVERY
Causes the breakpoint to occur every n iterations through the breakpoint location, where **n** is the value following -EVERY

-COUNT
Can be used to set the breakpoint counter

-IGNORE
Sets the ignore flag, suppressing the breakpoint

-NIGNORE
Deactivates the ignore flag

-EDIT
Invokes the Debugger's command line editor so you can modify a breakpoint action list

Identify entry/exit breakpoints by one of the following three formats:

- BRK program-block-name\\breakpoint-type
- BRK \breakpoint-type
- BRK program-block-name\

The **breakpoint-type** can be either ENTRY or EXIT. The **program-block-name** is the name of the called program block where you want to break.

Breakpoints at statements or statement labels suspend execution immediately before the statement or labeled statement.

Breakpoints at the entry of a program block suspend execution inside the called program block immediately after argument transfer. Exit breakpoints suspend execution outside the program block after the block has returned.

▶ CALL variable [(argument-list)]

CALL allows you to call a program block from Debugger command level.

The **variable** is the name of the program block you want to call. The **argument-list** is a list of expressions, or "parameters," that are supplied, or "passed," to the program block according to the rules of the host language. In the argument-list, expressions are separated by commas.

When you give a CALL command, the Debugger evaluates each argument and calls the block, supplying the values as arguments. To call a block within another external program block, specify the block name or external block name followed by a \ (backslash) before the variable.

▶ CLEAR [breakpoint-identifier]

CLEAR deletes one breakpoint or one tracepoint.

The **breakpoint-identifier** must be any valid breakpoint or tracepoint identifier, such as a source line number or statement label. Used by itself, with no breakpoint identifier, CLEAR deletes the breakpoint or tracepoint specified by the execution environment pointer.

▶ CLEARALL [program-block-name [-DESCEND]] [-BREAKPOINTS | -TRACEPOINTS]

CLEARALL deletes all breakpoints and tracepoints in either the debugging environment or in a specific program block.

The **program-block-name** is the name of the program block containing the breakpoints and/or tracepoints that you want to delete.

-DESCEND
Deletes all breakpoints and tracepoints in a specified program block and in all the nested program blocks or "descendants" contained in the specified block.

-BREAKPOINTS
Deletes only breakpoints.

-TRACEPOINTS
Deletes only tracepoints
Used without any arguments, CLEARALL deletes all breakpoints and tracepoints in the debugging environment

▶ CMDLINE

CMDLINE allows you to enter the command line arguments that your program expects from Debugger command level. After typing the command, you get the prompt

```
Enter command line:
```

▶ CONTINUE

CONTINUE resumes program execution following a breakpoint, a single step operation, or an error condition. Program execution resumes at the location specified by the execution environment pointer

▶ ENVIRONMENT

[**program-block-name[\activation-number]**]
[**-POP**]

ENVIRONMENT changes the evaluation environment, which is the program block the Debugger considers current

The **program-block-name** is the name of the program block that you want as the new evaluation environment. The **activation-number** specifies a particular activation of program-block-name. The -POP option removes or "pops" an environment

from the evaluation environment stack. Used by itself, with no argument, ENVIRONMENT displays the name of the current evaluation environment.

► ENVLIST

ENVLIST displays the current evaluation environment and the contents of the evaluation environment stack.

► ETRACE { ON | ARGS | OFF }

ETRACE displays a trace message each time a program block is called or returned. This is known as entry tracing.

ON
Displays a trace message when each program block is called and returned.

ARGS
Displays trace messages at the entry and exits to called program blocks and displays the values of arguments passed to each called block at each entry (but not each exit).

OFF
Turns off entry tracing.

► GOTO [program-block-name\[activation-number\]]statement-identifier

GOTO moves the location of the execution environment pointer to another statement in your program.

The **program-block-name** is the name of the active program block containing the statement to which you are transferring control. The **statement-identifier** is the statement to which you are transferring control. It can be a source line number, statement label, or any other valid identifier. The **activation-number** specifies that control is transferred to a statement in a particular activation of a program block.

After a GO TO, the evaluation environment pointer is set to the new program block. If the specified program block is written in another language, the debugger sets the language of evaluation to that language.

► HELP [-LIST | -SYM_LIST | **command-name** | **syntax-symbol**]

HELP displays information about Debugger commands and features.

The **command-name** is the name of any Debugger command for which you want command line syntax information. The **syntax-symbol** is any symbol used in command syntax descriptions. The -LIST option lists all Debugger commands in alphabetical order. The -SYM_LIST option lists all Debugger syntax symbols used in Debugger command line syntax.

