Reference and Messages

Version 7.1
Reference and Messages

Version 7.1
Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 363.

Third Edition (December 2006)

This edition applies to Debug Tool for z/OS, Version 7.1 (Program Number 5655-R44), with the PTF for APAR PK34211 applied, which supports the following compilers:

- AD/Cycle C/370 Version 1 Release 2 (Program Number 5688-216)
- C/C++ for MVS/ESA Version 3 (Program Number 5655-121)
- C/C++ feature of OS/390 (Program Number 5647-A01)
- C/C++ feature of z/OS (Program Number 5694-A01)
- OS/VS COBOL, Version 1 Release 2.4 (5740-CB1) - with limitations
- VS COBOL II Version 1 Release 3 and Version 1 Release 4 (Program Numbers 5668-958, 5688-023) - with limitations
- COBOL/370™ Version 1 Release 1 (Program Number 5688-197)
- COBOL for MVS & VM Version 1 Release 2 (Program Number 5688-197)
- COBOL for OS/390 & VM Version 2 (Program Number 5648-A25)
- Enterprise COBOL for z/OS and OS/390 Version 3 (Program Number 5655-G53)
- High Level Assembler for MVS & VM & VSE Version 1 Release 4, Version 1 Release 5 (Program Number 5696-234)
- PL/I for MVS & VM Version 1 Release 1 (Program Number 5688-235)
- VisualAge PL/I for OS/390 Version 2 Release 2 (Program Number 5655-B22)
- Enterprise PL/I for z/OS and OS/390 Version 3.5 or earlier (Program Number 5655-H31)

Parts of this edition apply to Debug Tool Utilities and Advanced Functions for z/OS, Version 7.1 (Program Number 5655-R45).

This edition also applies to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters.

You can order publications online at www.ibm.com/shop/publications/order, or order by phone or fax. IBM Software Manufacturing Solutions takes publication orders between 8:30 a.m. and 7:00 p.m. Eastern Standard Time (EST). The phone number is (800)879-2755. The fax number is (800)445-9269.

You can find out more about Debug Tool by visiting the IBM Web site for Debug Tool at: http://www.ibm.com/software/awdtools/debugtool

© Copyright International Business Machines Corporation 1992, 2006. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
## Contents

**About this document** ........................ vii
Who might use this document .................. vii
Accessing z/OS licensed documents on the Internet vii
Using LookAt to look up message explanations ...... viii
How this document is organized ................ viii
Terms used in this document ................... ix
How to read syntax diagrams .................. x
  Symbols ................................ x
  Syntax items ............................. x
  Syntax examples .......................... xi
How to send your comments .................... xii

**Summary of changes** ........................ xiii
Changes introduced with PTF for APAR PK34211 xiii
Changes introduced with Debug Tool V7.1 xiii
Changes introduced with Debug Tool Utilities and
  Advanced Functions V7.1 .................... xv

**Chapter 1. Debug Tool run-time options**  1
  Non-Language Environment positional parameter . 2
  COUNTRY run-time option .................. 2
  NATLANG run-time option .................. 2
  TEST run-time option ..................... 3
    Syntax of the TEST run-time option ..... 3
  TRAP run-time option ..................... 9

**Chapter 2. Common syntax elements in**
  **Debug Tool commands** 11
  block_name ................................ 11
  block_spec ................................ 11
  condition ................................ 12
  compile_unit_name ........................ 12
  cu_spec ................................ 13
  expression ................................ 14
  load_module_name ........................ 14
  load_spec ................................ 15
  references ................................ 15
  statement_id ............................. 16
  statement_id_range and stmt_id_spec ..... 16
    Specifying a range of statements ...... 17
  statement_label .......................... 17

**Chapter 3. Syntax for assembler and**
  **disassembly expressions** 19
  Common syntax elements .................... 19
  Operators ................................ 20
    Operators that can be used in any expression 20
    Operators that can be used only in conditional
      expressions .......................... 21

**Chapter 4. Syntax for OS/VS COBOL**
  **expressions** 23
  Restrictions on OS/VS COBOL expressions .... 23
  Common syntax elements ................... 24
    Operators ................................ 24
      Operators that can be used in any expression 24
      Operators that can be used only in conditional
        expressions ........................ 25

**Chapter 5. Debug Tool commands** 27
  ? command ................................ 30
  ALLOCATE command ........................ 31
  ANALYZE command (PL/I) .................. 32
  Assignment command (assembler and disassembly) 33
    Assignment rules ........................ 33
  Assignment command (OS/VS COBOL) ........ 36
  Assignment command (PL/I) ................ 36
  AT command ................................ 37
    every_clause syntax .................... 40
    AT ALLOCATE (PL/I) .................... 40
    AT APPEARANCE ......................... 42
    AT CALL ................................ 43
    AT CHANGE ............................. 45
    AT CURSOR (full-screen mode) .......... 50
    AT DATE (COBOL) ........................ 51
    AT DELETE ............................. 51
    AT ENTRY and AT EXIT .................. 52
    AT GLOBAL ............................ 54
    AT LABEL .............................. 55
    AT LINE ................................ 58
    AT LOAD ................................ 58
    AT OCCURRENCE ......................... 60
    AT OFFSET (disassembly) ............... 63
    AT PATH ................................ 63
    AT Prefix (full-screen mode) .......... 64
    AT STATEMENT .......................... 65
    AT TERMINATION ......................... 66
  BEGIN command (PL/I) ................. 67
  block command (C and C++) .............. 68
  break command (C and C++) ............... 69
  CALL command ........................... 69
    CALL %CEBR ....................... 70
    CALL %CECI ...................... 71
    CALL %DUMP ....................... 71
    CALL %FA ......................... 76
    CALL %HOGAN ....................... 76
    CALL %VER ......................... 77
    CALL entry_name (COBOL) ............ 77
    CALL procedure ..................... 79
  CLEAR command .......................... 79
    CLEAR prefix (full-screen mode) ...... 83
  COMMENT command ........................ 84
  COMPUTE command (COBOL) .............. 84
  CURSOR command (full-screen mode) ..... 85
  Declarations (assembler, disassembly, and
    OS/VS COBOL) ........................ 86
  Declarations (C and C++) ................ 87
  Declarations (COBOL) .................. 89
  DECLARE command (PL/I) ............... 91
DESCRIBE command ........................................ 93
DISABLE command ........................................ 96
DISABLE prefix (full-screen mode) ..................... 97
DO command (assembler, disassembly, and OS/VS COBOL) ........................................ 98
do/while command (C and C++) .......................... 98
DO command (PL/I) ........................................ 99
ENABLE command ......................................... 101
ENABLE prefix (full-screen mode) ...................... 102
EVALUATE command (COBOL) ............................ 102
Expression command (C and C++) ........................ 104
FIND command ........................................... 105
for command (C and C++) ................................ 109
FREE command ........................................... 110
GO command ............................................. 110
GOTO command .......................................... 111
GOTO LABEL command .................................... 112
IF command (assembler, disassembly, and OS/VS COBOL) ........................................ 113
if command (C and C++) .................................. 114
IF command (COBOL) ...................................... 115
   Allowable comparisons for the IF command (COBOL) ........................................ 116
IF command (PL/I) ........................................ 119
IMMEDIATE command (full-screen mode) .............. 120
INPUT command (C, C++, and COBOL) ................. 121
JUMPTO command ........................................ 122
JUMPTO LABEL command .................................. 123
LIST command ........................................... 124
   LIST (blank) .......................................... 125
   LIST AT ............................................... 125
   LIST CALLS ......................................... 128
   LIST CURSOR (full-screen mode) .................... 128
   LIST expression ...................................... 129
   LIST FREQUENCY ..................................... 132
   LIST LAST ........................................... 132
   LIST LINE NUMBERS ................................ 133
   LIST LINES .......................................... 133
   LIST MONITOR ...................................... 134
   LIST NAMES ......................................... 134
   LIST ON (PL/I) ...................................... 136
   LIST PROCEDURES ................................... 136
   LIST REGISTERS .................................... 137
   LIST STATEMENT NUMBERS ......................... 137
   LIST STATEMENTS .................................... 138
   LIST STORAGE ....................................... 139
LOAD command ........................................... 140
LOADDEBUGDATA ........................................ 141
MONITOR command ...................................... 142
MOVE command (COBOL) .................................. 145
   Allowable moves for the MOVE command (COBOL) ........................................ 146
Null command ........................................... 149
ON command (PL/I) ...................................... 149
PANEL command (full-screen mode) ..................... 151
PERFORM command (COBOL) .............................. 153
PLAYBACK commands ..................................... 155
   PLAYBACK ENABLE command ......................... 156
   PLAYBACK START command ........................... 157
   PLAYBACK FORWARD command ....................... 158
   PLAYBACK BACKWARD command ..................... 158
PLAYBACK STOP command ................................ 159
PLAYBACK DISABLE command ......................... 159
Prefix commands (full-screen mode) ................... 159
PROCEDURE command ................................... 160
QUALIFY commands (full-screen mode) ............... 160
QUERY command ........................................ 161
   QUERY for command (full-screen mode) ........... 161
   QUERY prefix (full-screen mode) ................... 166
QUIT command .......................................... 166
QQUIT command ......................................... 167
RESTORE command ...................................... 168
RETRIEVE command (full-screen mode) .............. 169
RUN command .......................................... 170
RUNTO command ......................................... 170
RUNTO prefix command (full-screen mode) .......... 171
SCROLL command (full-screen mode) ................... 171
SELECT command (PL/I) ................................ 173
SET command .......................................... 174
   SET ASSEMBLER ...................................... 176
   SET AUTOMONITOR .................................. 177
   SET CHANGE ........................................ 179
   SET COLOR (full-screen and line mode) .......... 180
   SET COUNTRY ....................................... 182
   SET DBCS ........................................... 182
   SET DEFAULT LISTINGS ............................. 183
   SET DEFAULT SCANL (full-screen mode) .......... 184
   SET DEFAULT VIEW .................................. 185
   SET DEFAULT WINDOW (full-screen mode) ....... 185
   SET DISASSEMBLY .................................... 186
   SET DYNDEBUG ...................................... 188
   SET ECHO ............................................ 188
   SET EQUATE ......................................... 188
   SET EXECUTE ........................................ 189
   SET FREQUENCY ..................................... 190
   SET HISTORY ........................................ 191
   SET INTERCEPT (C, C++, and COBOL) ............... 191
   SET KEYS (full-screen mode) ....................... 192
   SET LDD ............................................. 193
   SET LIST TABULAR ................................... 193
   SET LOG ............................................. 194
   SET LOG NUMBERS (full-screen mode) .............. 195
   SET LONGCUNAME .................................... 195
   SET MONITOR ....................................... 196
   SET MSGID ........................................... 198
   SET NATIONAL LANGUAGE ............................ 198
   SET PAGE ........................................... 199
   SET PFKEY ........................................... 200
   SET PROGRAMMING LANGUAGE ....................... 201
   SET PROMPT (full-screen mode) ..................... 202
   SET QUALIFY ........................................ 203
   SET REFRESH (full-screen mode) ..................... 205
   SET RESTORE ........................................ 205
   SET REWRITE ......................................... 207
   SET SAVE ............................................ 207
   SET SCREEN (full-screen and line mode) .......... 210
   SET SCROLL DISPLAY (full-screen mode) .......... 211
   SET SEQUENCE (PL/I) ................................ 211
   SET SOURCE ......................................... 211
   SET SUFFIX (full-screen mode) ........................ 214
   SET TEST ............................................ 214
   SET WARNING (C, C++, COBOL, and PL/I) ........ 215
   SET command (COBOL) ................................ 216
About this document

Debug Tool combines the richness of the z/OS® environment with the power of Language Environment® to provide a debugger for programmers to isolate and fix their program bugs and test their applications. Debug Tool gives you the capability of testing programs in batch, using a nonprogrammable terminal in full-screen mode, or using a workstation interface to remotely debug your programs.

This document contains descriptions of the commands, functions, and variables available through Debug Tool and Debug Tool Utilities and Advanced Functions, as well as the messages that you might see as you use Debug Tool. Debug Tool commands are similar to commands from the supported high-level languages (HLLs).

Who might use this document

This document is intended for programmers using Debug Tool to debug high-level languages (HLLs) with Language Environment and assembler programs either with or without Language Environment. Throughout this document, the HLLs are referred to as C, C++, COBOL, and PL/I.

The following operating systems and subsystems are supported:

- z/OS
  - CICS®
  - DB2®
  - IMS™
  - JES batch
  - TSO
  - UNIX® System Services in remote debug mode or full-screen mode through a VTAM terminal only
  - WebSphere® in remote debug mode or full-screen mode through a VTAM terminal only

To use this document and debug a program written in one of the supported languages, you need to know how to write, compile, and run such a program.

Accessing z/OS licensed documents on the Internet

z/OS licensed documentation is available on the Internet in PDF format at the IBM® Resource Link™ Web site at:

http://www.ibm.com/servers/resourcelink

Licensed documents are available only to customers with a z/OS license. Access to these documents requires an IBM Resource Link user ID and password, and a key code. With your z/OS order you received a Memo to Licensees, (GI10-0671), that includes this key code.

To obtain your IBM Resource Link user ID and password, log on to:

http://www.ibm.com/servers/resourcelink

To register for access to the z/OS licensed documents:

1. Sign in to Resource Link using your Resource Link user ID and password.
2. Select User Profiles located on the left-hand navigation bar.

**Note:** You cannot access the z/OS licensed documents unless you have registered
for access to them and received an e-mail confirmation informing you that
your request has been processed.

Printed licensed documents are not available from IBM.

You can use the PDF format on either z/OS Licensed Product Library CD-ROM or
IBM Resource Link to print licensed documents.

---

**Using LookAt to look up message explanations**

LookAt is an online facility that lets you look up explanations for most of the IBM
messages you encounter, as well as for some system abends and codes. Using
LookAt to find information is faster than a conventional search because in most
cases LookAt goes directly to the message explanation.

You can use LookAt from the following locations to find IBM message
explanations for z/OS elements and features, z/VM®, VSE/ESA™, and Clusters for
AIX® and Linux®:

- The Internet. You can access IBM message explanations directly from the LookAt
- Your z/OS TSO/E host system. You can install code on your z/OS or z/OS.e
  systems to access IBM message explanations, using LookAt from a TSO/E
  command line (for example, TSO/E prompt, ISPF, or z/OS UNIX System
  Services running OMVS).
- Your Microsoft® Windows® workstation. You can install code to access IBM
  message explanations on the z/OS Collection (SK3T-4269), using LookAt from a
  Microsoft Windows command prompt (also known as the DOS command line).
- Your wireless handheld device. You can use the LookAt Mobile Edition with a
  handheld device that has wireless access and an Internet browser (for example,
  Internet Explorer for Pocket PCs, Blazer, or Eudora for Palm OS, or Opera for
  Linux handheld devices). Link to the LookAt Mobile Edition from the LookAt
  Web site.

You can obtain code to install LookAt on your host system or Microsoft Windows
workstation from a disk on your z/OS Collection (SK3T-4269), or from the LookAt
Web site (click **Download**, and select the platform, release, collection, and location
that suit your needs). More information is available in the LOOKAT.ME files
available during the download process.

---

**How this document is organized**

This document is divided into areas of similar information for easy retrieval of
appropriate information. The following list describes how the information is
grouped:

- Chapter 1 describes the syntax of the TEST run-time option.
- Chapters 2, 3, 4, and 5 describe the complete syntax of the Debug Tool
  commands.
- Chapters 6 and 7 describe the syntax of Debug Tool built-in functions and
  variables.
- Chapters 8, 9, 10, and 11 lists all the messages.
The last several chapters list notices, bibliography, and glossary of terms.

### Terms used in this document

Because of differing terminology among the various programming languages supported by Debug Tool, as well as differing terminology between platforms, a group of common terms has been established. The table below lists these terms and their equivalency in each language.

<table>
<thead>
<tr>
<th>Debug Tool term</th>
<th>C and C++ equivalent</th>
<th>COBOL equivalent</th>
<th>PL/I equivalent</th>
<th>assembler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile unit</td>
<td>C and C++ source file</td>
<td>Program or class</td>
<td>• Program</td>
<td>CSECT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• PL/I source file for Enterprise PL/I</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• A package statement or the name of the main procedure for Enterprise PL/I</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block</td>
<td>Function or compound statement</td>
<td>Program, nested program, method or PERFORM group of statements</td>
<td>Block</td>
<td>CSECT</td>
</tr>
<tr>
<td>Label</td>
<td>Label</td>
<td>Paragraph name or section name</td>
<td>Label</td>
<td>Label</td>
</tr>
</tbody>
</table>

**Notes:**

1. The PL/I program must be compiled with the Enterprise PL/I for z/OS, Version 3.5, compiler with the PTFs for APARs PK35230 and PK35489 applied and must be running in Language Environment Version 1.4 through 1.8 with the PTF for APAR PK33738 applied.

Debug Tool provides facilities that apply only to programs compiled with specific levels of compilers. Because of this, [Debug Tool Reference and Messages](#) uses the following terms:

**assembler**

Refers to assembler programs with debug information assembled by using the High Level Assembler (HLASM).

**COBOL**

Refers to the all COBOL compilers and dialects supported by Debug Tool except OS/VS COBOL.

**disassembly or disassembled**

Refers to high-level language programs compiled without debug information or assembler programs without debug information. The debugging support Debug Tool provides for these programs is through the disassembly view.
Enterprise PL/I
Refers to the Enterprise PL/I for z/OS and OS/390 and the VisualAge® PL/I for OS/390 compilers.

full-screen mode through a VTAM terminal
Refers to the debugging mode that requires a second terminal, a VTAM® terminal, be started and used to debug an application. After the VTAM terminal has been started, you can optionally use the Debug Tool Terminal Interface Manager to identify that terminal to Debug Tool by using a user ID instead of a LU name.

OS/VS COBOL
Refers to COBOL programs compiled using the IBM OS/VS COBOL compiler.

As you read through the information in this document, remember that OS/VS COBOL programs are non-Language Environment programs, even though you might have used Language Environment libraries to link and run your program. Please read the information regarding non-Language Environment programs for instructions on how to start Debug Tool and debug OS/VS COBOL programs, unless OS/VS COBOL-specific information is provided.

PL/I
Refers to all levels of PL/I compilers. Exceptions will be noted in the text that describe which specific PL/I compiler is being referenced.

separate debug file
Refers to the Enterprise COBOL for z/OS and OS/390 side file and the Enterprise PL/I for z/OS Version 3 Release 5 separate debug file.

How to read syntax diagrams
This section describes how to read syntax diagrams. It defines syntax diagram symbols, items that may be contained within the diagrams (keywords, variables, delimiters, operators, fragment references, operands) and provides syntax examples that contain these items.

Syntax diagrams pictorially display the order and parts (options and arguments) that comprise a command statement. They are read from left to right and from top to bottom, following the main path of the horizontal line.

Symbols
The following symbols may be displayed in syntax diagrams:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>➞</td>
<td>Indicates the beginning of the syntax diagram.</td>
</tr>
<tr>
<td>➞</td>
<td>Indicates that the syntax diagram is continued to the next line.</td>
</tr>
<tr>
<td>➞</td>
<td>Indicates that the syntax is continued from the previous line.</td>
</tr>
<tr>
<td>➞</td>
<td>Indicates the end of the syntax diagram.</td>
</tr>
</tbody>
</table>

Syntax items
Syntax diagrams contain many different items. Syntax items include:
- Keywords - a command name or any other literal information.
- Variables - variables are italicized, appear in lowercase and represent the name of values you can supply.
• Delimiters - delimiters indicate the start or end of keywords, variables, or operators. For example, a left parenthesis is a delimiter.
• Operators - operators include add (+), subtract (-), multiply (*), divide (/), equal (=), and other mathematical operations that may need to be performed.
• Fragment references - a part of a syntax diagram, separated from the diagram to show greater detail.
• Separators - a separator separates keywords, variables or operators. For example, a comma (,) is a separator.

Keywords, variables, and operators may be displayed as required, optional, or default. Fragments, separators, and delimiters may be displayed as required or optional.

<table>
<thead>
<tr>
<th>Item type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>Required items are displayed on the main path of the horizontal line.</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional items are displayed below the main path of the horizontal line.</td>
</tr>
<tr>
<td>Default</td>
<td>Default items are displayed above the main path of the horizontal line.</td>
</tr>
</tbody>
</table>

**Syntax examples**
The following table provides syntax examples.

<table>
<thead>
<tr>
<th>Item</th>
<th>Syntax example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required item.</td>
<td>Required items appear on the main path of the horizontal line. You must</td>
</tr>
<tr>
<td></td>
<td>specify these items.</td>
</tr>
<tr>
<td>Required choice.</td>
<td>A required choice (two or more items) appears in a vertical stack on the</td>
</tr>
<tr>
<td></td>
<td>main path of the horizontal line. You must choose one of the items in the</td>
</tr>
<tr>
<td></td>
<td>stack.</td>
</tr>
<tr>
<td>Optional item.</td>
<td>Optional items appear below the main path of the horizontal line.</td>
</tr>
<tr>
<td>Optional choice.</td>
<td>An optional choice (two or more items) appears in a vertical stack below</td>
</tr>
<tr>
<td></td>
<td>the main path of the horizontal line. You may choose one of the items in the</td>
</tr>
<tr>
<td></td>
<td>stack.</td>
</tr>
<tr>
<td>Default.</td>
<td>Default items appear above the main path of the horizontal line. The</td>
</tr>
<tr>
<td></td>
<td>remaining items (required or optional) appear on (required) or below (optional)</td>
</tr>
<tr>
<td></td>
<td>the main path of the horizontal line. The following example displays a</td>
</tr>
<tr>
<td></td>
<td>default with optional items.</td>
</tr>
</tbody>
</table>
Table 1. Syntax examples (continued)

<table>
<thead>
<tr>
<th>Item</th>
<th>Syntax example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable.</td>
<td><img src="image" alt="Syntax Example" /></td>
</tr>
<tr>
<td>Variables appear in lowercase italics. They represent names or values.</td>
<td><img src="image" alt="Syntax Example" /></td>
</tr>
<tr>
<td>Repeatable item.</td>
<td><img src="image" alt="Syntax Example" /></td>
</tr>
<tr>
<td>An arrow returning to the left above the main path of the horizontal line indicates an item that can be repeated.</td>
<td><img src="image" alt="Syntax Example" /></td>
</tr>
<tr>
<td>A character within the arrow means you must separate repeated items with that character.</td>
<td><img src="image" alt="Syntax Example" /></td>
</tr>
<tr>
<td>An arrow returning to the left above a group of repeatable items indicates that one of the items can be selected, or a single item can be repeated.</td>
<td><img src="image" alt="Syntax Example" /></td>
</tr>
<tr>
<td>Fragment.</td>
<td><img src="image" alt="Syntax Example" /></td>
</tr>
<tr>
<td>The fragment symbol indicates that a labelled group is described below the main syntax diagram. Syntax is occasionally broken into fragments if the inclusion of the fragment would overly complicate the main syntax diagram.</td>
<td><img src="image" alt="Syntax Example" /></td>
</tr>
</tbody>
</table>

How to send your comments

Your feedback is important in helping us to provide accurate, high-quality information. If you have comments about this document or any other Debug Tool documentation, contact us in one of these ways:

- Use the Online Readers’ Comment Form at [www.ibm.com/software/awdtools/rcf/](http://www.ibm.com/software/awdtools/rcf/). Be sure to include the name of the document, the publication number of the document, the version of Debug Tool, and, if applicable, the specific location (for example, page number) of the text that you are commenting on.
- Fill out the Readers’ Comment Form at the back of this document, and return it by mail or give it to an IBM representative. If the form has been removed, address your comments to:

  IBM Corporation  
  H150/090  
  555 Bailey Avenue  
  San Jose, CA 95141-1003  
  USA  

- Fax your comments to this U.S. number: (800)426-7773.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.
Summary of changes

This section lists the key changes made to Debug Tool for z/OS and Debug Tool Utilities and Advanced Functions for z/OS that affect this document.

Changes introduced with PTF for APAR PK34211

The following changes were introduced with the PTF for APAR PK34211:

- Debug Tool can now debug C and C++ programs compiled with the FORMAT(DWARF) suboption of the DEBUG compiler option. For a description of how to compile C and C++ programs with this option, see the section "Compiling your C program with the DEBUG(FORMAT(DWARF)) option" in Debug Tool User's Guide or the section "Compiling your C++ program with the DEBUG(FORMAT(DWARF)) option" in Debug Tool User's Guide. For a description of the DEBUG compiler option, see the section "Syntax for the C and C++ DEBUG compiler option" in Debug Tool User's Guide.

- You can now specify a network identifier when you specify the VTAM logical unit (LU) identifier for a terminal used in full-screen mode through a VTAM terminal. See Debug Tool Customization Guide for a description of how to install this support. See "TEST run-time option" on page 3 for a description of the new suboption.

- If your PL/I program was compiled with the following compiler and it is running in the following environment, you can specify the package name or the name of the main procedure instead of a fully qualified data set name.
  - Enterprise PL/I for z/OS, Version 3.5, with the PTFs for APARs PK35230 and PK35489 applied
  - Language Environment Version 1.4 through 1.8 with the PTF for APAR PK33738 applied

- The size and display of windows are saved. By using the SET SAVE command, you can save the size of each window displayed in full screen mode, as well as which window is displayed in full screen mode. Then, you can use the SET RESTORE command to reinstate those settings. Changes were made to the SET RESTORE, SET SAVE, and QUERY commands to enable this feature.

- There have been some changes to the restrictions on load module names when you use the SET QUALIFY CU command. See "SET QUALIFY" on page 203 for a description of the SET QUALIFY CU command.

- For Enterprise PL/I programs, you can now specify the name of the main procedure or, if present, the package statement as the CU name. You must have the PTFs for APARs PK35230 and PK35489 applied to your Enterprise PL/I for z/OS Version 3.5 compiler to use this feature.

- Miscellaneous updates.

Changes introduced with Debug Tool V7.1

Minor changes were made to the second edition that support changes introduced with Debug Tool Version 7.1.

- The following enhancements have been made to the monitoring functions:
  - For COBOL programs, Debug Tool does not prefix the program name to the output, allowing more data to be displayed on the same line.
- You can now display the value of variables, including members of an array or structure, in a columnar format. Debug Tool provides a new command, SET MONITOR COLUMN, which you can use to indicate that you want the Monitor window to display information in columnar format.

- Debug Tool now displays a ruler, which indicates the offset from the start of the display to current cursor position.

- You can now update large variables directly in the Monitor window.

- You can now use the HEX prefix command on only one member of an array or a sublevel of a structure. Previously, you could use the HEX prefix command only on the entire array or structure.

- You can now update an array or a structure member without making the full name of the array or structure visible. Previously, you could update an array or structure member only if the full name of the array or structure was visible.

- You can now access source code (C, C++, Enterprise PL/I if a separate debug file is not used) stored in library systems that require data sets to be allocated as a DSORG DA or VSAM data set with the SUBSYS=sss allocation parameter, where ssss is a subsystem provided by the library system vendor. Debug Tool provides a method (by using EQAOPTS) that instructs Debug Tool to use the SUBSYS=sss allocation parameter when it allocates the data set.

- You can now indicate that you want COPE facilities to continue operating while Debug Tool is active. Standardware Corporation’s COPE product is used in an IMS environment to deliver some additional capabilities for applications and systems administrators. You use a new option (through EQAOPTS) to enable this behavior.

- You can now indicate what Debug Tool should do if the terminal using the full-screen mode through a VTAM terminal facility or the remote debugger is not available. Use a new option (through EQAOPTS) to select the new behavior.

- Debug Tool has enhanced the LIST, CLEAR, ENABLE, and DISABLE commands to support suspended breakpoints.

- The LIST STORAGE and STORAGE commands have been enhanced so that you can provide a starting byte offset. Previously, Debug Tool used the start of the area of storage allocated to the variable as the starting byte.

- You can now set COBOL level 88 condition variables to TRUE.

- The Debug Tool CICS control utility (DTCN) has been enhanced so that you can temporarily inactivate a profile, then reactivate it at a later time.

- DTCN has been enhanced so you can select CICS tasks to debug based on the client IP name or address.

- A new option TASK has been added to the QUIT DEBUG command to help you terminate debugging sessions that involve pseudo-conversational applications. If you specify the TASK option, Debug Tool terminates immediately. It does not wait until the end of the current CICS pseudo-conversational task, which can be indicated by, for example, an EXEC CICS RETURN TRANSID. When a new task is started in the pseudo-conversation, Debug Tool resumes debugging.

- An option is added to the Debug Tool Utilities primary panel to invoke IBM File Manager (FM) for z/OS functions — FM base, FM/DB2, and FM/IMS. (Note that File Manager is a separately installed product.)

- When you specify a preference or command file in the TEST run time option, you can specify whether Debug Tool interprets the data set name as fully or partially qualified.

- Debug Tool now supports the DEBUG compile option with FORMAT(ISD), which is available with the z/OS C/C++ Version 1.6 (and later) compiler. This option
helps you specify the granularity of the compiled-in hooks that the compiler inserts and the amount of debug data to save.

- You can now view the Debug Tool Setup Utility’s File Allocation Panel as a full screen panel by using the ShowDD command.
- New messages have been added.
- Miscellaneous updates.

Changes introduced with Debug Tool Utilities and Advanced Functions
V7.1

- The following enhancements have been made to the monitoring functions:
  - You can use a new command, SET MONITOR WRAP, to indicate how you want to display the value of a variable, which is being monitored or automonitored, in the Monitor window. Debug Tool can display the value of a variable in either a wrapped format or on a scrollable line. In a wrapped format, if the value exceeds the width of the display, Debug Tool continues the value on the next line. On a scrollable line, if the value exceeds the width of the display, you can scroll left or right to see the rest of the value.
  - You can use a new command, SET MONITOR DATATYPE, to indicate whether you want to display the data type of a variable that is being monitored or automonitored.
- You can display the source for a compile unit or CSECT (CU) before the load module containing the CU has been loaded or run. You can work with breakpoints (for example, examine existing breakpoints or set new breakpoints) as you would for a CU that has been loaded or run. Debug Tool applies these breakpoints when the CU becomes active. This feature is also available on WebSphere Developer for zSeries® and WebSphere Developer Debugger for zSeries.
- In Debug Tool Utilities, the layout of the panels and arrangement of parameters in the Create Private Message Regions function of the Manage IMS Programs section has been improved.
- You can specify which interface (Main Frame Interface (MFI), WebSphere Developer for zSeries, or WebSphere Developer Debugger for zSeries) to start when you want to debug a DB2 Stored Procedure, IMS Transaction Manager (TM), or batch program. Through an user exit, you can specify a TEST run-time option string that indicates which interface you want to start.
- You can use a utility, EQALANGP, to create a readable listing from a Fault Analyzer side file (IDILANGX or EQALANGX) or a SYSDEBUG file, which is generated by using the COBOL TEST(,,SEPARATE) compiler option. If you do not keep compiler listings in order to conserve DASD space, EQALANGP can help you create a compiler listing that resembles the original compiler listing.

Note: EQALANGP, which is shipped as a component of Debug Tool Utilities and Advanced Functions, is functionally equivalent to the IDILANGP program shipped as a component of Fault Analyzer for z/OS.
- The Debug Tool Coverage Utility (DTCU) SVC installer has been updated to help you ensure that the SVC numbers that you choose to use for the DTCU breakpoint SVCs do not conflict with the SVC numbers chosen for another program.
- The Coverage Utility Annotated Listing report is updated for COBOL programs so that you can add an HTML version of the report. The HTML version contains colored lines that indicate statements that were not executed and recomputed statistics based on the annotations in the listing instead of the raw coverage...
data. In addition, a new HTML Targeted Coverage Report, which contains an Annotated Listing with lines that were changed between two versions of source files, is available for COBOL programs

- You can use a new command, SET LIST TABULAR, to indicate how you want the output of the LIST command displayed. This helps you format the display so it matches the display of the MONITOR LIST command.

- You can use a new command, DESCRIBE LOADMODS, to indicate that you want to display information about all load modules or a specific load module, which are known to Debug Tool. Debug Tool displays information about where the load module or load modules are loaded from, and the size, the name, and the programs and CSECTs that are contained in a load module if information on a specific load module is requested.

- Enhancements have been added to better integrate Debug Tool Utilities and Advanced Functions with WebSphere Developer for zSeries and WebSphere Developer Debugger for zSeries.
Chapter 1. Debug Tool run-time options

This chapter describes the run-time options that you can use to control the operation of Debug Tool. For more information about these options, refer to the z/OS Language Environment Programming Reference.

You can provide these options in any of the following ways:

- Specify them when you start a Language Environment program
- Specify them when you start the Debug Tool non-Language Environment debugger (EQANMDBG)
- Pass them as run-time options by using the CICS DTCN or CADP profile definitions

In the first case they are the standard Language Environment run-time options. Otherwise they are Debug Tool keyword options with the same syntax and semantics as the corresponding Language Environment option. In all cases, you can omit these options if the default values are acceptable.

When you specify these options for a Language Environment program, they are handled by Language Environment and the following rules apply:

- You can mix them with other Language Environment run-time options in any order.
- Separate them with either blanks or commas.
- Separate all run-time options from user-program options with a slash ("/").
- The placement of these options (before or after the slash) depends on the programming language of the MAIN routine.
- For DB2 stored procedures of type MAIN, IMS TM programs, or batch programs, you can link with the Debug Tool Language Environment user exit, and can specify these options in a TEST run-time option data set.

When you specify these options for a non-Language Environment program by using EQANMDBG under z/OS batch or TSO, Debug Tool processes the options and the following rules apply:

- You must specify the name of the program to be debugged as the first parameter; this is a positional parameter.
- Specify the run-time options in any order following the name of the program to be debugged.
- Separate all options with commas.
- Separate the run-time options from user-program options with a slash ("/"). If you do not specify any run-time options, the slash follows the name of the program.
- Specify any parameters to the user-program after the slash.
- If no user-program parameters are required, you can omit the slash.

Refer to the following sections for more information related to the material discussed in this section.

Related tasks
Starting non-Language Environment Debug Tool under MVS™ batch or TSO

Debug Tool User’s Guide
Non-Language Environment positional parameter

If you use EQANMDBG to start Debug Tool to debug MVS batch or TSO programs that do not run in Language Environment, the first positional parameter must be the name of the program you want to debug. This name must be immediately followed by one of the following options:
- one or more of the Debug Tool keyword run-time options described in the following sections of this chapter and then a slash ('/') and any user-program parameters
- a slash ('/') and any user-program parameters

If no user-program parameters are required, the slash is optional.

There are other methods of starting Debug Tool to debug MVS batch or TSO programs that do not run in Language Environment. See Debug Tool User's Guide for a description of these other methods.

COUNTRY run-time option

Use the COUNTRY option to specify the country code to be used by Debug Tool. The default is always US.

The syntax for this option is:

```
--COUNTRY--(country_code)
```

country_code
A valid country code, one of:
- US United States of America
- JP Japan

NATLANG run-time option

Use the NATLANG option to specify the desired national language for Debug Tool. This determines the language that is used to display Debug Tool output, such as messages. If you do not specify NATLANG, the installation default is used.

The syntax for this option is:

```
--NATLANG--(language_Id)
```

language_Id
A valid national language identifier, one of:
- ENU English
- UEN Upper-case English
- JPN Japanese
- KOR Korean
**TEST run-time option**

The TEST run-time option gives control of your program to Debug Tool.

This section describes the TEST run-time option. For a description of how to use the TEST run-time option, refer to the [Debug Tool User's Guide](#).

The suboptions of the TEST run-time option control how, when, and where Debug Tool gains control of your program.

**Syntax of the TEST run-time option**

You can combine any of the suboptions for the TEST run-time option but only in the order specified by the TEST syntax. Any option or suboption referred to as “default” is the IBM-supplied default, and might have been changed by your system administrator during installation.

The syntax for this option is:

```
NOTEST
TEST
(test_level, commands_file)

prompt_levelpreferences_file)
```

**test_level:**

```
ALL
ERROR
NONE
```

**commands_file:**

```
commands_file_designator
VADSCPnnnnn
```

**prompt_level:**

```
PROMPT
NOPROMPT
```

Chapter 1. Debug Tool run-time options
preferences_file:

MFI

%terminal_id

VTAM_LU_id

network_identifier

VTAM

%user_id

(1)

%8000

TCPIP

tcpip_workstation_id

(1)

%port_id

VADTCP/IP

/SM590000

/SM590000

INSPPREF

preferences_file_designator

Notes:

1 Specifies remote debug mode.

The following list explains what actions are taken by each option and suboption.

NOTEST

Specifies that Debug Tool is not started at program initialization. However, starting Debug Tool is still possible through the use of CEETEST, PLITEST, or the __ctest() function. In such a case, the suboptions specified with NOTEST are used when Debug Tool is started.

TEST

Specifies that Debug Tool is given control according to the specified suboptions. The TEST suboptions supplied are used if Debug Tool is started with CEETEST, PLITEST, or __ctest().

If Debug Tool is started by using CALL CEETEST (or an equivalent entry), you cannot debug higher-level non-Language Environment programs or intercept non-Language Environment events that occur in higher-level programs after you return from the program that started Debug Tool.

test_level:

ALL (or blank)

Specifies that the occurrence of an attention interrupt, termination of your program (either normally or through an ABEND), or any program or Language Environment condition of Severity 1 and above causes Debug Tool to gain control, regardless of whether a breakpoint is defined for that type of condition. If a condition occurs and a breakpoint exists for the condition, the commands specified in the breakpoint are executed. If a condition occurs and a breakpoint does not exist for that condition, or if an attention interrupt occurs, Debug Tool does the following:

• In full-screen mode, Debug Tool reads commands from a commands file (if it exists and is available) or prompts you for commands.
• In batch mode, Debug Tool reads commands from the commands file. If none is available, the program runs uninterrupted.
ERROR
Specifies that only the following conditions cause Debug Tool to gain control
without a user-defined breakpoint.

- For C and C++:
  - An attention interrupt
  - Program termination
  - A predefined Language Environment condition of Severity 2 or above
  - Any C and C++ condition other than SIGUSR1, SIGUSR2, SIGINT or SIGTERM.
- For COBOL:
  - An attention interrupt
  - Program termination
  - A predefined Language Environment condition of Severity 2 or above.
- For PL/I:
  - An attention interrupt
  - Program termination
  - A predefined Language Environment condition of Severity 2 or above.

If a breakpoint exists for one of the above conditions, commands specified in
the breakpoint are executed. If no commands are specified, Debug Tool reads
commands from a commands file or prompts you for them in interactive
mode.

NONE
Specifies that Debug Tool gains control from a condition only if a breakpoint is
defined for that condition. If a breakpoint exists for the condition, the
commands specified in the breakpoint are executed. An attention interrupt
does not cause Debug Tool to gain control unless Debug Tool was started. To
change the TEST level after you start your debug session, use the SET TEST
command.

commands_file:
* (or blank)
Indicates that no commands file is supplied. The terminal, if available, is used
as the source of Debug Tool commands.

commands_file_designator
Valid designation for the primary commands file. A commands file is used
instead of the terminal as the initial source of commands, and only after the
preferences file, if specified, is processed.

The designation can be either a DD name or a data set name. Debug Tool uses
the following procedure to determine if the designation is a DD name or data
set name:

- If the designation does not contain periods (), Debug Tool considers it a DD
  name.
- Otherwise, if you are running under CICS, Debug Tool considers it a
  fully-qualified data set name.
- Otherwise, Debug Tool considers it a partially-qualified data set name and
  prefixes it with the user ID to form the fully-qualified data set name. If you
  want Debug Tool to interpret the data set name as a fully-qualified name,
  put a minus sign (-) in front of the name. In this case, Debug Tool will not
  append the user ID to the data set name.

If the designation contains non-alphanumeric characters (for example, a
parenthesis), the designation must be enclosed in either single or double
quotes. However, when a data set name is enclosed in quotes, Debug Tool still
considers the data set name a partially-qualified data set name and prefixes the user ID to form the fully-qualified data set name.

The commands_file_designator has a maximum length of 80 characters.

If the specified DD name is longer than eight characters, it is automatically truncated. No error message is issued.

The primary commands file is required when you debug in batch mode. Debug Tool reads and executes commands listed in the commands file until the file runs out of commands or the program finishes running. You can use a log file from one Debug Tool session as the commands file for a subsequent Debug Tool session.

The primary commands file is shared across multiple enclaves.

Commands file is not supported in remote debug mode.

**VADSCPnnnnn**

Specifies a CCSID (Coded Character Set Identifiers) to use when you are debugging programs that contain NLS characters in remote debug mode. Consult with your system programmer to determine what values of VADSCPnnnnn are supported on your system. See [Debug Tool Customization Guide](#) for more information.

The following examples show how to use VADSCPnnnnn:

- For Japanese EBCDIC CCSID 930
  
  TEST(ALL,VADSCP930,,TCPIP&9.10.11.12%8001:*)

- For Japanese EBCDIC CCSID 939
  
  TEST(ALL,VADSCP939,,TCPIP&9.10.11.12%8001:*)

- For German EBCDIC CCSID 1141
  
  TEST(ALL,VADSCP1141,,TCPIP&9.10.11.12%8001:*)

- For Korean EBCDIC CCSID 933
  
  TEST(ALL,VADSCP933,,TCPIP&9.10.11.12%8001:*)

**prompt_level:**

**PROMPT (or ; or blank)**

Indicates that you want Debug Tool started immediately after Language Environment initialization. Commands are read from the preferences file and then any designated primary commands file. If neither file exists, commands are read from your terminal or workstation.

**NOPROMPT (or *)**

Indicates that you do not want Debug Tool started immediately after Language Environment initialization. Instead, your application begins running. When Debug Tool is running without the Language Environment run time (started by using EQANMDBG), the NOPROMPT option is ignored; PROMPT is always in effect.

If you specify the NOPROMPT suboption, you cannot debug higher-level non-Language Environment programs or intercept non-Language Environment events that occur in higher-level programs after you return from the program that started Debug Tool.

**command**

One or more valid Debug Tool commands. Debug Tool is started immediately
after program initialization, and then the command (or command string) is executed. The command string can have a maximum length of 250 characters, and must be enclosed in double quotation marks. Multiple commands must be separated by a semicolon.

If you include a STEP command or GO command in your command string, none of the subsequent commands are processed. The use of a command in prompt_level is not supported in remote debug mode.

preferences_file:

MFI (Main Frame Interface)
  Specifies Debug Tool should be started in full-screen mode for your debug sessions.

terminal_id (CICS only)
  Specifies up to a four-character terminal id to receive Debug Tool screen output during dual terminal session. The corresponding terminal should be in service and acquired, ready to receive Debug Tool-related I/O.

network_identifier (full-screen mode through a VTAM terminal only)
  Specifies an optional 1-8 character network name that identifies the network in which the partner LU, identified by the VTAM_LU_id parameter, resides.

VTAM_LU_id (full-screen mode through a VTAM terminal only)
  Specifies up to an eight-character VTAM logical unit (LU) identifier for a terminal used in full-screen mode through a VTAM terminal. The VTAM_LU_id parameter cannot be used to debug CICS applications. Contact your system programmer to determine how to access this type of terminal LU at your site. See Debug Tool User's Guide for information on how to use this terminal.

VTAM (full-screen mode through a VTAM terminal using the Debug Tool Terminal Interface Manager only)
  Specifies Debug Tool should be started in full-screen mode through a VTAM terminal for your debug sessions and that you have used the Debug Tool Terminal Interface Manager to assign a user ID to the terminal.

user_id (full-screen mode through a VTAM terminal using the Debug Tool Terminal Interface Manager only)
  Specifies the user ID that you used to log on to the Debug Tool Terminal Interface Manager. See the entry for VTAM_LU_id for more information.

INSPREF (or blank)
  Debug Tool-supplied default preferences file DD name. Any preferences file that is specified to Debug Tool becomes the first source of Debug Tool commands after Debug Tool is started. It is often used to set up the Debug Tool environment.

preferences_file_designator
  A valid DD name or data set designation specifying the preferences file to use.

This file is read the first time Debug Tool is started and must contain a sequence of Debug Tool commands to be executed.

The designation can be either a DD name or a data set name. Debug Tool uses the following procedure to determine if the designation is a DD name or data set name:

- If the designation does not contain periods (.), Debug Tool considers it a DD name.
- Otherwise, if you are running under CICS, Debug Tool considers it a fully-qualified data set name.
• Otherwise, Debug Tool considers it a partially-qualified data set name and prefixes it with the user ID to form the fully-qualified data set name. If you want Debug Tool to interpret the data set name as a fully-qualified name, put a minus sign (-) in front of the name. In this case, Debug Tool will not append the user ID to the data set name.

If the designation contains non-alphanumeric characters (for example, a parenthesis), the designation must be enclosed in either single or double quotes. However, when a data set name is enclosed in quotes, Debug Tool still considers the data set name a partially-qualified data set name and prefixes the user ID to form the fully-qualified data set name.

* Specifies that no preferences file is supplied.

INSPPREF and preferences_file_designator are not supported when using remote debug mode. * is always assumed.

The following TEST suboptions are for use only in remote debug mode:

**TCP/IP** (remote debug mode only)

Specifies that Debug Tool is using a single-socket connection to interface with one of the following remote debuggers:

• IBM Distributed Debugger
• Compiled Language Debugger component of WebSphere Studio Enterprise Developer
• Compiled Language Debugger component of WebSphere Developer for zSeries
• WebSphere Developer Debugger for zSeries

This connection type is recommended because it uses less resource and works better with workstations that are running firewall software. However, if you use a version of the IBM Distributed Debugger that is earlier than version 9.2 (copyright date 10/19/2003), you must specify a multiple-socket connection (VADTCP/IP&).

**VADTCP/IP** (remote debug mode only)

Specifies that Debug Tool is using a multiple-socket connection to interface with the IBM Distributed Debugger. You must use this connection type if you use a version of the IBM Distributed Debugger that is earlier than version 9.2 (copyright date 10/19/2003).

*tcpip_workstation_id* (remote debug mode only)

TCP/IP name or address of the workstation where the remote debug daemon is executing. The name can be specified as a symbolic TCP/IP address, such as some.name.com. The address can also be specified as a numeric TCP/IP address, such as 9.112.26.333.

*%8000* (remote debug mode only)

Default *port_id*. If this suboption is omitted, Debug Tool uses 8000 as the port ID.

*%port_id* (remote debug mode only)

Specifies a unique TCP/IP port on your workstation that is used by the remote debug daemon. The following remote debuggers use 8001 as the default TCP/IP port ID:

• Compiled Language Debugger component of WebSphere Studio Enterprise Developer
• Compiled Language Debugger component of WebSphere Developer for zSeries
• WebSphere Developer Debugger for zSeries

If you did not change the default TCP/IP port settings used by these remote debuggers, you must specify %8001 as the port ID in your TEST run-time options string. For example: TEST(ALL,'*',PROMPT,'TCPIP&9.112.26.333 %8001: ')

Usage notes
• If the VADSCP suboption is not specified, the default code page (00037) is used for the debug session.
• If the code page is not specified correctly or the conversion images are not available in the system, the default code page (00037) is used for the debug session.
• If the code page is specified correctly and the conversion images are available in the system, but the string conversion is not successful, default code page (00037) is used for this conversion.

Refer to the following sections for more information related to the material discussed in this section.

Related references
- z/OS Language Environment Debugging Guide

Related tasks
- Debug Tool User’s Guide

TRAP run-time option

Use the TRAP option to specify how Debug Tool handles ABENDs and program interrupts.

The syntax for this option is:

```
TRAP(ON|OFF)
```

**ON**  Enable Debug Tool to trap ABENDs.

**OFF**  Prevent Debug Tool from trapping ABENDs; an ABEND causes abnormal termination of both Debug Tool and the program under test.
Chapter 2. Common syntax elements in Debug Tool commands

Several syntax elements are used in multiple Debug Tool commands. These elements are described in the following topics. Some of these syntax elements are generic and do not require a syntax diagram.

Refer to the following sections for more information related to the material discussed in this section.

Related references
- "block_name" on page 12
- "block_spec" on page 13
- "compile_unit_name" on page 13
- "cu_spec" on page 14
- "expression" on page 14
- "load_module_name" on page 14
- "load_spec" on page 14
- "references" on page 14
- "statement_id" on page 15
- "statement_id_range and stmt_id_spec" on page 16
- "statement_label" on page 17

block_name

A block_name identifies:
- A C and C++ function or a block statement
- A COBOL nested program or method contained within a complete COBOL program
- A PL/I block

The current block qualification can be changed by using the SET QUALIFY BLOCK command.

For C++ only:
- Include full declaration in block qualification.

For COBOL only:
- Enclose the block name in double (') or single (') quotes if it is case sensitive. If the name is not inside quotes, Debug Tool will convert the name to uppercase.
- If a name contains an internal double quote, you should enclose the name in single quotes. Similarly, if the name contains an internal single quote, you should enclose the name in double quotes.

You can use block_name only for blocks known in the current enclave.

block_spec

A block_spec identifies a block in the program being debugged.
block_name

Name of the block. See “block_name” on page 11.