► IF **expression action-list [ELSE action-list]**

IF executes a breakpoint action list or any Debugger command conditionally, depending on the result of an expression.

The **expression** is any valid expression in the host language. The expression can be either true or false. If the expression is true, the first action list immediately following the expression is executed, and the ELSE clause, if there, is ignored. If the expression is false, the first action list is ignored, but the ELSE action list, if there, is executed. (*See* discussion of the action list in "Debugger Terms and Concepts.")

You can use an IF command clause within the action list of another IF command clause to form a nested action list.

▶ IN

IN continues program execution until the next program block is called and suspends execution inside that block immediately before the first executable statement. Do not use GOTO at this point, because it may prevent initialization of the program block. Issue a STEP command before using GOTO.

▶ INFO program-block-name\ statement-identifier

INFO displays information about a program block or statement.

The **program-block-name** is the name of the program block you want information about. The **statement-identifier** is the executable statement you want information about. For a statement, the Debugger displays the memory address of the first instruction.

► LANGUAGE [FORTRAN | F77 | PL1G | PASCAL | COBOL | RPG | C]

LANGUAGE changes the language of evaluation, which is the language the Debugger uses to evaluate expressions (also called host language)

Used without an argument, LANGUAGE displays the name of the current host language The default language of evaluation is the source language of the program block containing evaluation environment pointer To change the current language to another language, use the appropriate argument

► **LET variable = expression**

LET assigns a new value to any variable defined by the program

The **variable** is a variable name The **expression** is any expression permitted by the host language whose resultant value can be converted to the data type of the variable

► **LIST [breakpoint-identifier]**

LIST displays the attributes of one breakpoint or one tracepoint

The **breakpoint-identifier** is the breakpoint or tracepoint that you want to display Used without the breakpoint identifier, LIST displays the attributes for the breakpoint or tracepoint defined by the execution environment pointer

► LISTALL [**program-block-name** [-DESCEND]] [-BREAKPOINTS | -TRACEPOINTS]

LISTALL lists the attributes of all breakpoints and tracepoints

The **program-block-name** is the name of the program block that contains the breakpoints and tracepoints you want to display.

-DESCEND
Displays all breakpoints and tracepoints for a specified block and for all nested program blocks or "descendants" contained in the specified block

-BREAKPOINTS
Displays only breakpoints

-TRACEPOINTS
Displays only tracepoints

If LISTALL is used without arguments, it displays a list of all breakpoint and tracepoint attributes.

► LOADSTATE **filename**

LOADSTATE puts previously saved breakpoints, tracepoints, and macros into your debugging session. They were saved with SAVESTATE

The **filename** is the pathname of the SAVESTATE file.

► MACRO { **macro-name** { **command-list** | -DELETE | -EDIT } | -CHANGE_NAME **old-macro-name new-macro-name** | -ON | -OFF }

Creates new commands, called macros, that can be used in place of one or more Debugger commands

macro-name is the name of the macro that you want to create **command-list** is the list of one or more Debugger commands that you want your macro name to stand for

You must enclose the command list within square brackets and separate the commands with semicolons

-DELETE
Deletes a specified macro

-EDIT
Invokes the Debugger command line editor so that you can modify the macro specified by macro name

-CHANGE_NAME
Changes the name of a macro from **old-macro-name** to **new-macro name**

-OFF
Turns off the use of macros without destroying your current macros

-ON
Enables the use of macros once again

To create a macro so that you can use one or more parameters as desired, enclose a positive integer within percent signs (%) in the command list for every parameter you may want to use

► MACROLIST [macro-name]

Displays one or all of your currently defined macros and their command lists

The **macro name** is the name of a specific macro that you want to display Used by itself, with no macro name, MACROLIST displays all the macros in the macrolist and in their command lists

► **MAIN [program-block-name]**

MAIN tells the Debugger what the main program block should be. The main program is the program block that the Debugger calls when a RESTART command is entered

The **program-block-name** is the name of the program block that you want the Debugger to call when a RESTART command is entered Used by itself, with no program block name, MAIN displays the name of the main program that the Debugger currently recognizes

► **OUT**

OUT continues program execution until the current block, defined by the execution environment pointer, returns

► **PAUSE**

PAUSE temporarily suspends your debugging session and returns you to PRIMOS command level. You must enter only internal PRIMOS commands with PAUSE, not external commands

► **PMODE print-mode variable-1 [,variable-2 . . .]**

PMODE sets the print mode of a variable to a specified print mode Whenever tht variable is displayed in your debugging session, it is displayed in the specified print mode

The **print-mode** is the print mode you want to specify. It can be ASCII, BIT, DECIMAL, FLOAT, HEX, OCTAL, or DEFAULT.

variable-1 and **variable-2 . . .** are the variables whose print mode you want to set.