%BLOCK

Represents the currently qualified block. See Chapter 7, “Debug Tool
variables,” on page 245.

cu_spec

A valid compile unit specification; see “cu_spec” on page 13.

You can use block_name only for blocks known in the current enclave.

For C++ only:

Block_spec must include the formal parameters for the function. The correct
block qualification is:

```c
int function(int, int) is function(int, int)
```

Use Describe CUS to determine correct block_spec for blocks known in the
current enclave.

Refer to the following sections for more information related to the material
discussed in this section.

Related references

- “block_name” on page 11
- Chapter 7, “Debug Tool variables,” on page 245
- “cu_spec” on page 13

condition

A simple relational condition. Particular rules for forming the relational condition
depend on the current programming language setting.

Refer to the following sections for more information related to the material
discussed in this section.

Related references

- “Allowable comparisons for the IF command (COBOL)” on page 116

compile_unit_name

A compile_unit_name identifies any of the following items:

- An assembler CSECT name
- A C or C++ source file
- An OS/VS COBOL program
- A COBOL program or class
- The external procedure name of a PL/I for MVS program
- The package statement or the name of the main procedure, for an Enterprise
  PL/I program compiled with the following compiler and running in the
  following environment, :
  
  - Enterprise PL/I Version 3.5 with the PTFs for APARs PK35230 and PK35489
    applied
Language Environment Version 1.4 through 1.8 with the PTF for APAR PK33738 applied

The name of the source file, for an Enterprise PL/I program compiled with a compiler earlier than Enterprise PL/I for z/OS, Version 3.5, with the PTFs for APARs PK35230 and PK35489 applied.

**For C and C++ only:**

The compile unit name must always be enclosed in double quotes ("). For example, the following statement is ambiguous because the compile unit and a function in that compile unit have the same name:

```
LIST CU2::CU2::var1
```

To avoid the ambiguity, use the following statement to list the value of the variable `var1` correctly scoped to the function `CU2`:

```
LIST "CU2"::CU2::var1
```

Escape sequences in compile unit names that are specified as strings are not processed if the string is part of a qualification statement.

**For COBOL only:**

Enclose the compile unit name in double (" or single (’) quotes if it is case sensitive. If the name is not inside quotes, Debug Tool will convert the name to uppercase.

**For Enterprise PL/I only:**

The compile unit name must be enclosed in single quotes (’). If your program was compiled with the following compiler and is running in the following environment, you do not need to enclose the compile unit name in quotes:

- Enterprise PL/I for z/OS, Version 3.5, with the PTFs for APARs PK35230 and PK35489 applied
- Language Environment Version 1.4 through 1.8 with the PTF for APAR PK33738 applied

If the compile unit name is not a valid identifier in the current programming language, it must be entered as a character string constant in the current programming language.

The current compile unit qualification can be changed using the `SET QUALIFY CU` command.

---

**cu_spec**

A `cu_spec` identifies a compile unit in the application being debugged. In PL/I, the compile unit name is the same as the outermost procedure name in the program.

```
| load_spec  ::=  > |
| %CU        |
| %PROGRAM   |
|
```

If `cu_spec` is omitted, the current load module qualification is used.

**compile_unit_name**

The name of the compile unit, depending on the programming language. See "`compile_unit_name` on page 12"
**load_spec**

The name of the load module. See "load_spec" on page 15.

**%CU**

Represents the currently qualified compile unit. %CU is equivalent to %PROGRAM.

**%PROGRAM**

Is equivalent to %CU.

You can use cu_spec to specify compile units only in an enclave that is currently running. Therefore, you can qualify only variable names, function names, labels, and statement_ids to blocks within compile units in the current enclave.

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

- "load_spec" on page 15
- "compile unit name" on page 12
- Chapter 7, "Debug Tool variables," on page 245

---

**expression**

An expression is a combination of references and operators that result in a value. For example, it can be a single constant, a program, session, or Debug Tool variable, a built-in function reference, or a combination of constants, variables, and built-in function references, or operators and punctuation (such as parentheses).

Particular rules for forming an expression depend on the current programming language setting and what release level of the language run-time library under which Debug Tool is running. For example, if you upgrade your version of the HLL compiler without upgrading your version of Debug Tool, certain application programming interface inconsistencies might exist.

You can use expressions for only variables contained in the current enclave.

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

- "references" on page 15

---

**load_module_name**

A load_module_name is the name of a file, object, or dynamic link library (DLL) that has been loaded by a supported HLL load service or a subsystem. For example, an enclave can contain load modules, which in turn contain compile units.

For C, escape sequences in load module names that are specified as strings are not processed if the string is part of a qualification statement.

If the load_module_name is omitted from a name that allows it as a qualifier, the current load module qualification is assumed. The load_module_name can be changed by using the SET QUALIFY LOAD command.

If two enclaves contain duplicate modules, references to compile units in the modules will be ambiguous, and will be flagged as errors. However, if the compile
unit is in the currently executing load module, that load module is assumed and
no check for ambiguity will be performed. Therefore, for Debug Tool, load module
names must be unique.

**load_spec**

A *load_spec* identifies a load module in the program being debugged.

```plaintext
load_module_name

%LOAD
```

The *load_spec* can be specified as a string constant in the current programming
language, for example, a string literal in C or a character literal in COBOL. If not
specified as such, it must be a valid identifier in the current programming
language.

*load_module_name*

Name of a file, object, or Dynamic Link Library (DLL) that has been loaded by
a supported HLL load service, or a subsystem. See “*load_module_name*” on
[page 14].

%LOAD

Represents the currently qualified load module.

Refer to the following sections for more information related to the material
discussed in this section.

**Related references**

“*load_module_name*” on page 14
Chapter 7, “Debug Tool variables,” on page 245

---

**references**

A *reference* is a subset of an *expression* that resolves to an area of storage, that is, a
possible target of an assignment statement. For example, it can be a program,
session, or Debug Tool variable, an array or array element, or a structure or
structure element, and any of these can be pointer-qualified (in programming
languages that allow it). Any identifying name in a reference can be optionally
qualified by containing structure names and names of blocks where the item is
visible. It is optionally followed by subscript and substring modifiers, following
the rules of the current programming language.

The specification of a qualified reference includes all containing structures and
blocks as qualifiers, and can optionally begin with a load module name qualifier.
For example, when the current programming language setting is C,
mod::cu::proc::struc1.struc2.array[23].

When the current programming language setting is C and C++, the term lvalue is
used in place of reference.

If you are debugging a program that was compiled with an Enterprise PL/I
compiler, Debug Tool does not support the use of a qualified reference that
includes block_spec, cu_spec, or load_spec. If you are debugging a program compiled
with the following compiler and running in the following environment, Debug
Tool does support the use of a qualified reference that includes block_spec, cu_spec,
or load_spec:
• Enterprise PL/I Version 3.5 with the PTFs for APARs PK35230 and PK35489 applied
• Language Environment Version 1.4 through 1.8 with the PTF for APAR PK33738 applied

If you are debugging a program that was compiled with an Enterprise PL/I compiler and Debug Tool is at an entry to a block, you can not list or reference any variable or expression that includes variables declared in the block being entered.

A COBOL reference can be a data name, which can be any of the following, according to the rules of the COBOL language:
• qualified
• subscripted
• indexed
• reference modified

A COBOL reference can be to any special register, except for the following special registers:
• ADDRESS-OF
• LENGTH-OF
• WHEN-COMPILED

Particular rules for forming a reference depend on the current programming language setting and what release level of the language run-time library Debug Tool is running under. For example, if you upgrade your version of the HLL compiler without upgrading your version of Debug Tool, certain application programming interface inconsistencies might exist.

**statement_id**

A *statement_id* identifies an executable statement in a manner appropriate for the current programming language. This can be a statement number, sequence number, or source line number. The statement id is an `integer` or `integer.integer` (where the first `integer` is the line number and the second `integer` is the relative statement number). For example, you can specify 3, 3.0, or 3.1 to signify the first relative statement on line 3. C, C++, COBOL, and PL/I allow multiple statements or verbs within a source line.

You can only use statement identifiers for statements that are known in the current enclave.

**statement_id_range and stmt_id_spec**

A *statement_id_range* identifies a source statement id or range of statement ids. *Stmt_id_spec* identifies a statement id specification.
stmt_id_spec:

- block_spec
- cu_spec

%LINE
%STATEMENT

block_spec
A valid block specification. The default is the currently qualified block. For the currently supported programming languages, block qualification is extraneous because statement identifiers are unique within a compile unit. Therefore, block qualification is ignored.

cu_spec
A valid compile unit specification; see “cu_spec” on page 13. The default is the currently qualified compile unit.

statement_id
A valid statement identifier number; see “statement_id” on page 16.

%LINE
Represents the currently suspended source statement or line. See Chapter 7, “Debug Tool variables,” on page 245. %LINE is equivalent to %STATEMENT.

%STATEMENT
Is equivalent to %LINE.

Specifying a range of statements
A range of statements can be identified by specifying a beginning and ending statement id, separated by a hyphen (-). When the current programming language setting is COBOL, blanks are required around the hyphen (-). Blanks are optional for C and C++ and PL/I. Both statement ids must be in the same block, the second statement cannot occur before the first in the source program, and they cannot be equal.

A single statement id is also an acceptable statement id range and is considered to begin and end at the same statement. A single statement id range consists of only one statement or verb even in a multistatement line.

Refer to the following sections for more information related to the material discussed in this section.

Related references
“block_spec” on page 11
“cu_spec” on page 13
“statement_id” on page 16
Chapter 7, “Debug Tool variables,” on page 245

statement_label
A statement_label identifies a statement using its source label. The specification of a qualified statement label includes all containing compile unit names or block names, and can optionally begin with a load module name qualifier. For example:

mod::>proc1::proc2>::block1::start

The form of a label depends on the current programming language:
- In C and C++, labels must be valid identifiers.
- In COBOL, labels must be valid identifiers and can be qualified with the section name.
- In PL/I, labels must be valid identifiers, which can include a label variable.

You can only use statement labels for labels that are known in the current enclave.
Chapter 3. Syntax for assembler and disassembly expressions

Use the syntax defined in this section to write expressions for Debug Tool commands while you debug an assembler or disassembly program.

Assembler expressions can be written in the following forms:

- A standard assembler expression with an implied length. The following are three examples:
  - X
  - 133
  - X+15
- A standard assembler expression without an implied length. Expressions can be written in this form only if the length can be specified or derived from an operand. For example: R3->+X'2C'
- A conditional assembler expression which is written with conditional operators and can be used only as the operand of an IF command. For example: X+1=Y & Z=4

Common syntax elements

You can use the following syntax elements to write an assembler expression:

\[ \text{dd}d \]
A decimal constant, where \text{dd}d are valid decimal digits. For example: 145

\['X\text{xxxx}' or 'X''xxxx'\]
A hexadecimal constant, where \text{xxxx} are valid hexadecimal digits.
Examples: X'1F4C' or X"1F4C"
If this constant is from 1 to 4 bytes in length, it can be used in arithmetic or string contexts. Otherwise, it can only be used in string contexts.

\['C\text{cccc}', 'cccc', or "cccc"\]
A character constant. For example: C'F$3' or "F$3"
If this constant is from 1 to 4 bytes in length, it can be used in arithmetic or string contexts. Otherwise, it can only be used in string contexts.

\text{symbol} A valid symbol used in the assembler source program. Examples:
lastName, UserVar8
If a symbol is defined by using the EQU instruction and the first usage of the symbol is as a register, the symbol is associated with that register. If you define a symbol with the intent to use the symbol as a register but you never reference the symbol or the first reference to the symbol is not as a register, Debug Tool defines the symbol as a constant, not as a register. For example, if you define the symbol R7 by using the instruction R7 EQU 7 and you never reference R7 or the first reference is not as a register, Debug Tool defines the symbol R7 as the constant 7, not as register R7.

Debug Tool implicitly defines the following symbols in all disassembly compilation units and in any assembler compilation units where the symbol is not already defined:
- R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15.
  These symbols are implicitly defined as Debug Tool general purpose
registers. For example, R0 is defined as %R0. If you are debugging an assembler compilation unit that defines the symbol R0 and R0 is not used as a register, you can use the %R0 variable to reference general purpose registers R0.

- _STORAGE. This symbol is implicitly defined as a symbol representing all of main memory. You can reference any area of memory by using the _STORAGE symbol with the substring notation defined in “Operators that can be used in any expression.” For example, _STORAGE(X’1FF3C’:4) references the four bytes of storage at address X’1FF3C’. The substring notation used by the _STORAGE symbol specifies an actual address; therefore, to reference the first byte of storage, use a 0 instead of a 1 in the substring notation.

%symbol
A valid Debug Tool variable. For example: %ADDRESS

Operators
You can use the operators defined in this section to write assembler expression and conditional assembler expressions.

Operators that can be used in any expression
Use the operators defined in this section to write assembler expressions.

+ Addition
– Subtraction or prefix minus
* Multiplication
/ Division
// Remainder
|| Concatenation (C and X-type operands only)
& Bitwise AND
| Bitwise OR
(…) Parenthesis to control the order of operation, specify the subscript of an array, or select a substring.

  symbol(subscript)
  Parenthesis to specify a subscript for an array. For example, if an array is defined by the instruction X DS 5F, you can specify the first word in the array as X(1).

  symbol(substring)
  Parenthesis to select a substring of a single byte from a character or hexadecimal variable

  symbol(substrstart:substrend)
  Parenthesis to select a substring of the bytes from substrstart to substrend from a character or hexadecimal variable

  symbol(substrstart::substrlen)
  Parenthesis to select a substring of substrlen bytes beginning at substrstart from a character or hexadecimal variable
For an array of character or hexadecimal strings, these forms can be combined by using symbol(subscript,substring), symbol(subscript,substrstart:substrend), or symbol(subscript,substrstart::substrlen).

\[ ->, \Rightarrow, \text{or} \; \Rightarrow \]

Indirection operator. You can use an indirection operator as follows:

\[
\text{operand1}<\text{indirection_operator}>\text{operand2}
\]

Use the contents of operand1 as the base address of the DSECT which contains operand2. For example, R1->DCBDDNAME instructs Debug Tool to use the contents of register 1 as the base address of the DSECT which contains DCBDDNAME.

\[
\text{operand1}<\text{indirection_operator}> \text{or} \; \text{operand2}<\text{indirection_operator}>+\text{operand2}
\]

Use the contents of operand1 as an address. If the <indirection_operator> is followed by a plus sign (+), use operand2 as an offset. For example, X-> instructs Debug Tool to use the contents of X as the address of the storage. For a second example, R3->+X'22' instructs Debug Tool to use the contents of register 3 and add hexadecimal 22 (the offset) to determine the address of storage.

If the indirection operator is not followed by a symbol, no length is implied. This form can be used only in situations where the length can be determined by another operand. For example, the command STORAGE(R10->,4)=22 provides the length in the second operand of the STORAGE command.

The three indirection operators indicate which address specification to use, as follows:

\[ -> \quad \text{Use the current Amode specification.} \]
\[ \Rightarrow \quad \text{Use a 31-bit address specification.} \]
\[ \Rightarrow \quad \text{Use a 24-bit address specification.} \]

\[ . \]

Dot operator (period). You can use a dot operator to qualify a name in a DSECT by the name on a labeled USING statement. The dot operator must be immediately preceded by a label from a previous labeled USING statement and must be immediately followed by a name defined in a DSECT.

\[ \text{ADDR'} \]

Returns the address of a symbol. If the operand of ADDR' is a symbol that is known in the current CU but resides in another CSECT, the ADDR' function returns 0. For example, ADDR'ABC returns the address of symbol ABC.

\[ \text{L'} \]

Returns the length of a symbol. For example, L'ABC returns the length of the symbol ABC.

**Operators that can be used only in conditional expressions**

The following operators can be used only in conditional expressions (for example, the IF command):

\[ = \quad \text{Compare the two operands for equality.} \]
\[ \sim = \quad \text{Compare the two operands for inequality.} \]
\[ < \quad \text{Determines whether the left operand is less than the right operand.} \]
> Determines whether the left operand is greater than the right operand.

<= Determines whether the left operand is less than or equal to the right operand.

>= Determines whether the left operand is greater than or equal to the right operand.

& Logical “and” operation.

| Logical “or” operation.
Chapter 4. Syntax for OS/VS COBOL expressions

You can use the syntax defined in this section to write expressions for Debug Tool commands while you debug OS/VS COBOL programs.

In general, whenever you enter an OS/VS COBOL expression (such as the operand of LIST expression, Assignment, IF, etc.), you must enclose the OS/VS COBOL expression in single-quotes. For example:

LIST 'A-B IN C';
'A' = 'B';
IF 'A = 22' THEN...

There are some Debug Tool commands that can be used for debugging OS/VS COBOL programs that use the assembler syntax. A note to this effect is found in the section describing each of these commands. For example, while debugging an OS/VS COBOL program you might use the following command:

STORAGE(X"1B4C0",3) = X"0102FC";

Restrictions on OS/VS COBOL expressions

In addition to the requirement that OS/VS COBOL expressions be enclosed in single-quotes, the following restrictions apply to OS/VS COBOL expressions:

- The following operators are supported by Debug Tool in OS/VS COBOL expressions:
  - IN or OF
  - Subscript / index
  - LENGTH OF
  - +, -, *, /
  - // (remainder)
  - || (concatenation)
  - ( )
- In a subscript or index list, the subscript or index expressions must be separated by a comma. A space is not sufficient for separating subscript or index expressions.
- Lower-case letters are accepted in contexts other than non-numeric literals as a substitute for (and equivalent to) upper case letters.
- The use of COBOL special registers such as DAY, DATE, TIME, etc. is not supported in Debug Tool OS/VS COBOL expressions.
- All non-numeric literals must be enclosed in double quotes. Single quotes cannot be used.
- You cannot use the LIST command or alter level 88 variables in OS/VS COBOL.
- Only the following subset of figurative constants are supported in Debug Tool OS/VS COBOL expressions:
  - HIGH-VALUE, HIGH-VALUES
  - LOW-VALUE, LOW-VALUES
  - QUOTE, QUOTES
  - SPACE, SPACES
  - ZERO, ZEROES, ZEROS
Common syntax elements

You can use the following syntax elements to write an OS/VS COBOL expression:

**ddd or ddd.ddd**
A decimal constant, where ddd are valid decimal digits. For example: 145 or 12.72.

**X”xxxx”**
A hexadecimal constant, where xxxx are valid hexadecimal digits. For example: 
X"1F4C"

"cccc"
A non-numeric literal. For example:
"F$3"

**symbol**
A valid symbol used in the OS/VS COBOL source program. Examples:
LASTNAME
USERVAR8
12CENTS

Debug Tool implicitly defines the _STORAGE symbol in all OS/VS COBOL programs as a symbol representing all of main memory. You can reference any area of memory by using the _STORAGE symbol with the substring notation defined in "Operators that can be used in any expression." For example, _STORAGE(X"1FF3C":4) references the four bytes of storage at address X"1FF3C". The substring notation used by the _STORAGE symbol specifies an actual address; therefore, to reference the first byte of storage, use a 0 instead of a 1 in the substring notation.

**%symbol**
A valid Debug Tool variable or built-in function. For example:
%ADDRESS
%HEx(expression)

Operators

You can use the operators defined in this section to write OS/VS COBOL expressions and conditional OS/VS COBOL expressions.

**Operators that can be used in any expression**

Use the operators defined in this section to write OS/VS COBOL expressions.

+  Addition
−  Subtraction or prefix minus
*  Multiplication
/  Division
// Remainder
|| Concatenation (non-arithmetic operands only)
(...) Parenthesis to control the order of operation, specify the subscript of an array, or select a substring,
symbol(subscript,subscript,...)  
Parenthesis to specify a subscript or index for an array. Note that commas are required between subscript or index values. Blanks alone are not acceptable.

symbol(substrstart:substrend)  
Parenthesis to select a substring of the bytes from substrstart to substrend from a character variable.

symbol(substrstart::substrlen)  
Parenthesis to select a substring of substrlen bytes beginning at substrstart from a character variable.

For an array of character strings, these forms can be combined by using
symbol(subscript,substrstart:substrend), or
symbol(subscript,substrstart::substrlen).

LENGTH OF  
Returns the length of a symbol. For example, LENGTH OF ABC returns the length of the symbol ABC.

Operators that can be used only in conditional expressions  
The following operators can be used only in conditional expressions (for example, the IF command):
  =  Compare the two operands for equality.
  ^= Compare the two operands for inequality.
  <  Determines whether the left operand is less than the right operand.
  >  Determines whether the left operand is greater than the right operand.
  <= Determines whether the left operand is less than or equal to the right operand.
  >= Determines whether the left operand is greater than or equal to the right operand.
  &  Logical "and" operation.
  |  Logical "or" operation.
Chapter 5. Debug Tool commands

Commands and keywords can be abbreviated. The abbreviations shown with some commands are the minimum abbreviations. However, you can use a minimum abbreviation or any string from the minimum to completely spelling out the keyword; all are valid. This is true of all keywords for commands.

If you are debugging in full-screen mode, you can get help with Debug Tool command syntax by either pressing PF1 or entering a question mark (?) on the command line. This lists all Debug Tool commands in the Log window.

To get a list of options for a command, enter a partial command followed by a question mark.