The next list provides the results that are printed for each print mode.

ASCII
Prints each group of 8 bits as an ASCII character

BIT
Prints each bit as a binary digit

DECIMAL
Prints each group of 16 bits as a signed single-precision decimal number

FLOAT
Prints each group of 32 bits as a single-precision floating point number

HEX
Prints each group of 4 bits as a hexadecimal digit

OCTAL
Prints each group of 16 bits as an unsigned octal number

DEFAULT
Sets the print mode to the default mode (the mode corresponding to the declared type of the variable)

▶ PSYMBOL

PSYMBOL displays a list of the names and current character values of special symbols. The Debugger recognizes six special symbols.

Erase
Erases the immediately preceding character

Kill
Ignores all characters typed so far on the line

Escape
Gives different meaning to the immediately following character

Separator
Separates commands on command lines

Wild
SOURCE command wildcard for FIND and LOCATE operations

Blanks
SOURCE command match for any number of blanks

▶ QUIT

QUIT ends the debugging session and returns you to PRIMOS command level

▶ RESTART [step-command]

RESTART starts or restarts program execution from within the Debugger

The **step-command** is an optional Debugger single-stepping command (STEP, STEPIN, IN, or OUT)

▶ RESUBMIT

RESUBMIT invokes the Debugger's command line editor so that you can modify the most recent command line entered

For a complete list of command line editor subcommands, see the section on the command line editor in "Summary of Debugger Features"

► SAVESTATE filename [-MACROS] [-BREAKPOINTS] [-TRACEPOINTS]

SAVESTATE saves your breakpoints, tracepoints, and/or macros and places them into a PRIMOS text file for future use.

The **filename** is the pathname of the PRIMOS file where you want to place your breakpoints, tracepoints, and/or macros. If you specify only the filename, the file will be placed in the directory to which you are attached.

-MACROS
Saves only your macros

-BREAKPOINTS
Saves only your breakpoints and their action lists

-TRACEPOINTS
Saves only your tracepoints

If you specify only a filename without an option, then all of your breakpoints, tracepoints, and macros are saved.

► SEGMENTS

SEGMENTS displays a list of segments in memory currently in use. The segments are classified by usage as follows:

- User procedure text, linkage text, and data
- Debugger procedure text
- Debugger linkage text, data, and symbol table
- Stack areas

► SOURCE source-command [argument]

SOURCE allows you to examine your source file while debugging

The **source-command** is any EDITOR command that you can use with SOURCE There are 14 that you can use, all of which examine, but do not modify, a file The **argument** is an EDITOR command object such as a line number or text string See the source EDITOR subcommands listed below.

*****

Repeat command line, see also Debugger REPEAT (*) Command

BOTTOM

Position pointer to bottom of file

BRIEF

Don't print target lines of FIND, LOCATE, POINT, and NEXT operations

FIND

Locate line with the specified text string beginning in a given column

LOCATE

Locate line with the specified text string

MODE

Set edit mode, the only mode implemented is NUMBER/NNUMBER

NEXT

Move line pointer forward or backward

POINT

Position to specific line

PRINT

Print one or more lines

PSYMBOL

Print character symbols, see also Debugger PSYMBOL command

SYMBOL

Set character symbol, see also Debugger SYMBOL command

TOP
Position line pointer to top of file

VERIFY
Print target lines of FIND, LOCATE, POINT, and NEXT operations

WHERE
Print current line number

There are three other special source subcommands.

EX
The EX subcommand sets the source file and EDITOR line pointer to the source line where execution resumes (the execution environment pointer), then displays that line. You cannot use this command when the execution environment pointer is at a program block exit.

NAME

[filename | -DEFAULT]

The NAME subcommand lets you look at the contents of another file from within the Debugger.

The -DEFAULT option brings you back to looking at the file corresponding to the evaluation environment.

Used with no argument, the NAME subcommand gives the current source pathname.

RENAME
filename [-BLOCK program-block name]

The RENAME subcommand resets the default source filename for a specified program.

The **filename** is the name you want for your default source file. The **program-block-name** is the name of the program block in which the default source file will be the specified filename. If you do not specify program-block name, the Debugger assumes it is the current evaluation environment. If the indicated program block is the same as the current block, the current source file is changed to **filename**.

▶ STATUS

STATUS displays information about the state of your debugging environment.

▶ STEP [value]

STEP executes one or more statements at a time and steps across calls to program blocks.

The **value** is the number of statements you want to execute before suspending execution. If no value is specified, one statement is executed by default.