The table below summarizes the Debug Tool commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;? command&quot; on page 30</td>
<td>Displays all Debug Tool commands in the Log window.</td>
</tr>
<tr>
<td>&quot;ALLOCATE command&quot; on page 31</td>
<td>Allocates a file to an existing data set, a concatenation of existing data sets, or a temporary data set.</td>
</tr>
<tr>
<td>&quot;ANALYZE command (PL/I)&quot; on page 32</td>
<td>Displays the process of evaluating an expression and the data attributes of any intermediate results.</td>
</tr>
<tr>
<td>&quot;Assignment command (assembler and disassembly)&quot; on page 33</td>
<td>Assigns the value of an expression to a specified storage location or register.</td>
</tr>
<tr>
<td>&quot;Assignment command (OS/VS COBOL)&quot; on page 36</td>
<td>Assigns the value of an expression to a specified reference.</td>
</tr>
<tr>
<td>&quot;Assignment command (PL/I)&quot; on page 36</td>
<td>Assigns the value of an expression to a specified reference.</td>
</tr>
<tr>
<td>&quot;AT command&quot; on page 37</td>
<td>Defines a breakpoint (gives control of your program to Debug Tool under the specified circumstances).</td>
</tr>
<tr>
<td>&quot;BEGIN command (PL/I)&quot; on page 67</td>
<td>BEGIN and END delimit a sequence of one or more commands to form one longer command.</td>
</tr>
<tr>
<td>&quot;block command (C and C++)&quot; on page 68</td>
<td>Allows you to group any number of Debug Tool commands into one command.</td>
</tr>
<tr>
<td>&quot;break command (C and C++)&quot; on page 69</td>
<td>Allows you to terminate and exit a loop (that is, do, for, and while) or switch command from any point other than the logical end.</td>
</tr>
<tr>
<td>&quot;CALL command&quot; on page 69</td>
<td>The CALL command calls either a procedure, entry name, or program name, or it requests that a utility function be run.</td>
</tr>
<tr>
<td>&quot;CLEAR command&quot; on page 79</td>
<td>Removes the actions of previously issued Debug Tool commands (such as breakpoints).</td>
</tr>
<tr>
<td>&quot;COMMENT command&quot; on page 84</td>
<td>Used to insert commentary into the session log.</td>
</tr>
<tr>
<td>&quot;COMPUTE command (COBOL)&quot; on page 84</td>
<td>Assigns the value of an arithmetic expression to a specified reference.</td>
</tr>
<tr>
<td>&quot;CURSOR command (full-screen mode)&quot; on page 85</td>
<td>Moves the cursor between the last saved position on the Debug Tool session panel (excluding the header fields) and the command line.</td>
</tr>
<tr>
<td>Declaration/Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>DECLARE command (PL/I)</td>
<td>Declares session variables that are effective during a Debug Tool session.</td>
</tr>
<tr>
<td>DECLARE command (C and C++)</td>
<td>Declares session variables and tags that are effective during a Debug Tool session.</td>
</tr>
<tr>
<td>DECLARES (assembler, disassembly, and OS/VS COBOL) on page 89</td>
<td>Declares session variables that are effective during a Debug Tool session.</td>
</tr>
<tr>
<td>DECLARES (COBOL)</td>
<td>Declares session variables that are effective during a Debug Tool session.</td>
</tr>
<tr>
<td>DISABLE command</td>
<td>Makes the AT breakpoint inoperative, but does not clear it; you can ENABLE it later without typing the entire command again.</td>
</tr>
<tr>
<td>do/while command (C and C++)</td>
<td>Performs a command before evaluating the test expression.</td>
</tr>
<tr>
<td>Enable command</td>
<td>Allows one or more commands to be collected into a group which can (optionally) be run repeatedly.</td>
</tr>
<tr>
<td>EVALUATE command (COBOL)</td>
<td>Provides a shorthand notation for a series of nested IF statements.</td>
</tr>
<tr>
<td>Expression command (C and C++)</td>
<td>Evaluates the given expression which can be used to either assign a value to a variable or to call a function.</td>
</tr>
<tr>
<td>FIND command</td>
<td>Provides full-screen, line, and batch mode searching of source and listing files, and full-screen searching of Log and Monitor windows.</td>
</tr>
<tr>
<td>for command (C and C++)</td>
<td>Provides iterative looping.</td>
</tr>
<tr>
<td>FREE command</td>
<td>Frees (deallocates) an allocated file.</td>
</tr>
<tr>
<td>GO command</td>
<td>Causes Debug Tool to start or resume running your program.</td>
</tr>
<tr>
<td>GOTO command</td>
<td>Causes Debug Tool to resume program execution at the specified statement id.</td>
</tr>
<tr>
<td>GOTO LABEL command</td>
<td>Causes Debug Tool to resume running program at the specified statement label.</td>
</tr>
<tr>
<td>IF command (assembler, disassembly, and OS/VS COBOL)</td>
<td>Lets you conditionally perform a command.</td>
</tr>
<tr>
<td>IF command (C and C++)</td>
<td>Lets you conditionally perform a command.</td>
</tr>
<tr>
<td>IF command (COBOL)</td>
<td>Lets you conditionally perform a command.</td>
</tr>
<tr>
<td>IF command (PL/I)</td>
<td>Lets you conditionally perform a command.</td>
</tr>
<tr>
<td>IMMEDIATE command (full-screen mode)</td>
<td>Causes a command within a command list to be performed immediately. For use with commands assigned to a PF key.</td>
</tr>
<tr>
<td>INPUT command (C, C++, and COBOL)</td>
<td>Provides input for an intercepted read and is valid only when there is a read pending for an intercepted file.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>JUMPTO command</td>
<td>Jumps to the specified statement and then stops the program at that statement.</td>
</tr>
<tr>
<td>LIST command</td>
<td>Displays information about your Debug Tool session.</td>
</tr>
<tr>
<td>LOAD command</td>
<td>Specifies that the named module should be loaded for debugging purposes.</td>
</tr>
<tr>
<td>LOADDEBUGDATA</td>
<td>Specifies that a compile unit (CU) as an assembler CU and loads debug data.</td>
</tr>
<tr>
<td>MONITOR command</td>
<td>Defines or redefines a command whose output is displayed in the Monitor window (full-screen mode), terminal output (line mode), or log file (batch mode).</td>
</tr>
<tr>
<td>MOVE command (COBOL)</td>
<td>Transfers data from one area of storage to another.</td>
</tr>
<tr>
<td>Null command</td>
<td>A semicolon written where a command is expected.</td>
</tr>
<tr>
<td>ON command (PL/I)</td>
<td>Establishes the actions to be executed when the specified PL/I condition is raised.</td>
</tr>
<tr>
<td>PANEL command</td>
<td>Displays special panels (for example, to customize your full-screen session).</td>
</tr>
<tr>
<td>PERFORM command (COBOL)</td>
<td>Identifies a series of commands to be run. The series of commands can be run repeatedly, if you use the UNTIL keyword of the command.</td>
</tr>
<tr>
<td>PLAYBACK commands</td>
<td>Commands to start and stop recording application execution states and replay the recorded execution states.</td>
</tr>
<tr>
<td>Prefix commands (full-screen mode)</td>
<td>Apply only to source listing lines and are typed into the Source window.</td>
</tr>
<tr>
<td>PROCEDURE command</td>
<td>Allows the definition of a group of commands that can be accessed using the CALL procedure command.</td>
</tr>
<tr>
<td>QUALIFY RESET command</td>
<td>Resets qualification to the block of the suspended program and scrolls the source window to display the current statement line.</td>
</tr>
<tr>
<td>QUERY command</td>
<td>Displays the current value of Debug Tool settings (such as the current location in the suspended program).</td>
</tr>
<tr>
<td>QUIT command</td>
<td>Ends a Debug Tool session (with a return code, if specified).</td>
</tr>
<tr>
<td>QUIT command</td>
<td>Ends a Debug Tool session (without additional prompting).</td>
</tr>
<tr>
<td>RETRIEVE command (full-screen mode)</td>
<td>Displays the last command entered on the command line.</td>
</tr>
<tr>
<td>RESTORE command</td>
<td>Enables explicit restoring of settings, breakpoints, and monitor specifications.</td>
</tr>
<tr>
<td>RUN command</td>
<td>Causes Debug Tool to start or resume running your program.</td>
</tr>
<tr>
<td>RUNTO command</td>
<td>Causes Debug Tool to run your program to a specific point (without setting a breakpoint).</td>
</tr>
<tr>
<td>SCROLL command (full-screen mode)</td>
<td>Provides horizontal and vertical scrolling in full-screen mode.</td>
</tr>
<tr>
<td>SELECT command (PL/I)</td>
<td>Chooses one of a set of alternate commands.</td>
</tr>
<tr>
<td>SET command</td>
<td>Controls various Debug Tool settings.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SET command (COBOL)</strong> on page 216</td>
<td>Assigns a value to a COBOL reference.</td>
</tr>
<tr>
<td><strong>SHOW prefix command</strong> (full-screen mode) on page 220</td>
<td>Specifies what relative statement (for C) or relative verb (for COBOL) within the line is to have its frequency count temporarily shown in the suffix area.</td>
</tr>
<tr>
<td><strong>STEP command</strong> on page 220</td>
<td>Causes Debug Tool to dynamically step through a program, running one or more program statements.</td>
</tr>
<tr>
<td><strong>STORAGE command</strong> on page 223</td>
<td>Enables you to alter up to eight bytes of storage.</td>
</tr>
<tr>
<td><strong>switch command (C and C++)</strong> on page 225</td>
<td>Enables you to transfer control to different commands within the switch body, depending on the value of the switch expression.</td>
</tr>
<tr>
<td><strong>SYSTEM command (z/OS)</strong> on page 227</td>
<td>Lets you issue TSO commands during a Debug Tool session.</td>
</tr>
<tr>
<td><strong>TRIGGER command</strong> on page 228</td>
<td>Raises the specified AT condition in Debug Tool, or raises the specified programming language condition in your program.</td>
</tr>
<tr>
<td><strong>TSO command (z/OS)</strong> on page 231</td>
<td>Lets you issue TSO commands during a Debug Tool session (this command is valid only in a TSO environment).</td>
</tr>
<tr>
<td><strong>USE command</strong> on page 232</td>
<td>Causes the Debug Tool commands in the specified file or data set to be either performed or syntax checked.</td>
</tr>
<tr>
<td><strong>while command (C and C++)</strong> on page 233</td>
<td>Enables you to repeatedly perform the body of a loop until the specified condition is no longer met or evaluates to false.</td>
</tr>
<tr>
<td><strong>WINDOW command (full-screen mode)</strong> on page 234</td>
<td>Opens, close, resizes, or expands to full screen (zooms) the specified window on the Debug Tool session panel.</td>
</tr>
</tbody>
</table>

Refer to the following sections for more information related to the material discussed in this section.

**Related tasks**
- [Debug Tool User’s Guide](#)

**Related references**
- Chapter 6, “Debug Tool built-in functions,” on page 239
- Chapter 7, “Debug Tool variables,” on page 245

---

### ? command

The ? command displays a list of Debug Tool commands in the Log window.

```
>>> ?
```

**Usage note**

In the following cases, Debug Tool does not display the syntax help after you enter the ? command:

- The Debug Tool SYSTEM and TSO commands followed by the ? command do not display the syntax help; instead the ? is sent to the host as part of the system command.
- The COMMENT command followed by the ? command does not display the syntax help.
• The SET PFx command accepts a ? as the "command" operand and, in this case, does not display syntax help.

**ALLOCATE command**

The ALLOCATE command allocates a file to an existing data set, a concatenation of existing data sets, or a temporary data set.

```
(1)
ALLOCATE FILE ddname attributes ;
```

**attributes:**

- `DSNAME dsn` The name of an existing data set.
- `DSNAME (dsn,dsn,...)` The names of the existing data sets that need to be concatenated.
- `TEMP TRACKS (primspc,secspc,...)` The number of tracks for the primary space (primspc) and secondary space (secspc) to allocate for the temporary data set.
- `OLD` Set the disposition of the data set to OLD.
- `SHR` Set the disposition of the data set to SHR.
- `MOD` Set the disposition of the data set to MOD.

**Notes:**

1. Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

**FILE ddname**

The DD name of the file.

**DSNAME dsn**

The name of an existing data set.

**DSNAME (dsn,dsn,...)**

The names of the existing data sets that need to be concatenated.

**TEMP**

A temporary data set is allocated.

**TRACKS (primspc,secspc,...)**

The number of tracks for the primary space (primspc) and secondary space (secspc) to allocate for the temporary data set.

**OLD**

Set the disposition of the data set to OLD.

**SHR**

Set the disposition of the data set to SHR.

**MOD**

Set the disposition of the data set to MOD.

**Usage note**

This command is not available under CICS.
ANALYZE command (PL/I)

The ANALYZE command displays the process of evaluating an expression and the data attributes of any intermediate results. To display the results of the expression, use the LIST command.

```
ANALYZE EXPRESSION(expression);
```

**EXPRESSION**

Requests that the accompanying expression be evaluated from the following points of view:

- What are the attributes of each element during the evaluation of the expression?
- What are the dimensions and bounds of the elements of the expression, if applicable?
- What are the attributes of any intermediate results that will be created during the processing of the expression?

**expression**

A valid Debug Tool PL/I expression.

**Usage notes**

- If SET SCREEN ON is in effect, and you want to issue ANALYZE EXPRESSION for an expression in your program, you can bring the expression from the Source window up to the command line by typing over any character in the line that contains the expression. Then, edit the command line to form the desired ANALYZE EXPRESSION command.
- If SET WARNING ON is in effect, Debug Tool displays messages about PL/I computational conditions that might be raised when evaluating the expression.
- Although the PL/I compiler supports the concatenation of GRAPHIC strings, Debug Tool does not.
- The ANALYZE command cannot be used to debug Enterprise PL/I programs.
- The ANALYZE command cannot be used while you replay recorded statements by using the PLAYBACK commands.
- The ANALYZE command cannot be used while you debug a disassembled program.

**Example**

This example is based on the following program segment:

```pli
DECLARE lo_point FIXED BINARY(31,5);
DECLARE hi_point FIXED BINARY(31,3);
DECLARE offset FIXED DECIMAL(12,2);
DECLARE percent CHARACTER(12);
lo_point = 5.4; hi_point = 28.13; offset = -6.77;
percent = '18';
```
The following is an example of the information prepared by issuing ANALYZE EXPRESSION. Specifically, the following shows the effect that mixed precisions and scales have on intermediate and final results of an expression:

```
ANALYZE EXPRESSION { (hi_point - lo_point) + offset / percent }
>>> Expression Analysis <<<
( HI_POINT - LO_POINT ) + OFFSET / PERCENT
HI_POINT - LO_POINT
    HI_POINT
        FIXED BINARY(31,3) REAL
    LO_POINT
        FIXED BINARY(31,5) REAL
        FIXED BINARY(31,5) REAL
OFFSET / PERCENT
    OFFSET
        FIXED DECIMAL(12,2) REAL
    PERCENT
        CHARACTER(12)
        FIXED DECIMAL(15,5) REAL
        FIXED BINARY(31,17) REAL
```

Refer to the following sections for more information related to the material discussed in this section.

**Related references**
- “SET WARNING (C, C++, COBOL, and PL/I)” on page 215
- “PLAYBACK commands” on page 155

### Assignment command (assembler and disassembly)

The Assignment command assigns the value of an expression to a specified memory location or register.

```
receiver [receiverlen] = sourceexpr;
```

- **receiver**
  - A valid Debug Tool assembler reference or expression.

- **receiverlen**
  - A valid Debug Tool assembler reference or expression enclosed in opening and closing brackets (<, >). The value of this reference is used as the length of the receiver.

- **sourceexpr**
  - A valid Debug Tool assembler expression.

### Assignment rules

An assembler assignment is an arithmetic assignment, a bit assignment, or a character assignment.

- Arithmetic assignments are padded (usually with zeros) and truncated on the left. If the source has a type of F or H, the arithmetic statement is padded with sign bits.
- Bit assignments are padded (with zeros) and truncated on the right.
- Character assignments are padded (with blanks) and truncated on the right.

The following table shows how the assignment type is determined from the source and receiver data types. In this table, the following definitions are used:
Indicates an unknown type, for example, R1->+2.

* Indicates any type or length.

**Arithmetic**
Indicates an arithmetic assignment. Padding is on left with sign bits.

**Bit**
Indicates a string assignment padded with zeros.

**Character**
Indicates a string assignment padded with blanks.

**Hex Float**
Hexadecimal floating point assignment.

**String assignment**
The number of bytes that correspond to the Min(receiver length, source length) are moved from the source to the receiver. If the receiver length is larger, it is padded. If the source length is larger, it is truncated. All padding and truncation is done on the right.

**Move**
The number of bytes that correspond to the receiver length are moved directly into the receiver location.

**Error**
Statement that is flagged as not valid.

**Table 2. Assignment rules depending on the source and receiver type**

<table>
<thead>
<tr>
<th>Receiver</th>
<th>Source</th>
<th>Assignment type</th>
<th>Pad or Truncate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Length</td>
<td>Type</td>
<td>Length</td>
</tr>
<tr>
<td>*</td>
<td>1 – *</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E, D, L</td>
<td>4, 8, 16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P, Z</td>
<td>1 – *</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X, B, C</td>
<td>&gt;4</td>
</tr>
<tr>
<td>Other</td>
<td>Other</td>
<td>Other</td>
<td>Error</td>
</tr>
<tr>
<td>X, B</td>
<td>1 – 4</td>
<td>F, H, A, Y</td>
<td>1 – 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P, Z</td>
<td>1 – *</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X, B</td>
<td>1 – *</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>Other</td>
<td>Other</td>
<td>Error</td>
</tr>
<tr>
<td>C</td>
<td>1 – 4</td>
<td>F, H, A, Y</td>
<td>1 – 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P, Z</td>
<td>1 – *</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X, B</td>
<td>1 – *</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>Error</td>
<td>Error</td>
<td></td>
</tr>
<tr>
<td>P, Z</td>
<td>1 – *</td>
<td>P, Z</td>
<td>1 – *</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F, H, A, Y, X, B, C</td>
<td>1 – 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E, D, L</td>
<td>4, 8, 16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Right - 0</td>
</tr>
</tbody>
</table>
Table 2. Assignment rules depending on the source and receiver type (continued)

<table>
<thead>
<tr>
<th>Receiver Type</th>
<th>Source Type</th>
<th>Assignment Type</th>
<th>Pad or Truncate</th>
</tr>
</thead>
<tbody>
<tr>
<td>E, D, L</td>
<td>X</td>
<td>Move</td>
<td>None</td>
</tr>
<tr>
<td>E, D, L</td>
<td>4, 8, 16</td>
<td>Hex Float</td>
<td>Right - 0</td>
</tr>
<tr>
<td>F, H, A, Y</td>
<td>1 - 4</td>
<td>Hex Float</td>
<td>Right - 0</td>
</tr>
<tr>
<td>P, Z</td>
<td>1 - *</td>
<td>Hex Float</td>
<td>Right - 0</td>
</tr>
<tr>
<td>?</td>
<td>1 - 4</td>
<td>Arithmetic</td>
<td>Left</td>
</tr>
<tr>
<td></td>
<td>1 - *</td>
<td>Bit</td>
<td>Right – 0</td>
</tr>
<tr>
<td>All others</td>
<td></td>
<td>Error</td>
<td></td>
</tr>
</tbody>
</table>

Usage notes
- When the receiver expression does not have an implicit length, you must specify a length override and enclose it in angle brackets (<>). For example, `%R1->+10 <4> * 20;` requires an explicit length expression because the receiver expression has no implicit length. However, `X=X+1;` (where `X` is defined as `X DS F`) would not normally have an explicit length specification.
- The Assignment command cannot be used while you replay recorded statements by using the PLAYBACK commands.

Examples
- Assign the value 6 to variable `x`.
  
  ```
  x = 6 ;
  ```
- Increment the value of `X` by 5.
  
  ```
  X = X + 5 ;
  ```
- Assign to `R5` the address of `name_table`.
  
  ```
  %R5 = addr'name_table ;
  ```
- Assign to the `prg_name` variable the value of the character string 'MYPROG'.
  
  ```
  prg_name = 'MYPROG' ;
  ```
- Assign the value of `X` to the 4 bytes at offset 8 from the contents of `R8`.
  
  ```
  %R8->+8 <1'x> = x;
  ```
- Move a string of 14 bytes pointed to by the contents of `R8` (where `R8` was an equated register used in the program) to 6 bytes past the location pointed to by `R2`.
  
  ```
  %R2->+6 <14> = R8->+0;
  ```
- Set 32 bytes pointed to by `R6` to zero.
  
  ```
  %R6->+0 <X'20'> = X'00';
  ```

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

- "references" on page 15
- "PLAYBACK commands" on page 155
Assignment command (OS/VS COBOL)

The Assignment command assigns the value of an expression to a specified reference. It is the equivalent of the OS/VS COBOL COMPUTE statement.

```plaintext
receiver = sourceexpr;
```

receiver
A valid Debug Tool OS/VS COBOL reference enclosed in single-quotes.

sourceexpr
A valid Debug Tool OS/VS COBOL expression enclosed in single-quotes.

Usage notes
- When `receiver` is an arithmetic variable, then `sourceexpr` can be a hexadecimal string of the same length as `receiver`. Debug Tool assumes that the correct internal representation is used and the hexadecimal value is moved directly into `receiver`.
- When `receiver` is a non-numeric string, then `sourceexpr` can be a hexadecimal string of any length. If the length of `sourceexpr` is less than the length of `receiver`, then `receiver` is padded on the right with binary zeros.
- When `receiver` is a COBOL INDEX variable, then Debug Tool assumes that `sourceexpr` is a subscript value and converts it to the proper offset before storing the value into `receiver`.
- The Assignment command cannot be used while you replay recorded statements by using the PLAYBACK commands.

Examples
- Assign the value 6 to variable x.
  ```plaintext
  'x' = '6';
  ```
- Increment the value of X by 5.
  ```plaintext
  'x' = 'x + 5';
  ```

Refer to the following sections for more information related to the material discussed in this section.

Related references
- "references" on page 15
- "PLAYBACK commands" on page 155

Assignment command (PL/I)

The Assignment command assigns the value of an expression to a specified reference.

```plaintext
reference = expression;
```

reference
A valid Debug Tool PL/I reference.

expression
A valid Debug Tool PL/I expression.

Usage notes
- The PL/I repetition factor is not supported by Debug Tool.
For example, the following is not valid: 

\[ rx = (16) '01'B; \]

- If Debug Tool was started because of a computational condition or an attention interrupt, using an assignment to set a variable might not give the expected results. This is because Debug Tool cannot determine variable values within statements, only at statement boundaries.
- The PL/I assignment statement option *BY NAME* is not valid in the Debug Tool.
- If you are debugging a Enterprise PL/I program, the target of an assignment command can not be the variables %EPRn, %FPRn, %GPRn, or %LPRn.
- The Assignment command cannot be used while you replay recorded statements by using the PLAYBACK commands.

**Examples**

- Assign the value 6 to variable *x*.
  
  \[
  x = 6;
  \]

- Assign to the Debug Tool variable %GPR5 the address of *name_table*.
  
  \[
  \%GPR5 = ADOR (name_table);
  \]

- Assign to the *prg_name* variable the value of Debug Tool variable %PROGRAM.
  
  \[
  prg_name = \%PROGRAM;
  \]

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

[“references” on page 15](#)

[“PLAYBACK commands” on page 155](#)

**AT command**

The AT command defines a breakpoint or a set of breakpoints. By defining breakpoints, you can temporarily suspend program execution and use Debug Tool to perform other tasks. By specifying an AT-condition in the AT command, you instruct Debug Tool when to gain control. You can also specify in the AT command what action Debug Tool should take when the AT-condition occurs.

A breakpoint for the specified AT-condition remains established until either another AT command establishes a new action for the same AT-condition or a CLEAR command removes the established breakpoint. An informational message is issued when the first case occurs. Some breakpoints might become obsolete during a debug session and will be cleared automatically by Debug Tool.

For MVS batch, TSO, and CICS programs, the SET SAVE and SET RESTORE commands can be used to automatically save and restore breakpoints between Debug Tool sessions. For all other programs, the SET SAVE and RESTORE commands can be used to automatically save and manually restore breakpoints between sessions.

**For CICS only:** If you do not use the SET SAVE and SET RESTORE commands to control the saving and restoring of breakpoints or monitor specifications and you use a DTCN profile to start a full-screen mode debugging session, Debug Tool preserves the following breakpoints for that session until the DTCN profile is deleted:

- APPEARANCE breakpoints
- CALL breakpoints
- DELETE breakpoints
• ENTRY breakpoints
• EXIT breakpoints
• GLOBAL APPEARANCE breakpoints
• GLOBALCALL breakpoints
• GLOBAL DELETE breakpoints
• GLOBAL ENTRY breakpoints
• GLOBAL EXIT breakpoints
• GLOBAL LABEL breakpoints
• GLOBAL LOAD breakpoints
• GLOBAL STATEMENT/LINE breakpoints
• LABEL breakpoints
• LOAD breakpoints
• OCCURRENCE breakpoints
• STATEMENT/LINE breakpoints
• TERMINATION breakpoint

If a deferred AT ENTRY breakpoint has not been encountered, it is not saved nor restored.

For optimized COBOL programs: The order in which breakpoints are encountered in optimized programs is generally the same as in unoptimized programs. There might be differences due to the effects of optimization.

The following table summarizes the forms of the AT command.

<table>
<thead>
<tr>
<th>AT Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT ALLOCATE (PL/I)* on page 40</td>
<td>Gives Debug Tool control when storage for a named controlled variable or aggregate is dynamically allocated by PL/I.</td>
</tr>
</tbody>
</table>
| AT APPEARANCE* on page 42 | Gives Debug Tool control:  
| | • For C and PL/I, when the specified compile unit is found in storage  
| | • For COBOL, the first time the specified compile unit is called |
| AT CALL* on page 43 | Gives Debug Tool control on an attempt to call the specified entry point. |
| AT CHANGE* on page 45 | Gives Debug Tool control when either the specified variable value or storage location is changed. |
| AT CURSOR (full-screen mode)* on page 50 | Defines a statement breakpoint by cursor pointing. |
| AT DATE (COBOL)* on page 51 | For COBOL, gives Debug Tool control for each date processing statement within the specified block. |
| AT DELETE* on page 51 | Gives Debug Tool control when a load module is deleted. |
| AT ENTRY and AT EXIT* on page 52 | Defines a breakpoint at the specified entry point or exit. |
| AT GLOBAL* on page 54 | Gives Debug Tool control for every instance of the specified AT-condition. |
| AT LABEL* on page 55 | Gives Debug Tool control at the specified statement label. |
| AT LINE* on page 58 | Gives Debug Tool control at the specified line. |
| AT LOAD* on page 58 | Gives Debug Tool control when the specified load module is loaded. |
**Usage notes**

- To set breakpoints at specific locations in a program, Debug Tool depends on that program being loaded into storage. If you issue an AT command for a specific EXIT, LABEL, LINE, or STATEMENT breakpoint and the program is not known by Debug Tool, a warning message is issued and the breakpoint is not set. For ENTRY, the breakpoint becomes a deferred breakpoint.

- To set a global breakpoint, you can specify an asterisk (*) with the AT command or you can specify an AT GLOBAL command. For example, if you want to set a global AT ENTRY breakpoint, specify:
  
  AT ENTRY *;
  or
  AT GLOBAL ENTRY;

- AT CHANGE, AT EXIT, AT LABEL, AT LINE, or AT STATEMENT breakpoints (when entered for a specific block, label, line, or statement) are automatically cleared when the containing compile unit is removed from storage. AT ENTRY breakpoints are converted to deferred AT ENTRY breakpoints.

- AT CHANGE breakpoints are usually automatically cleared when the containing blocks are no longer active or if the relevant variables are in dynamic storage that is freed by a language construct in the program (for example, a C call to free()). However, such breakpoints are not cleared when storage in an assembler or disassembly program is freed via a STORAGE RELEASE macro.

- Clearing of a breakpoint is independent of whether the breakpoint is enabled by using the ENABLE command or disable by using the DISABLE command.

- When multiple AT conditions are raised at the same statement or line, Debug Tool processes them in the following order:
  1. Any global breakpoints other than PATH.
  2. Any PATH breakpoints.
  3. Any statement breakpoints.
  4. Any CHANGE breakpoints

- If you want breakpoints to only stop your program under certain conditions, you can use a combination of the AT command and the IF command to establish a conditional breakpoint.

- The AT commands cannot be used while you replay recorded statements by using the PLAYBACK commands.

Refer to the following sections for more information related to the material discussed in this section.

**Related tasks**
every_clause syntax

Most forms of the AT command contain an optional every_clause that controls whether the specified action is taken based on the number of times a situation has occurred. For example, you might want an action to occur only every 10th time a breakpoint is reached.

The syntax for every_clause is:

```
EVERY integer FROM integer TO integer
```

**EVERY integer**

Specifies how frequently the breakpoint is taken. For example, EVERY 5 means that Debug Tool is started every fifth time the AT-condition is met. The default is EVERY 1.

**FROM integer**

Specifies when Debug Tool invocations are to begin. For example, FROM 8 means that Debug Tool is not started until the eighth time the AT-condition is met. If the FROM value is not specified, its value is equal to the EVERY value.

**TO integer**

Specifies when Debug Tool invocations are to end. For example, TO 20 means that after the 20th time this AT-condition is met, it should no longer start Debug Tool. If the TO value is not specified, the every_clause continues indefinitely.

**Usage notes**

- FROM integer cannot exceed TO integer and all integers must be ≥ 1.
- EVERY by itself is the same as EVERY 1 FROM 1.
- The EVERY, FROM, and TO clauses can be specified in any order.

**Examples**

- Break every third time statement 50 is reached, beginning with the 48th time and ending after the 59th time. The breakpoint action is performed the 48th, 51st, 54th, and 57th time statement 50 is reached.
  
  AT EVERY 3 FROM 48 TO 59 STATEMENT 50;

- At the fifth change of structure field member of the structure named mystruct, print a message saying that it has changed and list its new value. In addition, clear the CHANGE breakpoint. The current programming language setting is C.
  
  AT FROM 5 CHANGE mystruct.member {
      LIST ("mystruct.member has changed.
           It is now", mystruct.member);
      CLEAR AT CHANGE mystruct.member;
  }

**AT ALLOCATE (PL/I)**

AT ALLOCATE gives Debug Tool control when storage for a named controlled variable or aggregate is dynamically allocated by PL/I. When the AT ALLOCATE
breakpoint occurs, the allocated storage has not yet been initialized; initialization, if any, occurs when control is returned to the program.

### identifier
The name of a PL/I controlled variable whose allocation causes an invocation of Debug Tool. If the variable is the name of a structure, only the major structure name can be specified.

* Sets a breakpoint at every ALLOCATE.

### command
A valid Debug Tool command.

#### Usage notes
- The AT ALLOCATE command is not available to debug Enterprise PL/I programs.
- The AT ALLOCATE command cannot be used while you replay recorded statements by using the PLAYBACK commands.

#### Examples
- When the major structure area_name is allocated, display the address of the storage that was obtained.

  ```pli
  AT ALLOCATE area_name LIST ADDR (area_name);
  ```

- List the changes to temp where the storage for temp has been allocated.

  ```pli
  DECLARE temp CHAR(80) CONTROLLED INITIAL('abc');
  AT ALLOCATE temp;
  BEGIN;
      AT CHANGE temp;
      BEGIN;
          LIST (temp);
          GO;
      END;
      GO;
  END;
  GO;
  temp = 'The first time.';
  temp = 'The second time.';
  temp = 'The second time.';
  ```

When temp is allocated the value of temp has not yet been initialized. When it is initialized to 'abc' by the INITIAL phrase, the first AT CHANGE is recognized and 'abc' is listed. The three assignments to temp cause the value to be set again but the third assignment doesn't change the value. This example results in one ALLOCATE breakpoint and three CHANGE breakpoints.

Refer to the following sections for more information related to the material discussed in this section.

**Related references**
- "every_clause syntax" on page 40
- "PLAYBACK commands" on page 155
**AT APPEARANCE**

Gives Debug Tool control when the specified compile unit is found in storage. This is usually the result of a new load module being loaded. However, for modules with the main compile unit in COBOL, the breakpoint does not occur until the compile unit is first entered after being loaded.

\[
\text{AT \{every\_clause\} APPEARANCE \{cu\_spec\} \{command\}:} \]

* Sets a breakpoint at every APPEARANCE of any compile unit.

`command`

A valid Debug Tool command.

**Usage notes**

- If this breakpoint is set in a parent enclave it can be triggered and operated on with breakpoint commands while the application is in a child enclave.
- If the compile unit is qualified with a load module name, the AT APPEARANCE breakpoint will only be recognized for the compile unit that is contained in the specified load module. For example, if a compile unit cu that is in load module loady appears, the breakpoint AT APPEARANCE loadx:::cu will not be triggered.
- If the compile unit is *not* qualified with a load module name, the current load module qualification is not used.
- Debug Tool gains control when the specified compile unit is first recognized by Debug Tool. This can occur when a program is reached that contains a reference to that compile unit. This occurs late enough that the program can be operated on (setting breakpoints, for example), but early enough that the program has not yet been executed. In addition, for C, static variables can also be referenced.
- The AT APPEARANCE command cannot be used while you replay recorded statements by using the PLAYBACK commands.
- AT APPEARANCE is helpful when setting breakpoints in unknown compile units. You can set breakpoints at locations currently unknown to Debug Tool by using the proper qualification and embedding the breakpoints in the command list associated with an APPEARANCE breakpoint. However, there can be only one APPEARANCE breakpoint set at any time for a given compile unit and you must include all breakpoints for that unknown compile unit in a single APPEARANCE breakpoint.
- For a non-CICS application, the AT APPEARANCE breakpoint is cleared at the end of a process.
- Before you enter the AT APPEARANCE command while you debug an assembler or disassembled program, enter the SET ASSEMBLER ON or SET DISASSEMBLY ON command.
- **For C and C++ only:** AT APPEARANCE is not triggered for compile units that reside in a loaded module because the compile units are known at the time of the load.
- **For C, C++, and PL/I only:** An APPEARANCE breakpoint is triggered when Debug Tool finds the specified compile unit in storage. To be triggered, however, the APPEARANCE breakpoint must be set before the compile unit is loaded.

At the time the APPEARANCE breakpoint is triggered, the compile unit you are monitoring has not become the currently-running compile unit. The compile unit
that is current when the new compile unit appears in storage, triggering the
APPEARANCE breakpoint, remains the current compile unit until execution passes
to the new compile unit.

- **For COBOL only:** An APPEARANCE breakpoint is triggered when Debug Tool finds
the specified compile unit in storage. To be triggered, however, the APPEARANCE
breakpoint must be set before the compile unit is called.
At the time the APPEARANCE breakpoint is triggered, the compile unit you are
monitoring has not become the currently-running compile unit. The compile unit
that is current when the new compile unit appears in storage, triggering the
APPEARANCE breakpoint, remains the current compile unit until execution passes
to the new compile unit.

- **For CICS only:** The AT APPEARANCE breakpoint is cleared at the end of the last
process in the application.

**Examples**

- Establish an entry breakpoint when compile unit cu is found in storage. The
current programming language setting is C.

```
AT APPEARANCE cu {
   AT ENTRY a;
   60;
}
```

- Defer the AT EXIT and AT LABEL breakpoints until compile unit cuy is first
entered after being loaded into storage. The current programming language
setting is COBOL.

```
AT APPEARANCE cuy PERFORM
   AT EXIT cuy:>blocky LIST ('Exiting blocky.‘);
   AT LABEL cuy:>lab1 QUERY LOCATION;
END-PERFORM;
```

If cuy is later deleted from storage, the breakpoints that are dependent on cuy
are automatically cleared. However, if cuy is then loaded again, the APPEARANCE
breakpoint for cuy is triggered and the AT EXIT and AT LABEL breakpoints are
redefined.

Refer to the following sections for more information related to the material
discussed in this section.

**Related references**

“every_clause syntax” on page 40
“cu_spec” on page 13
“PLAYBACK commands” on page 155

**AT CALL**

Gives Debug Tool control when the application code attempts to call the specified
entry point. Using CALL breakpoints, you can simulate the execution of unfinished
subroutines, create dummy or stub programs, or set variables to mimic resultant
values, allowing you to test sections of code before the whole is complete.

```
AT every_clause CALL entry_name command;
```
**entry_name**

A valid external entry point name constant or zero (0); however, 0 can only be specified if the current programming language setting is C or PL/I.

* Sets a breakpoint at every **CALL** of any entry point.

**command**

A valid Debug Tool command.

**Usage notes**

- **AT CALL** intercepts the call itself, not the subroutine entry point. C, COBOL, and PL/I programs compiled with the **TEST(PATH)** compiler option identify call targets even if they are unresolved.
- A breakpoint set with **AT CALL** for a call to a C, C++, or PL/I built-in function is never triggered.
- **AT CALL** intercepts calls to entry points known to Debug Tool at compile time. Calls to entry variables are not intercepted, except when the current programming language setting is either C or COBOL (compiled with the **TEST** run-time option).
- **AT CALL 0** intercepts calls to unresolved entry points when the current programming language setting is C or PL/I (compiled with the **TEST** run-time option).
- **AT CALL** allows you to intercept or bypass the target program by using **GO BYPASS** or **GOTO**. If resumed by a normal **GO** or **STEP**, execution resumes by performing the call.
- If you set a breakpoint in a parent enclave, the breakpoint can be triggered and operated on with breakpoint commands while the application is in a child enclave.
- While debugging a CICS application, the breakpoint is cleared at the end of the last process in the CICS application. While debugging a non-CICS application, the breakpoint is cleared at the end of a process.
- The **AT CALL** command cannot be used while you replay recorded statements by using the **PLAYBACK** commands.
- You cannot use the **AT CALL** command while you debug a disassembly program.
- Debug Tool does not support the **AT CALL** command while you debug an OS/VS COBOL or VS COBOL II program.
- **For C and C++ only**: The following usage notes apply:
  - If your C and C++ program has unresolved entry points or entry variables, enter the command **AT CALL 0**.
  - To be able to set breakpoints in a C program using the **AT CALL** command, you must compile your program in one of the following ways:
    - With either the **PATH** or **ALL** suboption of the **TEST** compiler option. The default is **PATH**.
    - With either the **PATH** or **ALL** suboption of the **DEBUG** compiler option.
  - To be able to set breakpoints in a C++ program using the **AT CALL** command, you must compile your program in one of the following ways:
    - With the **TEST** compiler option.
    - With either the **PATH** or **ALL** suboption of the **DEBUG** compiler option.
- **For COBOL only**: The following usage notes apply:
  - **entry_name** can refer to a method as well as a procedure.
  - If **entry_name** is case sensitive, enclose it in double (") or single (') quotes.
To be able to set breakpoints in a COBOL program using the AT CALL command, you must compile your program with either the PATH or ALL suboption of the TEST compiler option. To be able to set breakpoints in a program compiled with the following compilers by using the AT CALL command, you must compile your program with either the PATH, ALL, or NONE suboption of the TEST compiler option:

- Enterprise COBOL for z/OS and OS/390
- COBOL for OS/390 & VM

For programs compiled with the following compilers and using the NONE suboption, AT CALL entry_name is not supported:

- Enterprise COBOL for z/OS and OS/390
- COBOL for OS/390 & VM

Instead, use AT CALL *.

- AT CALL 0 is not supported for use with COBOL programs. However, COBOL is able to identify CALL targets even if they are unresolved, and also identify entry variables and intercept them. Therefore, not all external references need be resolved for COBOL programs.

**For PL/I only:** The following usage notes apply:

- To be able to set CALL breakpoints in PL/I, you must compile your program with either the PATH or ALL suboptions of the TEST compiler option. AT CALL 0 is supported and is called for unresolved external references.
- CALL statements within an INITIAL attribute on a PL/I variable declaration will not trigger AT CALL breakpoints.

**For assembler only:** A CALL statement can be a call to an internal or external routine. A CALL statement is defined to be one of the following opcodes: BALR, BASR, BASSM, BAL, BAS, BRASL, SVC, or PC. You can use the command AT CALL MVS to give Debug Tool control at any SVC or PC instruction.

**Examples**

- Intercept all calls and request input from the terminal.
  
  AT CALL *;

- If the program starts function badsubr, intercept the call, set variable varbl to 50, and then bypass the target function. The current programming language setting is C.
  
  AT CALL badsubr {
  varbl = 50;
  GO BYPASS;
  }

Refer to the following sections for more information related to the material discussed in this section.

**Related tasks**

- Debug Tool User's Guide

**Related references**

- "every_clause syntax" on page 40
- "PLAYBACK commands" on page 155

**AT CHANGE**

Gives Debug Tool control when either the program or Debug Tool command changes the specified variable value or storage location.
AT every_clause CHANGE

reference WHEN condition

命令

Notes:

1. Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

condition
A valid Debug Tool conditional expression.

reference
A valid Debug Tool reference in the current programming language.

reference
A valid Debug Tool reference when the current programming language is OS/VS COBOL.

%STORAGE
A built-in function that provides an alternative way to select an AT CHANGE subject.

address
The starting address of storage to be watched for changes. This must be a hex constant written in one of the following formats:

- 0x in C
- H in COBOL (using either double (" ) or single ( ' ) quotes)
- A PX constant in PL/I.
- X'xxxx' or X"xxxx" in assembler or disassembly
- X"xxxx" in OS/VS COBOL

length
The number of bytes of storage being watched for changes. This must be a positive integer constant. The default value is 1.

command
A valid Debug Tool command.

Usage notes

- To use the AT CHANGE command for a COBOL Level 88 variable, the PTF for Language Environment APAR PK12834 must be installed on z/OS Version 1 Release 4, Version 1 Release 5, Version 1 Release 6 and Version 1 Release 7.
- If an AT CHANGE breakpoint is set on a file record of a BLOCKED QSAM file that is open OUTPUT or EXTEND, the breakpoint might not occur as expected when the WRITE statement is used. The breakpoint behavior in this case is not predictable because the file record is mapped onto the data management buffer.
To get predictable AT CHANGE behavior in this case, set up the file to use a SAME RECORD AREA clause.

- Data is watched only in storage; hence a value that is being kept in a register because of compiler optimization cannot be watched. In addition, the Debug Tool variables %GPRn, %Rn, %FPRn, %LPRn, %EPRn, and any assembler or disassembly symbols representing registers cannot be watched.
- Only entire bytes are watched; bits or bit strings within a byte cannot be singled out.
- Because AT CHANGE breakpoints are identified by storage address and length, it is not possible to have two AT CHANGE breakpoints for the same area (address and length) of storage. That is, an AT CHANGE command replaces a previous AT CHANGE command if the storage address and length are the same. However, any other overlap is ignored and the breakpoints are considered to be for two separate variables. For example, if the storage address is the same, but the length is different, the AT CHANGE command does not replace the previous AT CHANGE.
- When more than one AT CHANGE breakpoint is triggered at a time, AT CHANGE breakpoints are triggered in the order that they were entered. However, if the triggering of one breakpoint causes a variable watched by a different breakpoint to change, the ordering of the triggers will not necessarily be according to when they were originally entered. For example,

```
AT CHANGE y LIST y;
AT CHANGE x y = 4;
GO;
```

If the next statement to be executed in your program causes the value of x to change, the CHANGE x breakpoint is triggered when Debug Tool gains control. Processing of CHANGE x causes the value of y to change. If you type GO; after being informed that CHANGE x was triggered, Debug Tool triggers the CHANGE y breakpoint (before returning control to your program).

In this case, the CHANGE y breakpoint was entered first, but the CHANGE x breakpoint was triggered first (because it caused the CHANGE y breakpoint to be triggered).

- %STORAGE is a Debug Tool built-in function that is available only in conjunction with the CHANGE command.
- For a CICS application on Debug Tool, the CHANGE %STORAGE breakpoint is cleared at the end of the last process in the application. For a non-CICS application on Debug Tool, it is cleared at the end of a process.
- The referenced variables must exist when the AT CHANGE breakpoint is defined. One way to ensure this is to embed the AT CHANGE in an AT ENTRY.
- An AT CHANGE breakpoint gets removed automatically when the specified variable is no longer defined. AT CHANGEs for C static variables are removed when the module defining the variable is removed from storage. For C storage that is allocated using malloc() or calloc(), this occurs when the dynamic storage is freed using free().
- Changes are not detected immediately, but only at the completion of any command that has the potential of changing storage, variable values, or the logical condition. If you specify a single reference, you can restrict the circumstances under which the CHANGE condition is raised by specifying a WHEN condition. If you enter a Debug Tool command that modifies a variable being watched, the CHANGE condition is raised immediately if no WHEN condition is specified. If a WHEN condition is specified, the CHANGE condition is only raised if the variable is modified and the WHEN condition is true. You can force more or less frequent checking by using the SET CHANGE command.
- C and C++ AT CHANGE breakpoint requirements
- The variable must be an lvalue or an array.
- The variable must be declared in an active block if the variable is a parameter or has a storage class of auto.
- If you specify the address of the storage containing the variable, it must be specified with a hexadecimal constant.
- A CHANGE breakpoint defined for a static variable is automatically removed when the file in which the variable was declared is no longer active. A CHANGE breakpoint defined for an external variable is automatically removed when the module where the variable was declared is no longer active.

COBOL AT CHANGE breakpoint requirements
- AT CHANGE using a storage address should not reference a data item that follows a variable-size element or subgroup within a group. COBOL dynamically remaps the group when a variable-size element changes size.
- If you specify the address of the storage containing the variable, it must be with an \$ constant, delimited by either quotation marks or apostrophes. The \$ constant can only be used where an address or POINTER variable can be used. The COBOL hexadecimal notations for nonnumeric literals should be used for all other situations.
- Be careful when examining a variable whose allocated storage follows that of a variable-size element. COBOL dynamically remaps the storage for the element any time it changes size. This could alter the address of the variable you want to examine.
- You cannot set a CHANGE breakpoint for a COBOL file record before the file is opened.
- The variable, when in the local storage section, must be declared in an active block.

PL/1 AT CHANGE breakpoint requirements
- CHANGE breakpoint is removed for based or controlled variables when they are FREEd and for parameters and AUTOMATIC variables when the block in which they are declared is no longer active.
- CHANGE monitors only structures with single scalar elements. Structures containing more than one scalar element are not supported.
- The variable must be a valid reference for the current block.
- The breakpoint is automatically removed after the referenced variable ceases to exist.
- A CHANGE breakpoint monitors the storage allocated to the current generation of a controlled variable. If you subsequently allocate new generations, they are not monitored.
- If you specify the address of storage containing the variable, you must do so with a PX constant, delimited by single or double quotation marks. The PX constant can only be used where an address or pointer variable can be used.

- When you free storage with the STORAGE RELEASE macro in an assembler or disassembly program, it is not possible to detect when the storage is freed. If you set an AT CHANGE breakpoint on storage freed by a STORAGE RELEASE macro, unexpected results might occur, such as the triggering of the breakpoint at unexpected times.
- The AT CHANGE command cannot be used while you replay recorded statements by using the PLAYBACK commands.
- For optimized COBOL programs, the specified variable cannot be a variable that was discarded due to compiler optimization.
• When you use a Level 88 variable on an AT CHANGE command, the current setting of the value is saved. Debug Tool stops at the breakpoint only if the setting of the Level 88 variable changes from the saved value to a different value. For example, if the saved value was TRUE and the new value is FALSE, Debug Tool stops at the breakpoint.
• To use a Level 88 variable with the AT CHANGE command, you (through a Debug Tool command) or the program must have previously set the variable to one of the values specified in the variable’s declaration. If you do not do this, Debug Tool behavior becomes unpredictable.
• When you use a condition, the variables used in the condition or the condition are not evaluated at the time the breakpoint is set but when the location associated with the AT CHANGE command changes.
• Only the following conditional operators can be used in a condition:
  =  Compare the two operands for equality.
  ~=  Compare the two operands for inequality.
  <  Determines whether the left operand is less than the right operand.
  >  Determines whether the left operand is greater than the right operand.
  <=  Determines whether the left operand is less than or equal to the right operand.
  >=  Determines whether the left operand is greater than or equal to the right operand.
  &  Logical ”and” operation.
  |  Logical ”or” operation.
• If you use the AT CHANGE command with a WHEN condition, every time the variable changes the condition is evaluated. If the condition evaluates to true, Debug Tool stops and runs the command associated with the breakpoint.
• After Debug Tool detects a change in the value, if the evaluation of the condition fails, Debug Tool stops and informs you that the condition failed.

Examples
• Identify the current location each time variable varbl1 or varbl2 is found to have a changed value. The current programming language setting is COBOL.
  AT CHANGE (varbl1, varbl2) PERFORM
  QUERY LOCATION;
  GO;
  END-PERFORM;
• When storage at the hex address 22222 changes, print a message in the log. Eight bytes of storage are to be watched. The current programming language setting is C.
  AT CHANGE  %STORAGE (0x00022222, 8)
  LIST "Storage has changed at hex address 22222";
• Set two breakpoints when storage at the hex address 1000 changes. The variable x is defined at hex address 1000 and is 20 bytes in length. In the first breakpoint, 20 bytes of storage are to be watched. In the second breakpoint, 50 bytes of storage are to be watched. The current programming language setting is C.
  AT CHANGE  %STORAGE (0x00010000, 20)  /* Breakpoint 1 set */
  AT CHANGE  %STORAGE (0x00010000, 50)  /* Breakpoint 2 set */
  AT CHANGE  x  /* Replaces breakpoint 1, since x is at */
  /* hex address 1000 and is 20 bytes long */
• Stop when a variable reaches a value that is greater than 200.
AT CHANGE MYVAR WHEN MYVAR > 200;

MYVAR > 200 is a condition. Every time the value of MYVAR changes, the condition MYVAR > 200 is evaluated. Changes to MYVAR do not trigger the AT CHANGE breakpoint. Only when MYVAR changes and the condition MYVAR > 200 becomes true is the AT CHANGE breakpoint triggered.

Refer to the following sections for more information related to the material discussed in this section.

Related tasks
- Debug Tool User's Guide

Related references
- "every_clause syntax" on page 40
- "references" on page 15
- "PLAYBACK commands" on page 155

AT CURSOR (full-screen mode)

Provides a cursor controlled method for setting a statement breakpoint. It is most useful when assigned to a PF key.

TOGGLE

Specifies that if the cursor-selected statement already has an associated statement breakpoint then the breakpoint is removed rather than replaced.

Usage notes
- AT CURSOR does not allow specification of an every_clause or a command.
- Do not use a semicolon.
- The cursor must be in the Source window and positioned on a line where an executable statement begins. An AT STATEMENT command for the first executable statement in the line is generated and executed (or cleared if one is already defined and TOGGLE is specified). For optimized COBOL programs, the first statement on the line might have been discarded due to optimization effects. Therefore, the first executable statement might be the second statement or later.
- The AT CURSOR command cannot be used while you replay recorded statements by using the PLAYBACK commands.

Example

Define a PF key to toggle the breakpoint setting at the cursor position.

SET PF10 = AT TOGGLE CURSOR;

Refer to the following sections for more information related to the material discussed in this section.

Related references
- "PLAYBACK commands" on page 155
AT DATE (COBOL)

Gives Debug Tool control for each date processing statement within the specified block. A date processing statement is a statement that references a date field, or an EVALUATE or SEARCH statement WHEN phrase that references a date field.

```
AT DATE block_spec command;
```

* Sets a breakpoint at every date processing statement.

`command`
A valid Debug Tool command.

Usage notes
- When you use AT DATE, execution is halted only for COBOL compile units compiled with the DATEPROC compiler option.
- The AT DATE command cannot be used while you replay recorded statements by using the PLAYBACK commands.

Examples
- Each time a date processing statement is encountered in the nested subprogram, display the location of the statement.
  ```
  AT DATE subrx QUERY LOCATION;
  ```
- Each time a date processing statement is encountered in the compile unit, display the name of the compile unit.
  ```
  AT DATE * LIST %CU;
  ```
- Each time a date processing statement is encountered in the compile unit, display the location of the statement, list a specific variable, and resume running the program.
  ```
  AT DATE * PERFORM
    QUERY LOCATION;
    LIST DATE-FIELD
    GO;
  END-PERFORM;
  ```

Refer to the following sections for more information related to the material discussed in this section.

Related references
- “every_clause syntax” on page 40
- “block_spec” on page 11
- “PLAYBACK commands” on page 155

AT DELETE

Gives Debug Tool control when a load module is removed from storage by a Language Environment, MVS, or CICS delete service, such as on completion of a successful C release(), COBOL CANCEL, PL/I RELEASE, assembler DELETE macro, or EXEC CICS RELEASE.
Every time a load module is deleted, request input from the terminal.