▶ STEPIN [value]

STEPIN executes one or more statements at a time and steps into program blocks that are called.

The **value** is the number of statements you want to execute before suspending execution. If no value is specified, one statement is executed by default.

▶ STRACE { FULL | QUIET | OFF }

STRACE allows you to display a trace message before execution of every program statement or every labelled program statement. STRACE invokes the statement tracing feature.

FULL
: Displays a trace message before the execution of every program statement in your program

QUIET
Displays a trace message only before the execution of each labeled statement

OFF
Turns off statement tracing

▶ SYMBOL symbol-name character-value

SYMBOL changes the value of a special symbol recognized by the Debugger

The **symbol-name** is the name of the character symbol ERASE, KILL, ESCAPE, SEPARATOR, WILD, or BLANKS The **character-value** is the new character value of the symbol It may *not* be alphanumeric or identical to an existing character symbol value, and it may *not* be a space

▶ TRACEBACK [-FRAMES value [-LEAST_RECENT]] [-FROM value] [-TO value] [-REVERSE] [-DBG] [-ONUnits] [-ADDRESSES]

TRACEBACK allows you to look at the contents of the call/return stack, a list of currently active program blocks in your program execution

value is a positive non zero integer With no arguments, all frames on the stack are printed from most recent to least recent

-FRAMES
Specifies the number of frames displayed by **value** and display frames from the most recent frame to the least recent frame

-LEAST_RECENT
Displays the least recent **value** frames

-FROM
Starts the traceback from the frame number **value** that follows -FROM

-TO
Ends the traceback with the frame represented by **value**

-REVERSE
Lists the frames in reverse order from the least recent to the most recent

-DBG
Displays debugger-owned frames in expanded form along with other frames

-ONUNITS
Displays for each frame the names of all on-units and their addresses

-ADDRESSES
Displays internal address information

► **TRACEPOINT [breakpoint-identifier] [-AFTER value] [-BEFORE value] [-EVERY value] [-COUNT value] [-IGNORE | -NIGNORE]**

TRACEPOINT displays a trace message each time a statement, label, or entry/exit to a program block is encountered

The **breakpoint-identifier** is the statement, label, or entry/exit where you want to display a trace message

The -AFTER, -BEFORE, -EVERY, -COUNT, -IGNORE, and -NIGNORE options work the way they do for breakpoints (For an explanation of these options, see the discussion under the BREAKPOINT command in this section)

▶ TYPE expression

TYPE displays the data type and other attributes of a variable or expression.

expression is any expression permitted by the host language.

▶ UNWATCH { variable-1 [,variable-2 . . .] / -ALL }

UNWATCH removes one or more variables from the watch list (created during value tracing with the WATCH command).

variable-1 and **variable-2 . . .** are the variables you want to remove from the watch list. The -ALL option removes all variables from the watch list.

▶ UNWIND

UNWIND unwinds call/return stack and causes the execution environment pointer to become undefined.

▶ VPSD

VPSD invokes the 64V-mode Prime Symbolic Debugger (VPSD), which is one of Prime's machine-level debuggers.

► VTRACE { FULL | ENTRY_EXIT | OFF }

VTRACE can trace values at the entry or exit of a program block and turn off value tracing

ENTRY_EXIT
Enables value tracing on only the entries to and exits from program blocks

OFF
Suppresses value tracing without disturbing the contents of the watch list

FULL
Enables value tracing at every statement once again

► **WATCH variable-1 [,variable 2 . .]**

WATCH displays a message whenever the value of one or more variables changes during program execution This feature is known as value tracing

variable-1 and **variable-2 . . .** are the variables whose values you want to trace The variables that you trace are placed onto an internal Debugger table known as the watch list

Give a program block and activation number to watch an automatic variable at that activation only

program-block-name\activation-number\ variable-name

To watch any portion of an array or structure, use star extent or bound pair in reference (See the command)

The way variables are watched differs for each storage class

- The value of a static variable is saved when the WATCH command is given and is watched throughout the debugging session unless it is removed by UNWATCH (All COBOL variables are static.)
- Value of an automatic variable is saved upon program block entry and watched until the program block becomes inactive
- A PL/1-G based variable or Pascal dynamic variable is saved and watched according to the storage class of the locator (pointer)
- PL/1-G controlled variables cannot be watched

▶ WATCHLIST

WATCHLIST displays the names of variables currently in the watch list.

▶ WHERE [segment-number/offset]

WHERE displays the location of the execution environment pointer.

You can find the program location that corresponds to a given memory address by specifying the **segment-number** (octal), and the **offset** (octal), which is the address of the location in the segment.