- `AT DELETE *;`
- Stop watching variable `var1:>x` when load module `mymod` is deleted.

- `AT DELETE mymod CLEAR AT CHANGE (var1:>x);`

Refer to the following sections for more information related to the material discussed in this section.

**Related references**
- "every_clause syntax“ on page 40
- "load_spec“ on page 15
- "PLAYBACK commands“ on page 155

**AT ENTRY and AT EXIT**

Defines a breakpoint at the specified entry point or exit in the specified block.

- `AT ENTRY block_spec command;`
Usage notes

- For VS COBOL II programs, Debug Tool supports only the AT ENTRY * or AT EXIT * commands.
- An AT EXIT breakpoint can only be set for programs that are currently fetched or loaded. To set an exit breakpoint for a currently unknown compile unit, use the AT APPEARANCE command.
- To specify an AT ENTRY breakpoint for a program that is not currently known to Debug Tool, you must do one of the following:
  - If the name of the program is the same as the block_spec, you do not need to qualify the block_spec with the name of the program.
  - If the name of the program is not the same as the block_spec, you need to qualify the block_spec with a program name. When Debug Tool detects a program name that matches the one you specified, it sets the breakpoint.
- An ENTRY or EXIT breakpoint set for a compile unit that becomes nonactive (one that is not in the current enclave), is suspended until the compile unit becomes active. An ENTRY/EXIT breakpoint set for a compile unit that is deleted from storage is suspended until the compile unit is reloaded. A suspended breakpoint cannot be triggered until it is reactivated.
- For a CICS application on Debug Tool, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on Debug Tool, it is cleared at the end of a process.
- EXIT breakpoints for blocks in a fetched or loaded program are removed when that program is released.
- ENTRY breakpoints for blocks in a fetched or loaded program are converted to deferred breakpoints when that program is released.
- The AT ENTRY and AT EXIT commands cannot be used while you replay recorded statements by using the PLAYBACK commands.
- You cannot use the AT ENTRY command to stop at the entry to a Language Environment MAIN routine for an enclave other than the first enclave if you do not have debug data available for the containing compile unit.
- You cannot use the AT EXIT command when you are in a disassembly compile unit.
- You cannot use the AT EXIT command when you are in an OS/VS COBOL compile unit.
- For assembler only: AT EXIT gains control on exit from internal or external routines. An EXIT is defined to be one of the following opcodes:
  - BR
  - BALR, BASR, or BASSM when it is not followed by a valid instruction

Examples

- At the entry of program subrx, initialize variable ix and continue program execution. The current programming language setting is COBOL.
  AT ENTRY subrx PERFORM
  SET ix TO 5;
  GO;
  END-PERFORM;
- At exit of main, print a message and TRIGGER the SIGUSR1 condition. The current programming language setting is C.
AT EXIT main {
    puts("At exit of the program");
    TRIGGER SIGUSR1;
    GO;
}

Refer to the following sections for more information related to the material discussed in this section.

**Related references**
- "every_clause syntax" on page 40
- "block_spec" on page 11
- "AT APPEARANCE" on page 42
- "PLAYBACK commands" on page 155

**AT GLOBAL**

Gives Debug Tool control for every instance of the specified AT-condition. These breakpoints are independent of their nonglobal counterparts (except for AT PATH, which is identical to AT GLOBAL PATH). Global breakpoints are always performed before their specific counterparts.

```
AT GLOBAL every_clause command;
```

*command*

A valid Debug Tool command.

You should use GLOBAL breakpoints where you don’t have specific information of where to set your breakpoint. For example, you want to halt at entry to block Abcdefg_Unknown but cannot remember the name, you can issue AT GLOBAL ENTRY and Debug Tool will halt every time a block is being entered. If you want to halt at every function call, you can issue AT GLOBAL CALL.

**Usage notes**

- Debug Tool does not support the AT CALL, AT LABEL and AT PATH commands for disassembled or VS COBOL II programs.
- Debug Tool does not support the AT CALL command for OS/VS COBOL programs.
- To set a global breakpoint, you can specify an asterisk (*) with the AT command or you can specify an AT GLOBAL command.
- Although you can define GLOBAL breakpoints to coexist with singular breakpoints of the same type at the same location or event, COBOL does not allow you to define two or more single breakpoints of the same type for the same location or event. The last breakpoint you define replaces any previous breakpoint.
- The AT GLOBAL command cannot be used while you replay recorded statements by using the PLAYBACK commands.
Examples

- If you want to set a global AT ENTRY breakpoint, specify:
  
  ```
  AT ENTRY *;
  or
  AT GLOBAL ENTRY;
  ```

- At every statement or line, display a message identifying the statement or line. The current programming language setting is COBOL.
  ```
  AT GLOBAL STATEMENT LIST ('At Statement:', %STATEMENT);
  ```

- If you enter (for COBOL):
  ```
  AT EXIT table1 PERFORM
  LIST TITLED (age, pay);
  GO;
  END-PERFORM;
  ```

  then enter:
  ```
  AT EXIT table1 PERFORM
  LIST TITLED (benefits, scale);
  GO;
  END-PERFORM;
  ```

  only benefits and scale are listed when your program reaches the exit point of block table1. The second AT EXIT replaces the first because the breakpoints are defined for the same location. However, if you define the following GLOBAL breakpoint:
  ```
  AT GLOBAL EXIT PERFORM
  LIST TITLED (benefits, scale);
  GO;
  END-PERFORM;
  ```

in conjunction with the first EXIT breakpoint, when your program reaches the exit from table1, all four variables (age, pay, benefits, and scale) are listed with their values, because the GLOBAL EXIT breakpoint can coexist with the EXIT breakpoint set for table1.

- To set a GLOBAL DATE breakpoint, specify:
  ```
  AT DATE *;
  ```

or
  ```
  AT GLOBAL DATE;
  ```

- To combine a global breakpoint with other Debug Tool commands, specify:
  ```
  AT GLOBAL DATE QUERY LOCATION;
  ```

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

- "every_clause_syntax" on page 40
- "PLAYBACK commands" on page 155

**AT LABEL**

Gives Debug Tool control when execution has reached the specified statement label or group of labels. For C and PL/I, if there are multiple labels associated with a single statement, you can specify several labels and Debug Tool gains control at each label. For COBOL, AT LABEL lets you specify several labels, but for any group of labels that are associated with a single statement, Debug Tool gains control for that statement only once.
* Sets a breakpoint at every LABEL.

    command
    A valid Debug Tool command.

Usage notes

- Debug Tool does not support the AT LABEL command with VS COBOL II programs.
- A COBOL statement_label can have either of the following forms:
  - name
    This form can be used in COBOL for reference to a section name or for a COBOL paragraph name that is not within a section or is in only one section of the block.
  - name1 OF name2 or name1 IN name2
    This form must be used for any reference to a COBOL paragraph (name1) that is within a section (name2), if the same name also exists in other sections in the same block. You can specify either OF or IN, but Debug Tool always uses OF for output.

Either form can be prefixed with the usual block, compile unit, and load module qualifiers.
- For C, C++ or PL/I, you can set a LABEL breakpoint at each label located at a statement. This is the only circumstance where you can set more than one breakpoint at the same location.
- A LABEL breakpoint set for a nonactive compile unit (one that is not in the current enclave) is suspended until the compile unit becomes active. A LABEL breakpoint set for a compile unit that is deleted from storage is suspended until the compile unit is reloaded. A suspended breakpoint cannot be triggered until it is reactivated.
- For a CICS application on Debug Tool, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on Debug Tool, it is cleared at the end of a process.
- You cannot set LABEL breakpoints at PL/I label variables.
- LABEL breakpoints for label constants in a fetched, loaded program or DLL are removed when that program is released.
- To be able to set LABEL breakpoints in PL/I, you must compile your program with either the PATH and SYM suboptions or the ALL suboption of the TEST compiler option.
- For C, to be able to set LABEL breakpoints, you must compile your program in one of the following ways:
  - With either the PATH and SYM suboptions or ALL suboption of the TEST compiler option. The default is PATH.
  - With either the PATH and SYM suboptions or ALL suboption of the DEBUG compiler option.
- For C++, to be able to set LABEL breakpoints, you must compile your program in one of the following ways:
  - With the TEST compiler option.
- With either the PATH and SYM suboptions or ALL suboption of the DEBUG compiler option.
- You can set breakpoints for more than one label at the same location. Debug Tool is entered for each specified label.
- To be able to set LABEL breakpoints in COBOL, you must compile your program with either the STMT, PATH, or ALL suboption and the SYM suboption of the TEST compiler option.

When defining specific LABEL breakpoints Debug Tool sets a breakpoint for each label specified, unless there are several labels on the same statement. In this case, only the last LABEL breakpoint defined is set.
- For COBOL, a reference to a label or a label constant can take either of the following forms:
  - name
    This form is used to refer to a section name or the name of a paragraph contained in not more than one section of the block.
  - name1 OF name2 or name1 IN name2
    This form is used to refer to a paragraph contained within a section if the paragraph name exists in other sections in the same block. You can use either 0F or IN, but Debug Tool only uses 0F for output to the log file.
- For PL/I users:
  - If you are running any version of VisualAge PL/I or Enterprise PL/I Version 3 Release 1 through Version 3 Release 3 programs, you cannot use the AT LABEL command.
  - If you are running Enterprise PL/I Version 3 Release 4 programs and you comply with the following requirements, you can use the AT LABEL command to set breakpoints (except at a label variable):
    - You apply PTFs for APARs PK00118 and PK00339 to the Enterprise PL/I Version 3 Release 4 compiler.
- You cannot use the AT LABEL command while you use the disassembly view.
- The AT LABEL command cannot be used while you replay recorded statements by using the PLAYBACK commands.

Examples
- Set a breakpoint at label create in the currently qualified block.
  AT LABEL create;
- At program label para 0F sect1 display variable names x and y and their values, and continue program execution. The current programming language setting is COBOL.
  AT LABEL para 0F sect1 PERFORM
    LIST TITLED (x, y);
    GO;
  END-PERFORM;
- Set a breakpoint at labels label1 and label2, even though both labels are associated to the same statement. The current programming language setting is C.
  AT LABEL label1 LIST 'Stopped at label1'; /* Label1 is first */
  AT LABEL label2 LIST 'Stopped at label2'; /* Label2 is second */

Chapter 5. Debug Tool commands  57
Refer to the following sections for more information related to the material discussed in this section.

**Related references**
- “every_clause syntax” on page 40
- “statement label” on page 17
- “PLAYBACK commands” on page 155

### AT LINE

Gives Debug Tool control at the specified line.

The AT LINE command is synonymous to the AT STATEMENT command.

You cannot use the AT LINE while you debug a disassembled program. Instead, use the AT OFFSET command.

The AT LINE command cannot be used while you replay recorded statements by using the PLAYBACK commands.

Refer to the following sections for more information related to the material discussed in this section.

**Related references**
- “AT OFFSET (disassembly)” on page 63
- “PLAYBACK commands” on page 155
- “AT STATEMENT” on page 65

### AT LOAD

Gives Debug Tool control when the specified load module is brought into storage. For example, Debug Tool gains control on completion of a successful C fetch(), a PL/I FETCH, during a COBOL dynamic CALL, MVS LOAD service, or EXEC CICS LOAD. To stop at a compile unit or program in a COBOL DLL, use AT APPEARANCE. Once the breakpoint is raised for the specified load module, it is not raised again unless either the load module is released and fetched again or another load module with the specified name is fetched.

You can set LOAD breakpoints regardless of what compiler options are in effect.

```
 AT every_clause LOAD module_name
   
   load_spec command;
   
   * Sets a breakpoint at every LOAD of any load module.

command
   A valid Debug Tool command.
```

**Usage notes**
- Debug Tool gains control for loads that are affected by the Language Environment load service, the MVS LOAD service, or EXEC CICS LOAD. A LOAD
breakpoint is triggered when a new enclave is entered. If the Dynamic Debug facility is deactivated (by entering the SET DYNDEBUG OFF command) or SVC screening is disabled, Debug Tool is not notified of any loads that are affected by the MVS LOAD service. Refer to **Debug Tool Customization Guide** for instructions on how to control SVC screening.

- **AT LOAD** can be used to detect the loading of specific language library load modules; however, the loading of language library load modules does not trigger an **AT GLOBAL LOAD** or **AT LOAD**.
- **AT LOAD** cannot specify the initial load module because it is already loaded when Debug Tool is started.
- If this breakpoint is set in a parent enclave, it can be triggered and operated on with breakpoint commands while the application is in a child enclave.
- For a CICS application on Debug Tool, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on Debug Tool, it is cleared at the end of a process.
- **AT LOAD** on an implicitly or explicitly loaded DLL is not supported by Debug Tool.
- Debug Tool recognizes an implicitly loaded DLL only after a compile unit in that DLL is called. For example, if **LIST NAMES** **CUS** is issued after an implicit load of a DLL, there will be no entry in the output of the command of the DLL.
- Depending on the version of the C or C++ compiler used, Debug Tool might recognize a compile unit in a DLL only after it has had a function in it called. For example, if a DLL contains a function **fn1** in **CU file1** and it contains a function **fn2** in **CU file2**, a call to **fn1 will not** enable Debug Tool to recognize **file2**, only **file1**. Similarly, a call to **fn2 will not** enable Debug Tool to recognize **file1**.
- At the triggering of a **LOAD** breakpoint for C, C++, and PL/I, Debug Tool has enough information about the loaded module to set breakpoints and examine variables of static and external storage classes.
- At the triggering of a **LOAD** breakpoint for COBOL, C, and C++ DLL’s, Debug Tool does not have enough information about the loaded module to set breakpoints in blocks contained within the module. At the triggering of an **APPEARANCE** breakpoint, however, you can set such breakpoints.
- The **AT LOAD** command cannot be used while you replay recorded statements by using the **PLAYBACK** commands.

**Examples**

- Print a message when load module **mymod** is loaded. The current programming language setting is either C, C++, or COBOL.
  
  **AT LOAD** **mymod** **LIST** (*"Load module mymod has been loaded"*);

- Establish an entry breakpoint when load module **a** is fetched and then resume execution. The current programming language setting is C.
  
  **AT LOAD** **a** { 
  **AT ENTRY** **a**;
  **GO**;
  }

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

- **"every_clause_syntax" on page 40**
- **"load_spec" on page 15**
- **"PLAYBACK commands" on page 155**
AT OCCURRENCE

Gives Debug Tool control on a language or Language Environment condition or exception or an MVS or CICS ABEND.

\[ \text{AT } \text{every_clause} \text{ OCCURRENCE } \text{condition} \text{ command} ; \]

\[ (\text{condition}) \]

\textit{condition}

A valid condition or exception. This can be one of the following codes or conditions:

- A Language Environment symbolic feedback code.
- A language-oriented keyword or code, depending on the current programming language setting.
- An MVS System or User ABEND code Sxxx or Uxxx, where xxx is three hexadecimal digits corresponding to the desired ABEND code. These codes are valid only when you are running without the Language Environment run time.
- Any four-character string representing a CICS ABEND code. This code is valid only when you are running without the Language Environment run time.

Following are the C and C++ condition constants; they must be uppercase and not abbreviated:

<table>
<thead>
<tr>
<th>C and C++ Condition Constants</th>
<th>Language Environment Symbolic Feedback Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGABND</td>
<td>SIGILL</td>
</tr>
<tr>
<td>SIGABRT</td>
<td>SIGINT</td>
</tr>
<tr>
<td>SIGFPE</td>
<td>SIGIOERR</td>
</tr>
<tr>
<td></td>
<td>SIGSEGV</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When a C++ user specifies AT CONDITION THROWOBJ, Debug Tool transfers control to the user at the point of the throw in C++ code.

PL/I condition constants can be used. However, FILE condition constants and CONDITION condition constants can not be used while debugging Enterprise PL/I programs.

There are no COBOL condition constants. Instead, an Language Environment symbolic feedback code must be used, for example, CEE347.

The TRAP (ON) run-time option must be used to stop on Language Environment conditions or MVS or CICS Abends.

\textit{command}

A valid Debug Tool command.

Program conditions and condition handling vary from language to language. The methods the OCCURRENCE breakpoint uses to adapt to each language are described below.

**For C and C++:**

When a C and C++ or an Language Environment condition occurs during your session, the following series of events takes place:

1. Debug Tool is started before any C or C++ signal handler.
2. If you set an OCCURRENCE breakpoint for that condition, Debug Tool processes that breakpoint and executes any commands you have specified. If you did not set an OCCURRENCE breakpoint for that condition, and:
   - If the current test-level setting is ALL, Debug Tool prompts you for commands or reads them from a commands file.
   - If the current test-level setting is ERROR, and the condition has an error severity level (that is, anything but SIGUSR1, SIGUSR2, SIGINT, or SIGTERM), Debug Tool gets commands by prompting you or by reading from a commands file.
   - If the current test-level setting is NONE, Debug Tool ignores the condition and returns control to the program.

You can set OCCURRENCE breakpoints for equivalent C and C++ signals and Language Environment conditions. For example, you can set AT OCCURRENCE CEE345 and AT OCCURRENCE SIGSEGV during the same debug session. Both indicate an addressing exception and, if you set both breakpoints, no error occurs. However, if you set OCCURRENCE breakpoints for a condition using both its C, C++, and Language Environment designations, the Language Environment breakpoint is the only breakpoint triggered. Any command list associated with the C condition is not executed.

You can use OCCURRENCE breakpoints to control your program's response to errors.

**Usage notes**

- If the application program also has established an exception handler for the condition then that handler is entered when Debug Tool releases control, unless return is by use of 60 BYPASS or GOTO or a specific statement.
- OCCURRENCE breakpoints for COBOL IGZ conditions can only be set after a COBOL run-time module has been initialized.
- For C, C++, and PL/I, certain Language Environment conditions map to C and C++ SIGxxx values and PL/I condition constants. It is possible to enter two AT OCCURRENCE breakpoints for the same condition. For example, one could be entered with the Language Environment condition name and the other could be entered with the C and C++ SIGxxx condition constant. In this case, the AT OCCURRENCE breakpoint for the Language Environment condition name is triggered and the AT OCCURRENCE breakpoint for the C or C++ condition constant is not. However, if an AT OCCURRENCE breakpoint for the Language Environment condition name is not defined, the corresponding mapped C, C++, or PL/I condition constant is triggered.
- If this breakpoint is set in a parent enclave it can be triggered and operated on with breakpoint commands while the application is in a child enclave.
- For a CICS application on Debug Tool, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on Debug Tool, it is cleared at the end of a process.
- For COBOL, Debug Tool detects Language Environment conditions. If a Language Environment condition occurs during your session, the following series of events takes place:
  1. Debug Tool is started before any condition handler.
  2. If you set an OCCURRENCE breakpoint for that condition, Debug Tool processes that breakpoint and executes any commands you have specified. If you have not set an OCCURRENCE breakpoint for that condition, and:

     - If the current test-level setting is ALL, Debug Tool prompts you for commands or reads them from a commands file.
If the current test-level setting is ERROR, and the condition has a severity level of 2 or higher, Debug Tool gets commands by prompting you or by reading from a commands file.
- If the current test-level setting is NONE, Debug Tool ignores the condition and returns control to the program.

You can use OCCURRENCE breakpoints to control your program’s response to errors.
- For PL/I, Debug Tool detects Language Environment and PL/I conditions. If a condition occurs, Debug Tool is started before any condition handler. If you have issued an ON command or set an OCCURRENCE breakpoint for the specified condition, Debug Tool runs the associated commands.
- If there is no AT OCCURRENCE or ON set, then:
  - If the current test-level setting is ALL, Debug Tool prompts you for commands or reads them from a commands file.
  - If the current test-level setting is ERROR, and the condition has an error severity level of 2 or higher, Debug Tool gets commands by prompting you or by reading from a commands file.
  - If the current test-level setting is NONE, Debug Tool ignores the condition and returns control to the program.
- Once Debug Tool returns control to the program, any relevant PL/I ON-unit is run.
- If you are debugging a program that uses SPIE or ESPIE, while SPIE or ESPIE is active, the program behaves as if TRAP(0FF) was specified for all program checks except for a program check that might arise from the use of the CALL command.
- If you are debugging a program that uses ESTAE or ESTAEX, while ESTAE or ESTAEX is active, the program behaves as if TRAP(0FF) was specified for all abends except program checks. Debug Tool does not handle any conditions. The ESTAE or ESTAEX exit handles any abends except for program checks.
- The AT OCCURRENCE command cannot be used while you replay recorded statements using the PLAYBACK commands.

Examples
- When a data exception occurs, query the current location. The current programming language setting is either C or COBOL.
  AT OCCURRENCE CEE347 QUERY LOCATION;
- When you are running in MVS without the Language Environment run time, that is under EQANMDBG, when a System 0C1 ABEND occurs, list information about the current CUs with the following command:
  AT OCCURRENCE 0C1 DESCRIBE CUS;
- When the SIGSEGV condition is raised, set an error flag and call a user termination routine. The current programming language setting is C.
  AT OCCURRENCE SIGSEGV {
    error = 1;
    terminate (error);
  }
- Suppose SIGFPE maps to CEE347 and the following breakpoints are defined. The current programming language setting is C.
  AT OCCURRENCE SIGFPE LIST "SIGFPE condition";
  AT OCCURRENCE CEE347 LIST "CEE347 condition";
  If the Language Environment condition CEE347 is raised, the CEE347 breakpoint is triggered.
However, if a breakpoint had not been defined for CEE347 and the CEE347 condition is raised, the SIGFPE breakpoint is triggered (because it is mapped to CEE347).

Refer to the following sections for more information related to the material discussed in this section.

**Related references**
- “every_clause syntax” on page 40
- “ON command (PL/I)” on page 149
- "PLAYBACK commands" on page 155
- z/OS Language Environment Programming Guide
- z/OS Language Environment Debugging Guide
- PL/I for MVS and VM Language Reference

### AT OFFSET (disassembly)

Gives Debug Tool control at the specified offset in the disassembly view.

```plaintext
 AT OFFSET x'offset';
```

**offset**
- A hexadecimal offset in the disassembly view as seen in the Source window prefix area.

**Usage notes**
- The AT OFFSET command cannot be used while you replay recorded statements by using the PLAYBACK commands.

**Examples**
- Set a breakpoint at offset ‘2A’ in the current block:
  ```plaintext
  AT OFFSET X'2A';
  ```
- Set a breakpoint at offsets ‘2A’ and ‘30’ in the current block:
  ```plaintext
  AT OFFSET (X'2A',X'30');
  ```
- Set a breakpoint in the block MYPROG at offset ‘3A’:
  ```plaintext
  AT OFFSET MYPROG:X'3A';
  ```

Refer to the following sections for more information related to the material discussed in this section.

**Related references**
- "PLAYBACK commands” on page 155

### AT PATH

Gives Debug Tool control when the flow of control changes (at a path point). AT PATH is identical to AT GLOBAL PATH.

```plaintext
 AT [every_clause] PATH command;
```

**command**
- A valid Debug Tool command.

**Usage notes**
For a CICS application on Debug Tool, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on Debug Tool, it is cleared at the end of a process.

For C, to be able to set PATH breakpoints, you must compile your program in one of the following ways:
- With either the PATH or ALL suboption of the TEST compiler option. The default is PATH.
- With either the PATH or ALL suboption of the DEBUG compiler option.

For C++, to be able to set PATH breakpoints, you must compile your program in one of the following ways:
- With the TEST compiler option.
- With either the PATH or ALL suboption of the DEBUG compiler option.

For C++, COBOL, and PL/I, you can set PATH breakpoints at any time.

You cannot use the AT PATH command while you replay recorded statements by using the PLAYBACK commands.

Debug Tool does not support the AT PATH command while you debug a disassembled program or a VS COBOL II program.

Examples
- Whenever a path point has been reached, display the five most recently processed breakpoints and conditions.
  AT PATH LIST LAST 5 HISTORY;
- Whenever a path point has been reached, display a message and query the current location. The current programming language setting is COBOL.
  AT PATH PERFORM
    LIST "Path point reached";
    QUERY LOCATION;
    GO;
  END-PERFORM;
- Whenever a path point has been reached, the value of %PATHCODE contains the code representing the type of path point stopped at. If the program is stopped at the entry to a block, display the %PATHCODE.
  AT PATH LIST %PATHCODE;

Refer to the following sections for more information related to the material discussed in this section.

Related tasks
Debug Tool User’s Guide

Related references
- “every_clause syntax” on page 40
- “%PATHCODE” on page 252
- “PLAYBACK commands” on page 155

AT Prefix (full-screen mode)
Sets a statement breakpoint when you issue this command through the Source window prefix area. When one or more breakpoints have been set on a line, the prefix area for that line is highlighted.
integer
Selects a relative statement (for C, C++, and PL/I) or a relative verb (for COBOL) within the line. The default value is 1. For optimized COBOL programs, the default value is the first executable statement on the line, which was not discarded due to optimization effects.

Usage note
The AT Prefix command cannot be used while you replay recorded statements by using the PLAYBACK commands.

Example
Set a breakpoint at the third statement or verb in the line (typed in the prefix area of the line where the statement is found).
AT 3
No space is needed as a delimiter between the keyword and the integer; hence, AT 3 is equivalent to AT3.

Refer to the following sections for more information related to the material discussed in this section.
Related references
"PLAYBACK commands" on page 155

AT STATEMENT
Gives Debug Tool control at each specified statement or line within the given set of ranges.

<table>
<thead>
<tr>
<th>AT</th>
<th>every_clause</th>
<th>LINE</th>
<th>every_clause</th>
<th>statement_id_range</th>
</tr>
</thead>
</table>

command;

* Sets a breakpoint at every STATEMENT or LINE.

Usage notes
• You cannot use the AT STATEMENT command (except for the AT STATEMENT * form) while you debug a disassembled program. Instead, use the AT OFFSET command.
• A STATEMENT breakpoint set for a nonactive compile unit (one that is not in the current enclave), is suspended until the compile unit becomes active. A STATEMENT breakpoint set for a compile unit that is deleted from storage is suspended until the compile unit is reloaded. A suspended breakpoint cannot be triggered until it is reactivated.
• For a CICS application on Debug Tool, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on Debug Tool, it is cleared at the end of a process.
• You can specify the first relative statement on each line in any of three ways. If, for example, you want to set a STATEMENT breakpoint at the first relative
statement on line three, you can enter AT 3, AT 3.0, or AT 3.1. However, Debug Tool logs them differently according to the current programming language as follows:

- **For C and C++**
  The first relative statement on a line is specified with "0". All of the above breakpoints are logged as AT 3.0.

- **For COBOL or PL/I**
  The first relative statement on a line is specified with "1". All of the above breakpoints are logged as AT 3.1. For optimized COBOL programs, the first relative statement is the first executable statement. This might not be the first statement if the optimizer discarded the first statement.

  - When the STORAGE run-time option is in effect, the AT STATEMENT command cannot be used to set a breakpoint in the prologue of an assembler compile unit between the first BALR 14,15 instruction and the following LR 13,x instruction.
  - The AT STATEMENT command cannot be used while you replay recorded statements by using the PLAYBACK command.

**Examples**

- Set a breakpoint at statement or line number 23. The current programming language setting is COBOL.
  AT 23 LIST 'About to close the file';

- Set breakpoints at statements 5 through 9 of compile unit mycu. The current programming language setting is C.
  AT STATEMENT "mycu":+5 - 9;

- Set breakpoints at lines 19 through 23 and at statements 27 and 31.
  AT LINE (19 - 23, 27, 31);

  or

  AT LINE (27, 31, 19 - 23);

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

- "every_clause syntax" on page 40
- "statement_id_range and stmt_id_spec" on page 16
- "AT OFFSET (disassembly)" on page 63
- "PLAYBACK commands" on page 155

**AT TERMINATION**

Gives Debug Tool control when the application program is terminated.

>>> AT—TERMINATION—command—;

**command**

A valid Debug Tool command.

**Usage notes**

- The setting of the current programming language when the application program terminates might be unpredictable.
- AT TERMINATION does not allow specification of an every_clause because termination can only occur once.
If this breakpoint is set in a parent enclave, it can be triggered and operated on with breakpoint commands while the application is in a child enclave.

When Debug Tool gains control, normal execution of the program is complete; however, a CALL or function invocation from Debug Tool can continue to perform program code. When the AT TERMINATION breakpoint gives control to Debug Tool:
- Fetched load modules have not been released
- Files have not been closed
- Language-specific termination has been started yet no action has been taken
In C, the user atexit() lists have already been called.
In PL/I, the FINISH condition was already raised.

You are allowed to enter any command with AT TERMINATION. However, normal error messages are issued for any command that cannot be completed successfully because of lack of information about your program.

You can enter DISABLE AT TERMINATION; or CLEAR AT TERMINATION; at any time to disable or clear the breakpoint. It remains disabled or cleared until you reenable or reset it.

For a CICS application on Debug Tool, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on Debug Tool, it is cleared at the end of a process.

The AT TERMINATION command cannot be used while you replay recorded statements by using the PLAYBACK commands.

Examples
- When the program ends, check the Debug Tool environment to see what files have not been closed.
  AT TERMINATION DESCRIBE ENVIRONMENT;
- When the program ends, display the message "Program has ended" and end the Debug Tool session. The current programming language setting is C.
  AT TERMINATION {
      LIST "Program has ended";
      QUIT;
  }

Refer to the following sections for more information related to the material discussed in this section.

Related references
"PLAYBACK commands" on page 155

BEGIN command (PL/I)

BEGIN and END delimit a sequence of one or more commands to form one longer command. The BEGIN and END keywords cannot be abbreviated.

command
A valid Debug Tool command.

Usage notes
- The BEGIN command is most helpful when used in AT, IF, or ON commands.
- The BEGIN command does not imply a new block or name scope. It is equivalent to a PL/I simple DO.
- You cannot use the BEGIN command while you replay recorded statements by using the PLAYBACK commands.

Examples
- Set a breakpoint at statement 320 listing the value of variable x and assigning the value of 2 to variable a.
  
  AT 320 BEGIN;
  LIST (x);
  a = 2;
  END;
- When the PL/I condition FIXEDOVERFLOW is raised (that is, when the length of the result of a fixed-point arithmetic operation exceeds the maximum length allowed) list the value of variable x and assign the value of 2 to variable a. The current programming language setting is PL/I.
  
  ON FIXEDOVERFLOW BEGIN; LIST (x); a=2; END;

block command (C and C++)

The block command allows you to group any number of Debug Tool commands into one command. When you enclose Debug Tool commands within a single set of braces ({}), everything within the braces is treated as a single command. You can place a block anywhere a command is allowed.

\[
\{ \text{command} \};
\]

command

A valid Debug Tool command.

Usage notes
- Declarations are not allowed within a nested block.
- The C block command does not end with a semicolon. A semicolon after the closing brace is treated as a Null command.
- You cannot use the block command while you replay recorded statements by using the PLAYBACK commands.

Example

Establish an entry breakpoint when load module a is fetched.

AT LOAD a {
  AT ENTRY a;
  GO;
}
break command (C and C++)

The break command allows you to terminate and exit a loop (that is, do, for, and while) or switch command from any point other than the logical end. You can place a break command only in the body of a looping command or in the body of a switch command. The break keyword must be lowercase and cannot be abbreviated.

```
break;
```

In a looping statement, the break command ends the loop and moves control to the next command outside the loop. Within nested statements, the break command ends only the smallest enclosing do, for, switch, or while commands.

In a switch body, the break command ends the execution of the switch body and gives control to the next command outside the switch body.

Usage notes
• You cannot use the break command while you replay recorded statements by using the PLAYBACK commands.

Examples
• The following example shows a break command in the action part of a for command. If the i-th element of the array string is equal to '\0', the break command causes the for command to end.
```
for (i = 0; i < 5; i++) {
    if (string[i] == '\0')
        break;
    length++;  
}
```
• The following switch command contains several case clauses and one default clause. Each clause contains a function call and a break command. The break commands prevent control from passing down through subsequent commands in the switch body.
```
char key;

key = '-';
AT LINE 15 switch (key)
{
    case '+':
        add();
        break;
    case '-':
        subtract();
        break;
    default:
        printf("Invalid key\n");
        break;
}
```

CALL command

The CALL command calls either a procedure, entry name, or program name, or it requests that a utility function be run. The C and C++ equivalent for CALL is a function reference. PL/I subroutines or functions cannot be called dynamically during a Debug Tool session. The CALL keyword cannot be abbreviated.
In C++, calls can be made to any user function as long as the function is declared as:

```
extern "C"
```

In COBOL, the CALL command cannot be issued when Debug Tool is at initialization.

The following table summarizes the forms of the CALL command.

<table>
<thead>
<tr>
<th>CALL form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL %CEBR</td>
<td>Starts the CICS Temporary Storage Browser Program.</td>
</tr>
<tr>
<td>CALL %CECI on page 71</td>
<td>Starts the CICS Command Level Interpreter Program.</td>
</tr>
<tr>
<td>CALL %DUMP on page 71</td>
<td>Calls a dump service to obtain a formatted dump.</td>
</tr>
<tr>
<td>CALL %FA on page 76</td>
<td>Calls IBM Fault Analyzer to provide a formatted dump of the current machine state.</td>
</tr>
<tr>
<td>CALL %HOGAN on page 76</td>
<td>Starts Computer Sciences Corporation’s KORE-HOGAN application.</td>
</tr>
<tr>
<td>CALL %VER on page 77</td>
<td>Adds a line to the log describing the maintenance level of Debug Tool that you have installed on your system.</td>
</tr>
<tr>
<td>CALL entry_name (COBOL) on page 77</td>
<td>Calls an entry name in the application program (COBOL).</td>
</tr>
<tr>
<td>CALL procedure* on page 79</td>
<td>Calls a procedure that has been defined with the PROCEDURE command.</td>
</tr>
</tbody>
</table>

### CALL %CEBR

Starts the CICS Temporary Storage Browser Program.

```
(1)
CALL %CEBR
```

**Notes:**

1. Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

**Usage notes**

- Debug Tool performs an EXEC CICS LINK to the CICS browser program. When CEBR processing is complete, control is returned to Debug Tool through an EXEC CICS return.
- You can use this command only when you debug CICS programs in single-terminal mode in full-screen mode.

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

- CICS Supplied Transactions
- CICS Application Programming Guide
CALL %CECI

Starts the CICS Command Level Interpreter Program.

(1)

CALL %CECI ;

Notes:
1. Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

Usage notes
- Debug Tool performs an EXEC CICS LINK to the CICS command level interpreter program. When CECI processing is complete, control is returned to Debug Tool through an EXEC CICS return.
- You can use this command only when you debug CICS programs in single-terminal mode in full-screen mode.

Refer to the following sections for more information related to the material discussed in this section.

Related references
- CICS Supplied Transactions
- CICS Application Programming Guide

CALL %DUMP

Calls a dump service to obtain a formatted dump.

CALL %DUMP

(title)

options_string

A fixed-length character string, conforming to the current programming language syntax for a character string constant (that is, enclosed in quotes according to the rules of that programming language). The string length cannot exceed 80 bytes.

options_string

A fixed-length character string, conforming to the current programming language syntax for a character string constant, which specifies the type, format, and destination of dump information. The string length cannot exceed 247 bytes.

Options are declared as a string of keywords separated by blanks or commas. Some options have suboptions that follow the option keyword and are contained in parentheses. The options can be specified in any order, but the last option declaration is honored if there is a conflict between it and any preceding options.

The options_string can include the following:

THREAD (ALL | CURRENT)

Dumps the current thread or all threads associated with the current
enclave. The default is to dump only the current thread. Only one thread is supported. For enclaves that consist of a single thread, THREAD(ALL) and THREAD(CURRENT) are equivalent.

THREAD can be abbreviated as THR.
CURRENT can be abbreviated as CUR.

CICS: This option is not supported when you are running under CICS without Language Environment, where Debug Tool issues an EXEC CICS DUMP TRANSACTION.

TRACEBACK
Requests a traceback of active procedures, blocks, condition handlers, and library modules on the call chain. The traceback shows transfers of control from either calls or exceptions. The traceback extends backward to the main program of the current thread.

TRACEBACK can be abbreviated as TRACE.

NOTRACEBACK
Suppresses traceback.
NOTRACEBACK can be abbreviated as NOTRACE.

FILES
Requests a complete set of attributes of all files that are open and the contents of the buffers used by the files.

FILES can be abbreviated as FILE.

NOFILES
Suppresses file attributes of files that are open.
NOFILES can be abbreviated as NOFILE.

VARIABLES
Requests a symbolic dump of all variables, arguments, and registers.
Variables include arrays and structures. Register values are those saved in the stack frame at the time of call. There is no way to print a subset of this information.

Variables and arguments are printed only if the symbol tables are available. A symbol table is generated if a program is compiled using the compile options shown below for each language:

<table>
<thead>
<tr>
<th>Language</th>
<th>Compiler option</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>TEST(SYM)</td>
</tr>
<tr>
<td>C++</td>
<td>TEST</td>
</tr>
<tr>
<td>COBOL</td>
<td>TEST or TEST(h,SYM)</td>
</tr>
<tr>
<td>PL/I</td>
<td>TEST(SYM)</td>
</tr>
</tbody>
</table>

The variables, arguments, and registers are dumped starting with Debug Tool. The dump proceeds up the chain for the number of routines specified by the STACKFRAME option.

VARIABLES can be abbreviated as VAR.

NOVARIABLES
Suppresses dump of variables, arguments, and registers.

NOVARIABLES can be abbreviated as NOVAR.
BLOCKS

Produces a separate hexadecimal dump of control blocks.

Global control blocks and control blocks associated with routines on the call chain are printed. Control blocks are printed for Debug Tool. The dump proceeds up the call chain for the number of routines specified by the STACKFRAME option.

If FILES is specified, this is used to produce a separate hexadecimal dump of control blocks used in the file analysis.

BLOCKS can be abbreviated as BLOCK.

CICS: This option is not supported when you are running under CICS without Language Environment, where Debug Tool issues an EXEC CICS DUMP TRANSACTION.

NOBLOCKS

Suppresses the hexadecimal dump of control blocks.

NOBLOCKS can be abbreviated as NOBLOCK.

STORAGE

Dumps the storage used by the program.

The storage is displayed in hexadecimal and character format. Global storage and storage associated with each routine on the call chain is printed. Storage is dumped for Debug Tool. The dump proceeds up the call chain for the number of routines specified by the STACKFRAME option.

Storage for all file buffers is also dumped if the FILES option is specified. When the Dynamic Debug facility is activated, some of the original application instructions are not displayed because they are replaced by '0A91'x instructions.

STORAGE can be abbreviated as STOR.

NOSTORAGE

Suppresses storage dumps.

NOSTORAGE can be abbreviated as NOSTOR.

STACKFRAME(n|ALL)

Specifies the number of stack frames dumped from the call chain.

If STACKFRAME(ALL) is specified, all stack frames are dumped. No stack frame storage is dumped if STACKFRAME(0) is specified.

The particular information dumped for each stack frame depends on the VARIABLE, BLOCK, and STORAGE option declarations specified. The first stack frame dumped is the one associated with Debug Tool, followed by its caller, and proceeding backward up the call chain.

STACKFRAME can be abbreviated to SF.

PAGESIZE(n)

Specifies the number of lines on each page of the dump.

This value must be greater than 9. A value of zero (0) indicates that there should be no page breaks in the dump.

PAGESIZE can be abbreviated to PAGE.

FNAME(s)

Specifies the ddname of the file where the dump report is written.

The default ddname CEEDUMP is used if this option is not specified.
**CONDITION**
Specifies that for each condition active on the call chain, the following information is dumped from the Condition Information Block (CIB):
- The address of the CIB
- The message associated with the current condition token
- The message associated with the original condition token, if different from the current one
- The location of the error
- The machine state at the time the condition manager was started
- The ABEND code and REASON code, if the condition occurred because of an ABEND.

The particular information that is dumped depends on the condition that caused the condition manager to be started. The machine state is included only if a hardware condition or ABEND occurred. The ABEND and REASON codes are included only if an ABEND occurred.

**NOCONDITION**
Suppresses dump condition information for active conditions on the call chain.
**NOCONDITION** can be abbreviated as NOCOND.

**ENTRY**
Includes in the dump a description of the Debug Tool routine that called the dump service and the contents of the registers at the point of the call. For the currently supported programming languages, ENTRY is extraneous and will be ignored.

**CICS:** This option is not supported when you are running under CICS without Language Environment, where Debug Tool issues an EXEC CICS DUMP TRANSACTION.

**NOENTRY**
Suppresses the description of the Debug Tool routine that called the dump service and the contents of the registers at the point of the call.

**CICS:** This option is not supported when you are running under CICS without Language Environment, where Debug Tool issues an EXEC CICS DUMP TRANSACTION.

The defaults for the preceding options are:
- CONDITION
- FILES
- FNAME(CEEDUMP)
- NOBLOCKS
- NOENTRY
- NOSTORAGE
- PAGESIZE(60)
- STACKFRAME(ALL)
- THREAD(CURRENT)
- TRACEBACK
- VARIABLES

**Usage notes**
- If incorrect options are used, a default dump is written.
The service used to format the dump is determined by the following conditions:

**Language Environment is active**
Language Environment dump service: Debug Tool does not analyze any of the CALL %DUMP options, but just passes them to the Language Environment dump service. Some of these options might not be appropriate, because the call is being made from Debug Tool rather than from your program.

**Language Environment not active and you are running under CICS**
The command: EXEC CICS DUMP TRANSACTION DUMPCODE('DT') COMPLETE

**Language Environment not active and you are not running under CICS**
The MVS SNAP dump service

When you use CALL %DUMP, one of the following ddnames must be allocated for you to receive a formatted dump:
- CEEDUMP (default)
- SYSPRINT.

Control might not be returned to Debug Tool after the dump is produced, depending on the option string specified.

**CICS:** You do not need this allocation when you are running without Language Environment under CICS. Under those conditions, EXEC CICS DUMP TRANSACTION is issued, and a transaction dump with a code of DT$ is written to the CICS dump data set.

**COBOL** does not do anything if the FILES option is specified; the BLOCKS option gives the file information instead.

Using a small n (like 1 or 2) with the STACKFRAME option will not produce useful results because only the Debug Tool stack frames appear in your dump. Larger values of n or ALL should be used to ensure that application stack frames are shown.

When you use the CALL %DUMP command and the Language Environment run time is not active, the MVS SNAP macro or the EXEC CICS DUMP command is used to generate the dump. When you are not running under CICS, the following restrictions apply:
- The specified or default ddname must be allocated to a data set with these attributes: RECFM=VBA, LRECL=125, and BLKSIZE=1632
- The previously described options are mapped into SNAP options as shown in the following table:

<table>
<thead>
<tr>
<th>%DUMP option</th>
<th>SNAP option</th>
</tr>
</thead>
<tbody>
<tr>
<td>THREAD</td>
<td>ignored</td>
</tr>
<tr>
<td>TRACEBACK</td>
<td>SDATA=(PCDATA),PDATA=(SA,SAH)</td>
</tr>
<tr>
<td>FILES</td>
<td>SDATA=(DM,IO)</td>
</tr>
<tr>
<td>VARIABLES</td>
<td>SDATA=(CB)</td>
</tr>
<tr>
<td>BLOCKS</td>
<td>SDATA=(SQA,LSAQ,SWA)</td>
</tr>
<tr>
<td>STORAGE</td>
<td>PDATA=(LPA,JPA,SPLS)</td>
</tr>
<tr>
<td>STACKFRAME</td>
<td>ignored</td>
</tr>
<tr>
<td>PAGESIZE</td>
<td>ignored</td>
</tr>
<tr>
<td>FNAME</td>
<td>ddname for dump</td>
</tr>
<tr>
<td>CONDITION</td>
<td>SDATA=(Q,TRT,ERR)</td>
</tr>
</tbody>
</table>

Table 3. %DUMP options mapping to SNAP options

Chapter 5. Debug Tool commands
Table 3. %DUMP options mapping to SNAP options (continued)

<table>
<thead>
<tr>
<th>%DUMP option</th>
<th>SNAP option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTRY</td>
<td>PDATA=(SUBTASKS)</td>
</tr>
</tbody>
</table>

- The CALL %DUMP command cannot be used while you replay recorded statements by using the PLAYBACK commands.

Examples
- Request a formatted dump that traces active procedures, blocks, condition handlers, and library modules. Identify the dump as "Dump after read".
  CALL %DUMP ("TRACEBACK", "Dump after read");
- Call the dump service to obtain a formatted dump including traceback information, file attributes, and buffers.
  CALL %DUMP ("TRACEBACK FILES");

Refer to the following sections for more information related to the material discussed in this section.

Related references
- "PLAYBACK commands" on page 155
- z/OS Language Environment Programming Guide
- z/OS Language Environment Debugging Guide

CALL %FA

Starts and instructs IBM Fault Analyzer to provide a formatted dump of the current machine state.

(1)

Notes:
1. Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

CALL %HOGAN

Starts Computer Sciences Corporation’s KORE-HOGAN application, also known as SMART (System Memory Access Retrieval Tool).

(1)

Notes:
1
• Available only with Debug Tool Utilities and Advanced Functions (5655-R45).
• CICS only.

Usage notes
• You can use this command only when you debug CICS programs in single-terminal mode in full-screen mode.
• If you do not have the KORE-HOGAN application, do not use this command. If you do use this command, a Program not loadable error occurs, which raises an AEI0 exception.

CALL %VER
 Adds a line to the log describing the maintenance level of Debug Tool that you have installed on your system.

```plaintext
CALL %VER;
```

Usage note
You can use this command in remote debug mode.

Example
You have Debug Tool for z/OS, Version 7 Release 1, with the PTF for APAR PK9nnnnn installed on your system. Enter the CALL %VER command to display the following information in the Log window:

```
IBM Debug Tool Version 7 Release 1 Mod 0
2006/10/28 06:43:00 AM Level: V7R1 PK9nnnnn
5655-R44 and 5655-R45: (C) Copyright IBM Corp. 1992, 2006
```

The time stamp that is shown is the product build date and time.

CALL entry_name (COBOL)
 Calls an entry name in the application program. The entry name must be a valid external entry point name (that is, callable from other compile units).

```plaintext
CALL identifier literal USING identifier_clause;
```
identifier_clause:

identifier
   A valid Debug Tool COBOL identifier.

literal
   A valid COBOL literal.

Usage notes

- If you have a COBOL entry point name that is the same as a Debug Tool procedure name, the procedure name takes precedence when using the CALL command. If you want the entry name to take precedence over the Debug Tool procedure name, you must qualify the entry name when using the CALL command.

- You can use the CALL entry_name command to change program flow dynamically. You can pass parameters to the called module.

- The CALL follows the same rules as calls within the COBOL language.

- The COBOL ON OVERFLOW and ON EXCEPTION phrases are not supported, so END-CALL is not supported.

- Only calls to separately compiled programs are supported; nested programs are not callable by this Debug Tool command (they can of course be started by GOTO or STEP to a compiled-in CALL).

- All calls are dynamic, that is, the called program (whether specified as a literal or as an identifier) is loaded when it is called.

- See Enterprise COBOL for z/OS Language Reference for an explanation of the following COBOL keywords: ADDRESS, BY, CONTENT, LENGTH, OF, REFERENCE, USING.

- An entry_name cannot refer to a method.

- A windowed date field cannot be specified as the identifier containing the entry name.

- The CALL entry_name command cannot be used while you replay recorded statements by using the PLAYBACK commands by using the PLAYBACK command.

Example

Call the entry name sub1 passing the variables a, b, and c.

CALL "sub1" USING a b c;

Refer to the following sections for more information related to the material discussed in this section.

Related references
PLAYBACK commands" on page 155
CALL procedure

Calls a procedure that has been defined with the PROCEDURE command.

```
CALL procedure_name;
```

`procedure_name`

The name given to a sequence of Debug Tool commands delimited by a
PROCEDURE command and a corresponding END command.

Usage notes

- Because the Debug Tool procedure names are always uppercase, the procedure name is converted to uppercase even for programming languages that have mixed-case symbols.
- The CALL keyword is required even for programming languages that do not use CALL for subroutine invocations.
- The CALL command is restricted to calling procedures in the currently executing enclave.

Example

Create and call the procedure named proc1.

```
proc1: PROCEDURE;
  LIST (r, c);
END;
END;
AT 54 CALL proc1;
```

CLEAR command

The CLEAR command removes the actions of previously entered Debug Tool commands. Some breakpoints are removed automatically when Debug Tool determines that they are no longer meaningful. For example, if you set a breakpoint in a fetched or loaded compile unit, the breakpoint is discarded when the compile unit is released.
Notes:

1  Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

AT  Removes all breakpoints, including GLOBAL breakpoints, set by previously entered AT commands, except for AT TERMINATION breakpoints.

  AT_command

  A valid AT command that includes at least one operand. The AT command must be complete except that the every_clause and command are omitted.

  generic_AT_command

  A valid AT command without operands. It can be one of the following: ALLOCATE, APPEARANCE, CALL, CHANGE, CURSOR, DATE, DELETE, ENTRY, EXIT, LABEL, LOAD, OFFSET, OCCURRENCE, PATH, STATEMENT (the LINE keyword can be used in place of STATEMENTS), or TERMINATION.