Used by itself, with no argument, WHERE displays the current location of the execution environment pointer.

DEBUGGER TERMS AND CONCEPTS

Several Debugger terms and concepts are related to the Debugger functions. For more detailed information, see the *Source Level Debugger User's Guide*

- Action List

  An action list is a list of Debugger commands enclosed in square brackets and separated by semicolons For example

  [X, TYPE X, TYPE Y]

  (See the discussion under the BREAKPOINT command)
- Activation

  An activation refers to a particular execution of a program block An activation number specifies a particular activation of a program block when more than one activation can exist The activation numbers are either absolute or relative

  Absolute
  The actual number of the activation

  Relative
  The number of activations to count backwards from most recent activation Specify number with a minus sign and integer constant

- Active Program Blocks

  An active program block is a program block that has been called, but not yet returned

- Environments

  The environment identifies a program block or subroutine. The Debugger maintains two environment pointers

  Execution Environment Pointer
  Describes the location at which the Debugger resumes execution. (Defined only when program is active.)

  Evaluation Environment Pointer
  Describes the default program block block at which the Debugger looks for variables and statements. The default evaluation environment depends on how the Debugger is entered. (See *Source Level Debugger User's Guide*.)

- Language of Evaluation

  The default language that the Debugger uses at any given time is set to the source language of the program block containing the evaluation environment pointer. The language of evaluation tells the Debugger which language syntax rules to use in evaluating expressions.

- Program Blocks

  The universal language-independent term program block refers to any program unit in any of the seven supported languages. The Debugger uses the names of program blocks to identify variables and statements.

  Table 5 shows what program blocks are in the context of each of the languages and explains how the Debugger identifies the program blocks.

TABLE 5
PROGRAM BLOCKS

Language	*Program Block*	*Identification*
FORTRAN IV	Main Program	By name, if provided, in FORTRAN PROGRAM statement and by $MAIN if name not provided
	Subroutine	By name in SUBROUTINE statement
	Function	By name in FUNCTION statement
FORTRAN 77	(same as FORTRAN IV)	
PL1 Subset G	Procedure	By procedure name
	BEGIN block	By $BEGIN followed by source line number of BEGIN statement
PASCAL	Main program	By name, if provided, in PROGRAM statement and by $$MAIN$$ if name not provided
	Procedure	By name in PROCEDURE statement
	Function	By name in FUNCTION statement

TABLE 5 (CONTINUED)

Language	*Program Block*	*Identification*
COBOL 74	One complete program	By name specified in PROGRAM-ID statement
RPG II	Main program	By RPG$MAIN
	Subroutine	By name in BEGSR statement
C	Function	By function name

- Watch List

  The watch list is an internal Debugger table holding the variables that you want to trace during your program's execution. Use the WATCH command to specify the variables

- Special Characters

  The Debugger uses special characters either to do certain things or to be part of command syntax. See the list below. You or your System Administrator can change the erase and kill characters to other characters.

Erase character (")
Erases the previous character typed The double-quote is the system default

Kill character (?)
Causes the line typed thus far to be ignored. The question mark is the system default.

Backslash (\)
Qualifies a program block name in breakpoints, variable definitions and statement definitions.

Left bracket ([)
Begins an action list.

Right bracket (])
Terminates an action list

Quotation mark (' ")
Encloses a text string (You may use the double quote if you change the erase character or use the escape character with it) The Debugger interprets the text string literally It ignores the special meanings of separators, left and right brackets, and the type of quotation mark that did not begin the string (double quote if the string is enclosed by single quotes and vice versa) To include the same type of quote in a text string, supply two consecutive marks

Separator character (;)
Separates multiple commands on one line The semicolon is the Debugger default

Escape (^)
Entered directly in front of special and regular (nonspecial) characters, it gives them different meanings It negates the special meanings of certain special characters and gives special meanings to normal characters The circumflex, or up arrow, is the Debugger default

SPECIAL CONSIDERATIONS

FOR ALL LANGUAGES

Close Data Files Before Using RESTART: If your program is using one or more PRIMOS data files and you have suspended execution, you may not be able to use RESTART to rerun the program unless you close the input file.

Enter the ! command, the PRIMOS CLOSE command, and the name of the input file you want to close.

Closing files with CLOSE ALL: If your program closes file units indiscriminately (with CLOSE ALL), specify the FULL_INIT option on the DBG command line. Do not give the CLOSE ALL command when using quick initialization.

On-units for ILLEGAL_INST$ or ANY$: If your program creates an on-unit for the system condition ILLEGAL_INST$ or ANY$, the on-unit is invoked when breakpoints are encountered.

Therefore, if your program creates on-units for these conditions, do not use these Debugger commands: BREAKPOINT, TRACEPOINT, STEP, STEPIN, STRACE, and VTRACE.

Using Specific Segments in the Range 4001 through 4037: If your program uses specific segments in the 4001 through 4037 range without allocating them in SEG, the Debugger may overwrite them for its own storage.

Use the A/SYMBOL command for common blocks in SEG. For example, the SEG command A/SYMBOL TEMP1 4027 177777 tells the Debugger that the program is using segment 4027 for common blocks

FOR FORTRAN IV

Compile With the -64V or -DYNM Option: You must use the -64V or -DYNM option along with the -DEBUG option when compiling a FORTRAN IV program

Messages for Completed Execution: If your program block calls EXIT, you receive one of the following messages.

- "program stop at (statement-id)"
- "program exit from (statement-id)"

Exit Breakpoints and Alternate Returns: Program blocks that execute alternate returns execute a GOTO statement to a label. The label value would usually be supplied as an argument to the block. If a program executes an alternate return, you cannot use these Debugger features:

- Exit breakpoints
- Exit tracepoints

- OUT command
- CALL command
- Entry/exit tracing
- Statement tracing
- Value tracing

FOR FORTRAN 77

Messages For Completed Execution: If a program block calls EXIT, you receive one of these messages:

- "program stop at (statement-id)"
- "program exit from (statement-id)"

Suspended Execution at Entry: When execution is suspended at an entry to a program block, you cannot evaluate.

- Adjustable character arguments
- Adjustable arrays
- Assumed-size arrays

Execute the program up to the first statement and you can evaluate these values.

FOR PASCAL

There are no special considerations for Pascal.

FOR PL1 SUBSET G

There are no special considerations for PL1 Subset G.

FOR COBOL 74

Data Types in COBOL 74: Some of the names of data types in the Debugger differ from their COBOL equivalents as shown in Table 6.

TABLE 6
DATA TYPE EQUIVALENTS
COBOL/DEBUGGER

COBOL 74	*Debugger*
ALPHANUMERIC DISPLAY (PIC X)	alphanumeric
NUMERIC DISPLAY (PIC 9)	trailing overpunch
COMPUTATIONAL	binary-1
COMPUTATIONAL-1 (real)	computational-1
COMPUTATIONAL-2 (double precision real)	computational-2
COMPUTATIONAL-3 (packed decimal)	computational-3

Some Debugger data types do not exist in COBOL, so you cannot use some of the Debugger's built-in functions to evaluate expressions.

Breakpoints on Paragraph Headings: Breakpoints and tracepoints may be set on paragraph headings (the COBOL equivalent of labels) If a paragraph heading begins with a number, put a $ before it to distinguish it from a line number

One Program Block. COBOL does not support procedures as they are known to PASCAL and PL1G However, a called program acts like a program block

Reinitializing With LET: When you use RESTART, the Debugger does not reinitialize the variables initialized in the WORKING-STORAGE section of the program To test if the program is changing a variable correctly, you can reinitialize some data variables with the LET command, and then use RESTART

Record Element Names: Although the Debugger lists record elements in the form NAME1 NAME2 NAME3, you still have to enter these elements in the COBOL format when the language is defined as COBOL The COBOL format is NAME1 OF NAME2 OF NAME3

FOR RPG II

Setting Breakpoints: Set breakpoints only on calculation statements, which are the only executable statements

Using SOURCE: If your source program or output file is set up for 80 columns, some lines may wrap around to the next line when displayed with SOURCE

Evaluating Variables in RPG II: The names of data types in the Debugger differ from their RPG II equivalents, as shown in Table 7.

TABLE 7
DATA TYPE EQUIVALENTS
RPG II/DEBUGGER

RPG Variable Type	*Debugger Data Type*
Field	Alphanumeric or trailing overpunch
Data Structure	Alphanumeric
Array	Alphanumeric or trailing overpunch
Table	Alphanumeric or trailing overpunch
Table Index	Binary-1 (15)
Indicator	Binary-1 (15) external

Arrays and tables are one-dimensional arrays in RPG II. Each table has an internal index that references the currently selected element of the table.

- Reference the internal index by the name IX$yyy, where *yyy* are the last characters in the name of table TAB*yyy* (for example, IX$ABC for TABABC)
- Change the internal index with the LET command
- Reference indicators by the name IND$xx, where *xx* is any legal RPG indicator For example, IND$L3 is a reference for the L3 indicator The value for an indicator is always 0 or 1

Using RESTART: If you have a suspended program execution and you are using an input file, you must close the file before using RESTART (To use an input file you specify DISK as the input device.)

To start execution, enter

```
> ! CLOSE filename
> RESTART
```

Input and Output: Close the input or output file before using SOURCE NAME if

- You have specified DISK as the input device
- You have specified DISK or PRINTERS as the output device
- You want to examine either the input or the output file while program execution is suspended

To examine an input or output file, enter

```
> ! CLOSE filename
> SOURCE NAME filename
```

FOR C

Assignment: To assign values, use the evaluation (.) command with any of the special C assignment operators (= += *= /= %= >>= <<= &= ^= |= in addition to LET). The Debugger evaluates expressions exactly the same way as a C program does.

Do not assign a value to an rvalue, for instance, an expression within parentheses. It is an illegal operation but the Debugger does not report the error.

Prime C Operators: The Debugger supports all Prime C operators except the CAST operator Debugger operators for evaluating expressions are functionally identical to the corresponding operators in the Prime C compiler and produce the same expected side effects

Special Characters: The Debugger does not support the C escape character (/) Use the Debugger escape character (^) Generate a null character (/0) by evaluating a null string (" ")

Defaults for Constants: The default for a floating point constant is DOUBLE The default for an integer constant is LONG

The ?: Construct: The Debugger does not support the ? construct Use the IF-ELSE construct instead

DEBUGGER DEFINED BLOCKS

The Debugger defines two program blocks that contain all program blocks These blocks make it possible to reference variables globally (outside the current evaluation environment)

- $DBG
- $EXTERNAL

$DBG PROGRAM BLOCK

The Debugger defines three special Debugger-defined variables within $DBG $MR, $COUNT, and $COUNTERS, all built-in functions are "owned" by the $DBG block also

► $MR

Contains the values of the machine registers, as shown in Table 8 on the next page

► $COUNT

Contains the value of the breakpoint counter for the most recent breakpoint or tracepoint Useful in

conditional breakpoint action lists (with the IF command).

TABLE 8
MACHINE REGISTERS

Register Category	*Description*
SAVE-MASK	Bit string indicating which registers have been saved
V	V-mode registers (A, B, L, X, Y, E)
I	I-mode registers (general registers 0 through 7)
BR	Base registers (PB, SB, LB, XB)
KEYS	Process keys

▶ $COUNTERS

Counts information related to program size and program symbols, as shown in the table below. Valid only if you specify the -FULL_INIT option on the DBG command line To display the values enter.

```
> : $COUNTERS
```

The meaning of each type of count specified by $COUNTERS is listed below

STATEMENTS
Number of statements in procedures compiled in debug mode

OUTER_BLOCKS
Number of external program blocks compiled in debug and production modes

TOTAL_BLOCKS
Number of external and internal program blocks compiled in debug and production modes

TOP_LEVEL_SYMBOLS
Number of declared symbols, not including structure members

NON_TOP_LEVEL_SYMBOLS
Total number of structure members

PERMANENT_STORAGE
Number of halfwords allocated by the Debugger for the user program's symbol table

DATA_FILE_SIZE
Size in halfwords of the Debugger data file contained in the program's SEG file

FUNCTIONS

The Debugger defines built-in functions for the supported languages within $DBG You can use these functions in expressions They are listed in the introduction to the section "Alphabetical List of Debugger Commands."

$EXTERNAL PROGRAM BLOCK

Invisible to users, the Debugger's $EXTERNAL program block may be used to reference external variables that have not been declared in the current evaluation environment.

CONVERSION CHARTS

Use the decimal to-octal chart only if you know how to add or subtract in octal An asterisk (*) denotes negative numbers when signed

Octal to Decimal		*Decimal to Octal*	
1 to 7	1 to 7	1 to 7	1 to 7
10	8	8	10
11	9	9	11
12	10	10	12
13	11	11	13
14	12	12	14
15	13	13	15
16	14	14	16
17	15	15	17
20	16	16	20
30	24	17	21
40	32	18	22
50	40	19	23
60	48	20	24
70	56	30	36
100	64	40	50
200	128	50	62
300	192	60	74
400	256	70	106
500	320	80	120
600	384	90	132
700	448	100	144
1000	512	200	310

Octal to Decimal		*Decimal to Octal*	
2000	1024	300	454
3000	1536	400	620
4000	2048	500	764
5000	2560	600	1130
6000	3072	700	1274
7000	3584	800	1440
10000	4096	900	1604
20000	8192	1000	1750
30000	12288	2000	3720
40000	16384	3000	5670
50000	20480	4000	7640
60000	24576	5000	11610
70000	28672	6000	13560
100000	32768*	7000	15530
177777	65535*	8000	17500
		9000	21450
		10000	23420
		20000	47040
		30000	72460
		*40000	116100
		*50000	141520
		*60000	165140
		*65535	177777

* Indicates negative numbers when signed

ASCII CHARACTER SET (NON PRINTING)

Octal Value	*ASCII Char*	*Comments/Prime Usage*	*Control Char*
200	NULL	Null character—filler	^
201	SOH	Start of header (communications)	^ A
202	STX	Start of text (communications)	^ B
203	ETX	End of text (communications)	^ C
204	EOT	End of transmission (communications)	^ D
205	ENQ	End of I D (communications)	^ E
206	ACK	Acknowledge affirmative (communications)	^ F
207	BEL	Audible alarm (bell)	^ G
210	BS	Back space one position (carriage control)	^ H
211	HT	Physical horizontal tab	^ I
212	LF	Line feed, ignored as terminal input	^ J
213	VT	Physical vertical tab (carriage control)	^ K
214	FF	Form feed (carriage control)	^ L
215	CR	Carriage return (carriage control) (1)	^ M
216	SO	Shift out (switch to alternate character set) (2)	^ N
217	SI	Shift in (return to standard character set)	^ O
220	DLE	Data link escape (3)	^ P
221	DC1	Device control 1 (4)	^ Q
222	DC2	Device control 2 (5)	^ R
223	DC3	Device control 3 (6)	^ S
224	DC4	Device control 4 (7)	^ T
225	NAK	Negative acknowledgement (communications)	^ U
226	SYN	Synchronization (communications)	^ V

ASCII CHARACTER SET (NON PRINTING) (CONTINUED)

Octal Value	*ASCII Char*	*Comments/Prime Usage*	*Control Char*
227	ETB	End of transmission block (communications)	^ W
230	CAN	Cancel	^ X
231	EM	End of Medium	^ Y
232	SUB	Substitute	^ Z
233	ESC	Escape	^ [
234	FS	File separator	^ \
235	GS	Group separator	^]
236	RS	Record separator	^ ^
237	US	Unit separator	^ –

Notes

(1) Interpreted as NL (that is, line feed) at the application program level

(2) Examples of alternate character sets include red ribbon characters or graphics characters

(3) Has multiple functions, including
- From a terminal aborts (quits) user programs and returns to PRIMOS level
- Within a file specifies relative copy, next byte gives number of bytes to copy from corresponding position of previous line

(4) Has multiple functions including
- From a terminal XON, resume transmission
- Within a file relative horizontal tab, next byte specifies number of spaces to insert

(5) Can have multiple functions, including
- Within a file half line feed forward (carriage control)

(6) Has multiple functions, including
- From a terminal XOFF, suspends (freezes) transmission
- Within a file relative vertical tab, next byte specifies number of lines to insert

(7) Can have multiple functions including
- Within a file half line feed reverse (carriage control)

ASCII CHARACTER SET (PRINTING)

OCTAL Value	*ASCII Character*	*OCTAL Value*	*ASCII Character*	*OCTAL Value*	*ASCII Character*
240	SPACE(1)	300	@	340	`(9)
241	!	301	A	341	a
242	"(2)	302	B	342	b
243	#	303	C	343	c
244	$ (3)	304	D	344	d
245	%	305	E	345	e
246	&	306	F	346	f
247	'(4)	307	G	347	g
250	(	310	H	350	h
251	)	311	I	351	i
252	*	312	J	352	j
253	+	313	K	353	k
254	,(5)	314	L	354	l
255	—	315	M	355	m
256	.	316	N	356	n
257	/	317	O	357	o
260	0	320	P	360	p
261	1	321	Q	361	q
262	2	322	R	362	r
263	3	323	S	363	s
264	4	324	T	364	t
265	5	325	U	365	u
266	6	326	V	366	v
267	7	327	W	367	w
270	8	330	X	370	x
271	9	331	Y	371	y
272	:	332	Z	372	z
273	;	333	[	373	{
274	<	334	\	374	|
275	=	335	]	375	}
276	>	336	^(7)	376	~(10)
277	?(6)	337	_ (8)	377	DEL(11)

(1) Space forward one position
(2) Terminal usage. default erase character (erases previous character)
(3) (£) in British use
(4) Apostrophe/single quote
(5) Comma
(6) Terminal usage default kill character (kills line)
(7) 1963 standard (up-arrow)
(8) 1963 standard (back-arrow)
(9) Grave accent
(10) 1963 standard ESC
(11) Rubout, ignored, unless the user has assigned it a particular action (for example, as the erase or kill character)