DECLARE

Removes previously defined variables and tags. If no identifier follows DECLARE, all session variables and tags are cleared. DECLARE is equivalent to VARIABLES.
identifier
The name of a session variable or tag declared during the Debug Tool session. This operand must follow the rules for the current programming language.

EQUATE
Removes previously defined symbolic references. If no identifier follows EQUATE, all existing SET EQUATE synonyms are cleared.

identifier
The name of a previously defined reference synonym declared during the Debug Tool session using SET EQUATE. This operand must follow the rules for the current programming language.

LOAD
Removes the load module. This command has the following sub-parameter:

module_name
The name of one or more load modules that were loaded by Debug Tool using the LOAD command.

LOG
Erases the log file and clears out the data being retained for scrolling. In line mode, CLEAR LOG clears only the log file.
If the log file is directed to a SYSOUT type file, CLEAR LOG will not clear the log contents in the file.

MONITOR
Clears the commands defined for MONITOR. If no number follows MONITOR, the entire list of commands affecting the monitor window is cleared; the monitor window is empty.

number
A positive integer that refers to a monitored command. If a list of integers is specified, all commands represented by the specified list are cleared.

ON (PL/I)
Removes the effect of an earlier ON command. If no pli_condition follows ON, all existing ON commands are cleared.

pli_condition
Identifies an exception condition for which there is an ON command defined.

PROCEDURE
Clears previously defined Debug Tool procedures. If no procedure_name follows PROCEDURE, all inactive procedures are cleared.

procedure_name
The name given to a sequence of Debug Tool commands delimited by a PROCEDURE command and a corresponding END command. The procedure must be currently in storage and not active.

VARIABLES
Removes previously defined variables and tags. If no identifier follows VARIABLES, all session variables and tags are cleared. VARIABLES is equivalent to DECLARE.

identifier
The name of a session variable or tag declared during the Debug Tool session. This operand must follow the rules for the current programming language.
Usage notes

- You can use the CLEAR AT command to clear either active or suspended breakpoints. However, you cannot use it to clear suspended label breakpoints.
- If you want to clear a suspended breakpoint, you must specify both the load module and CU name.
- You can use the CLEAR LOAD command in remote debug mode when you use one of the following remote debuggers:
  - Compiled Language Debugger component of WebSphere Studio Enterprise Developer
  - Compiled Language Debugger component of WebSphere Developer for zSeries
  - WebSphere Developer Debugger for zSeries
- In some environments, a loaded module cannot be removed from storage. In this case the command fails and the load module remains in storage.
- You can enter CL in the prefix area of the monitor window to clear the selected line in the monitor window.
- You can use the CLEAR MONITOR n command to clear an automonitor entry in the Monitor window.
- Only an AT LINE or AT STATEMENT breakpoint can be cleared with a CLEAR AT CURSOR command.
- To clear every single breakpoint in the Debug Tool session, issue CLEAR AT followed by CLEAR AT TERMINATION.
- To clear a global breakpoint, you can specify an asterisk (*) with the CLEAR AT command or you can specify a CLEAR AT GLOBAL command.
  If you have only a global breakpoint set and you specify CLEAR AT ENTRY without the asterisk (*) or GLOBAL keyword, you get a message saying there are no such breakpoints.
- The CLEAR AT, CLEAR DECLARE, CLEAR ON, and CLEAR VARIABLES commands cannot be used while you replay recorded statements by using the PLAYBACK commands.

Examples

- Remove the LABEL breakpoint set in the program at label create.
  CLEAR AT LABEL create;
- Remove previously defined variables x, y, and z.
  CLEAR DECLARE (x, y, z);
- Remove the effect of the ninth command defined for MONITOR.
  CLEAR MONITOR 9;
- Remove the structure type definition tagone (assuming all variables declared interactively using the structure tag have been cleared). The current programming language setting is C.
  CLEAR VARIABLES struct tagone;
- Establish some breakpoints with the AT command and then remove them with the CLEAR command (checking the results with the LIST command).
  AT 50;
  AT 56;
  AT 55 LIST (r, c);
  LIST AT;
  CLEAR AT 50;
  LIST AT;
  CLEAR AT;
  LIST AT;
If you want to clear an AT ENTRY * breakpoint, specify:

```
CLEAR AT ENTRY *
```
or

```
CLEAR AT GLOBAL ENTRY;
```

If you want to remove the DATE breakpoint for block MYBLOCK, specify:

```
CLEAR AT DATE MYBLOCK;
```

If you want to remove a generic DATE breakpoint, specify:

```
CLEAR AT DATE *;
```

Refer to the following sections for more information related to the material discussed in this section.

**Related references**
- “CLEAR prefix (full-screen mode)”
- “AT command” on page 37
- “LIST command” on page 124
- “PLAYBACK commands” on page 155

### CLEAR prefix (full-screen mode)

Clears a breakpoint when you enter this command through the Source window prefix area or clears a selected member of the current set of MONITOR commands when you enter this command through the Monitor window prefix area.

```
CLEAR integer ;
```

**integer**

Selects a relative statement (for C and PL/I) or a relative verb (for COBOL) within the line to remove the breakpoint if there are multiple statements on that line. The default value is 1. For optimized COBOL programs, the first relative statements is the first executable statement, which was not discarded by the optimizer.

**Usage notes**

- The CLEAR prefix command cannot be used while you replay recorded statements by using the PLAYBACK commands.
- Use CL in the Monitor window prefix area to clear a member of Monitor window.

**Examples**

- In the Source window, clear a breakpoint at the third statement or verb in the line (typed in the prefix area of the line where the statement is found).

```
CLEAR 3
```

No space is needed as a delimiter between the keyword and the integer; hence, CLEAR 3 is equivalent to CLEAR3.
- In the Monitor window, type CL in the prefix area to on the line that displays the entry you want to remove, then press Enter.
**COMMENT command**

The COMMENT command can be used to insert commentary into the session log. The COMMENT keyword cannot be abbreviated.

```
>>-COMMENT-[commentary];
```

*commentary*

Commentary text not including a semicolon. An embedded semicolon is not allowed; text after a semicolon is treated as another Debug Tool command. DBCS characters can be used within the commentary.

The COMMENT command can be used as an executable command, that is it can be the subject of a conditional command, but it is treated as a Null command.

**Examples**

- Comment that varblxx seems to have the wrong value.
  
  `COMMENT At this point varblxx seems to have the wrong value;`

- Combine a commentary with valid Debug Tool commands.
  
  `COMMENT Entering subroutine testrun; LIST (x); GO;`

**COMPUTE command (COBOL)**

The COMPUTE command assigns the value of an arithmetic expression to a specified reference. The COMPUTE keyword cannot be abbreviated.

```
>>-COMPUTE-[reference]=-[expression];
```

*reference*

A valid Debug Tool COBOL numeric reference.

*expression*

A valid Debug Tool COBOL numeric expression.

**Usage notes**

- If you are debugging an optimized COBOL program, you can use the COMPUTE command to assign a value to a program variable only if you first enter the SET WARNING OFF command.
- If you are debugging an optimized COBOL program and you specify an expression, you can reference program variables that were not discarded by the optimizer.
- If Debug Tool was started because of a computational condition or an attention interrupt, using an assignment to set a variable might not give expected results. This is due to the uncertainty of variable values within statements as opposed to their values at statement boundaries.
- COMPUTE assigns a value only to a single receiver; unlike COBOL, multiple receiver variables are not supported.
- Floating-point receivers are not supported; however, floating-point values can be set by using the MOVE command.
- The COBOL EQUAL keyword is not supported ("=" must be used).
- The COBOL ROUNDED and SIZE ERROR phrases are not supported, so END-COMPUTE is not supported.
• COMPUTE cannot be used to perform a computation with a windowed date field if the expression consists of more than one operand.
• Any expanded date field specified as an operand in the expression is treated as a nondate field.
• The result of the evaluation of the expression is always considered to be a nondate field.
• If the expression consists of a single numeric operand, the COMPUTE will be treated as a MOVE and therefore subject to the same rules as the MOVE command.
• If the DATA parameter of the PLAYBACK ENABLE command is in effect for the current compile unit, the COMPUTE command can be used while you replay recorded statements by using the PLAYBACK commands. The target of the COMPUTE command must be a session variable.
• The value assigned to a variable is always assigned to the storage for that variable. In an optimized program, a variable can be temporarily assigned to a register, and a new value assigned to that variable does not necessarily alter the value used by the program.

Examples
• Assign to variable x the value of a + 6.
  COMPUTE x = a + 6;
• Assign to the variable mycode the value of the Debug Tool variable %PATHCODE + 1.
  COMPUTE mycode = %PATHCODE + 1;
• Assign to variable xx the result of the expression (a + e(1)) / c * 2.
  COMPUTE xx = (a + e(1)) / c * 2;
  You can also use table elements in such assignments as shown in the following example.
  COMPUTE itm-2(1,2) = (a + 10) / e(2);
• To assign a value to a session variable named TSO or SYSTEM, append the "=" to the reference as shown in the following example.
  COMPUTE tso= 5;

Refer to the following sections for more information related to the material discussed in this section.

Related references
"MOVE command (COBOL)" on page 145
"PLAYBACK commands" on page 155
"SET WARNING (C, C++, COBOL, and PL/I)" on page 215

CURSOR command (full-screen mode)

The CURSOR command moves the cursor between the last saved position on the Debug Tool session panel (excluding the header fields) and the command line.

----------CURSOR----------

Usage notes
• The cursor position can be saved by typing the CURSOR command on the command line and moving the cursor before pressing Enter, or by moving the cursor and pressing a PF key with the CURSOR command assigned to it.
• If the CURSOR command precedes any command on the command line, the cursor is moved before the other command is performed. This behavior can be useful in saving cursor movement for commands that are performed repeatedly in one of the windows.
• The CURSOR command is not logged.

Example

Move the cursor between the last saved position on the Debug Tool session panel and the command line.
CURSOR;

Declarations (assembler, disassembly, and OS/VS COBOL)

Use declarations to declare session variables that are effective during a Debug Tool session. Session variables remain in effect for the entire debug session, or process in which they were declared. Variables declared with declarations can be used in other Debug Tool commands as if they were declared to the compiler. Declared variables are removed when your Debug Tool session ends or when the CLEAR command is used to remove them.

```
<table>
<thead>
<tr>
<th>identifier</th>
<th>DS</th>
</tr>
</thead>
<tbody>
<tr>
<td>F, FLn, X, XLn, C, CLn, H, HLn, A, ALn, B, BLn, P, PLn, Z, ZLn, E, D, L</td>
<td></td>
</tr>
</tbody>
</table>
```

**identifier**

A valid assembler identifier.

**F, FLn, X, XLn, C, CLn, H, HLn, A, ALn, B, BLn, P, PLn, Z, ZLn, E, D, L**

Type codes that correspond to the types used in the assembler DC instruction. See the High Level Assembler for MVS & VM & VSE: Language Reference for details about the meaning of these type codes.

Usage note

The range of valid $n$ values depends on the type specifier as follows:
• C and X: 1 to 65525
• F, H, and A: 1 to 4
• B: 1 to 256
• P and Z: 1 to 16
Declarations (C and C++)

Use declarations to declare session variables and tags that are effective during a Debug Tool session. Session variables remain in effect for the entire debug session, or process in which they were declared. Variables and tags declared with declarations can be used in other Debug Tool commands as if they were declared to the compiler. Declared variables and tags are removed when your Debug Tool session ends or when the CLEAR command is used to remove them. The keywords must be the correct case and cannot be abbreviated.

You can also declare enum, struct, and union data types. The syntax is identical to C except that enum members can only be initialized to an optionally signed integer constant.

```
scalar_def:
  char [signed | unsigned | int | long | short | double | float | void]
  enum_def [signed | unsigned | struct_def | union_def]

declarator:
  identifier
  [identifier]
  [identifier] [integer]
```
enum_def:

```plaintext
enum
  identifier
  constant_expr
```

struct_def:

```plaintext
_Packed
struct
  identifier
  enum_def
  scalar_def
  struct_def
  union_def
```

union_def:

```plaintext
_Packed
union
  identifier
  enum_def
  scalar_def
  struct_def
  union_def
```

*  A C indirect operator.

identifier
  A valid C identifier.

integer
  A valid C array bound integer constant.

constant_expr
  A valid C integer constant.

Usage notes
- As in C and C++, the keywords can be specified in any order. For example, `unsigned long int` is equivalent to `int unsigned long`. Some permutations are shown in the syntax diagram to make sure that every keyword is shown at least once in the initial position.
- As in C and C++, the identifiers are case-sensitive; that is, "X" and "x" are different names.
- A structure definition must have either an `identifier`, a `declarator`, or both specified.
- Initialization is not supported.
- A declaration cannot be used in a command list; for example, as the subject of an `if` command or case clause.
- Declarations of the form `struct tag identifier` must have the tag previously declared interactively.
- See the C and C++ Language References for an explanation of the following keywords:
char  short
double signed
enum  struct
float union
int   unsigned
long  void
//Packed(1)

(1) _Packed is not supported in C++.

- You cannot use the declarations command while you replay recorded
  statements by using the PLAYBACK commands by using the PLAYBACK command.

Examples
- Define two C integers.
  ```
  int myvar, hisvar;
  ```
- Define an enumeration variable status that represents the following values:
  ```
  Enumeration Constant   Integer Representation
  run                  0
  create              1
  delete             5
  suspend           6
  ```
  ```
  enum statustag {run, create, delete=5, suspend} status;
  ```
- Define a variable in a struct declaration.
  ```
  struct atag {
    char foo;
    int vari;
  } avar;
  ```
- Interactively declare variables using structure tags.
  ```
  struct tagone {int a; int b;} c;
  ```

then specify:
  ```
  struct tagone d;
  ```

Refer to the following sections for more information related to the material
discussed in this section.

Related tasks
Debug Tool User's Guide

Declarations (COBOL)

Use declarations to declare session variables that are effective during a Debug Tool
session. Session variables remain in effect for the entire debug session, or process
in which they were declared. Variables declared with declarations can be used in
other Debug Tool commands as if they were declared to the compiler. Declared
variables are removed when your Debug Tool session ends or when the CLEAR
command is used to remove them. The keywords cannot be abbreviated.
attribute:

```
| PIC | PICTURE | IS | picture | usage_attribute |
```

usage_attribute:

```
| USAGE | IS | POINTER | BINARY | COMP | COMPUTATIONAL | COMPUTATIONAL-1 | COMPUTATIONAL-2 |
```

*level*

1 or 77.

*identifier*

A valid COBOL data name (including DBCS data names).

*picture*

A sequence of characters from the set: S X 9 (replication factor is optional).

If *picture* is not X(*), the COBOL USAGE clause is required.

**Usage notes**

- A declaration cannot be used in a command list; for example, as the subject of an IF command or WHEN clause.
- BINARY and COMP are equivalent.
- Use BINARY or COMP for COMPUTATIONAL-4.
- COMP-1 is short floating point (4 bytes).
- COMP-2 is long floating point (8 bytes).
- Only COBOL PICTURE and USAGE clauses are supported.
- Short forms of COMPUTATIONAL (COMP) are supported.

**Examples**

- Define a variable named *floattmp* to hold a floating-point number.
  ```
  01 floattmp USAGE COMP-1;
  ```
- Define an integer variable name *temp*.
  ```
  77 temp PIC 9(9) USAGE COMP;
  ```

Refer to the following sections for more information related to the material discussed in this section.

**Related tasks**

- [Debug Tool User's Guide](#)

**Related references**

- [Enterprise COBOL for z/OS Language Reference](#)
The DECLARE command declares session variables that are effective during a Debug Tool session. Variables declared this way can be used in other Debug Tool commands as if they were declared to the compiler. They are removed with the CLEAR command or when your Debug Tool session ends. The keywords cannot be abbreviated.

**major_structure:**

- **level**
  - An unsigned positive integer. Level 1 must be specified for major structure names.

- **name**
  - A valid PL/I identifier. The name must be unique within a particular structure level.

When name conflicts occur, Debug Tool uses session variables before using other variables of the same name that appear in the running programs. Use qualification to refer to the program variable during a Debug Tool session. For example, to display the variable a declared with the DECLARE command as well as the variable a in the program, issue the LIST command as follows:

```
LIST (a, %BLOCK:a);
```

If a name conflict occurs because the variable was declared earlier with a DECLARE command, the new declaration overrides the previous one.

**attribute**

- A PL/I data or storage attribute.

Acceptable PL/I data attributes include:

- BINARY
- BIT
- CHARACTERS
- COMPLEX
- CPLX
- DECIMAL
- EVENT
- FIXED
- FLOAT
- GRAPHIC
- LABEL
- OFFSET
- POINTER
- REAL
- OFFSET
- VARYING
Acceptable PL/I storage attributes include:
  BASED  ALIGNED  UNALIGNED

Pointers cannot be specified with the BASED option.

Only simple factoring of attributes is allowed. DECLAREs such as the following are not allowed:
DCL (a(2), b) PTR;
DCL (x REAL, y CPLX) FIXED BIN(31);

Also, the precision attribute and scale factor as well as the bounds of a dimension can be specified. If a session variable has dimensions and bounds, these must be declared following PL/I language rules.

Usage notes
• DECLARE is not valid as a subcommand. That is, it cannot be used as part of a DO/END or BEGIN/END block.
• Initialization is not supported.
• Program DEFAULT statements do not affect the DECLARE command.
• If you are debugging a Enterprise PL/I program, you can not declare arrays, structures, factor attributes, or multiple session variables in one command line.
• The DECLARE command cannot be used while you replay recorded statements by using the PLAYBACK commands.

Examples
• Declare x, y, and z as variables that can be used as pointers.
  DECLARE (x, y, z) POINTER;
• Declare a as a variable that can represent binary, fixed-point data items of 15 bits.
  DECLARE a FIXED BIN(15);
• Declare d03 as a variable that can represent binary, floating-point, complex data items.
  DECLARE d03 FLOAT BIN COMPLEX;

  This d03 will have the attribute of FLOAT BINARY(21).
• Declare x as a pointer, and setx as a major structure with structure elements a and b as fixed-point data items.
  DECLARE x POINTER, 1 setx, 2 a FIXED, 2 b FIXED;

  This a and b will have the attributes of FIXED DECIMAL(5).

Refer to the following sections for more information related to the material discussed in this section.

Related tasks
Debug Tool User’s Guide

Refer to the following sections for more information related to the material discussed in this section.

Related references
Enterprise PL/I for z/OS Language Reference
DESCRIBE command

The DESCRIBE command displays the file allocations or attributes of references, compile units, known load modules, and the run-time environment.

Notes:
1 Available only with Debug Tool Utilities and Advanced Functions (5655-R45)

CURSOR (Full-Screen Mode only)
Provides a cursor-controlled method for describing variables, structures, and arrays. If you have assigned DESCRIBE to a PF key, you can display the attributes of a selected variable by positioning the cursor at that variable and pressing the assigned PF key.

ALLOCATIONS
Lists the current file allocations.

USER
Indicates that files allocated in the user’s address space are to be described.

ALL
Indicates that both USER and SYSTEM allocations are to be described.

SYSTEM
Indicates that all of the following allocations are to be described.
LINKLIST
Indicates that the current LINKLIB, JOBLIB, STEPLIB, and TASKLIB allocations are to be described.

LPALIST
Indicates that the current LPA list is to be described.

APFLIST
Indicates that the current list of APF authorized data sets is to be described.

CATALOG
Indicates that the current list of active catalogs is to be described.

PARMLIB
Indicates that the current PARMLIB concatenation is to be described.

PROCLIB
Indicates that the current PROCLIB concatenation is to be described.

ATTRIBUTES
Displays the attributes of a specified variable or, in C and C++, an expression.
The attributes are ordinarily those that appeared in the declaration of a variable or are assumed because of the defaulting rules. DESCRIBE ATTRIBUTES works only for variables accessible to the current programming language. All variables in the currently qualified block are described if no operand is specified.

reference
A valid Debug Tool reference in the current programming language. Note the following points:

In C and C++, this can be a valid expression. For a C and C++ expression, the type is the only attribute displayed. For a C and C++ structure or class, DESCRIBE ATTRIBUTES displays only the attributes of the structure or class. To display the attributes of a data object within a structure or data member in a class, use DESCRIBE ATTRIBUTES for the specific data object or member.

In COBOL, this can be any user-defined name appearing in the DATA DIVISION. Names can be subscripted or subtringed per their definitions (that is, if they are defined as alphanumeric data or as arrays).

In PL/I, if the variable is in a structure, it can have inherited dimensions from a higher level parent. The inherited dimensions appear as if they have been part of the declaration of the variable.

In optimized COBOL programs, if reference refers to a variable that was discarded by the optimizer, the address information is replaced with a message.

reference'
A valid Debug Tool OS/VS COBOL reference. This form must be used for OS/VS COBOL. It can contain a simple variable or a variable with IN or OF qualifications.

* Describes all variables in the compile unit. The * is not supported for assembler, disassembly, PL/I, or OS/VS COBOL programs.

CUS
Describes the attributes of compile units, including such things as the compiler options and list of internal blocks. The information returned is dependent on the HLL that the compile unit was compiled under. CUS is equivalent to PROGRAMS.
* Describes all compile units.

**PROGRAMS**

Is equivalent to CUS.

**ENVIRONMENT**

The information returned includes a list of the currently opened files. Names of files that have been opened but are currently closed are excluded from the list. COBOL, assembler, and disassembly do not provide any information for DESCRIBE ENVIRONMENT.

**LOADMODS**

This command displays information about load modules known to Debug Tool and the known or potential CUs in these load modules.

If no operand is specified, the currently active load module is assumed.

* Displays a list of all load modules known to Debug Tool along with the address, length, entry point, and the dataset from which the module was loaded.

  ```plaintext
  load_spec
  ```

Display information about the specified load module or load modules and all known and potential CUs in these load modules. This CU information consists of CSECT name, address, length, and programming language.

**Usage notes**

- You can use the DESCRIBE CUS command in remote debug mode.
- The DESCRIBE ALLOCATIONS command is not available under CICS.
- Cursor pointing can be used by typing the DESCRIBE CURSOR command on the command line and moving the cursor to a variable in the Source window before pressing Enter, or by moving the cursor and pressing a PF key with the DESCRIBE CURSOR command assigned to it.
- When using the DESCRIBE CURSOR command for a variable that is located by the cursor position, the variable's name cannot be split across different lines of the source listing.
- In C, C++, and COBOL, expressions containing parentheses () must be enclosed in another set of parentheses when used with the DESCRIBE ATTRIBUTES command. For example, DESCRIBE ATTRIBUTES ((x + y) / z);
- For COBOL, if DESCRIBE ATTRIBUTES * is specified and your compile unit is large, you might receive an out of storage error message.
- For PL/I, DESCRIBE ATTRIBUTES returns only the top-level names for structures. DESCRIBE ATTRIBUTES * is not supported for PL/I. To get more detail, specify the structure name as the reference.

In order to use DESCRIBE ATTRIBUTES in an Enterprise PL/I program, the PTF for Language Environment APAR PK30522 must be installed on z/OS Version 1 Release 4, Version 1 Release 5, Version 1 Release 6, Version 1 Release 7, and Version 1 Release 8.
- OS/VS COBOL PIC attributes might not match the original PIC specification in the following situations:
  - A COMP-3 variable will always have an odd number of digits in its PIC value.
  - All non-numeric strings will have a PIC value of X's.
- If the DATA option of the PLAYBACK ENABLE command is in effect for the current compile unit, the DESCRIBE ATTRIBUTES and DESCRIBE CURSOR commands can be used while you replay recorded statements by using the PLAYBACK commands.
• The DESCRIBE ENVIRONMENT command cannot be used while you replay recorded statements by using the PLAYBACK commands.
• Information displayed by DESCRIBE LOADMODS does not include information on either load modules or CUs that are ignored by Debug Tool because their names indicated that they were “system” modules (LE, MVS, CICS, etc.)
• CU information displayed by DESCRIBE LOADMODS includes information about the following types of CUs:
  – Known CUs (CUs that appear in LIST NAMES CUS output)
  – Hidden disassembly CUs (If SET DISASSEMBLY OFF is in effect these are the names of the CUs that would be created if you SET DISASSEMBLY ON)
  – Hidden COBOL CUs (COBOL CUs that have not yet been entered)

Examples
• Describe the attributes of argc, argv, boolean, i, ld, and structure.
  DESCRIBE ATTRIBUTES (argc, argv, boolean, i, ld, structure);
• Describe the current environment.
  DESCRIBE ENVIRONMENT;
• Display information describing program myprog.
  DESCRIBE PROGRAMS myprog;

Refer to the following sections for more information related to the material discussed in this section.

Related references
“references” on page 15
“cu_spec” on page 13

DISABLE command

The DISABLE command makes the AT breakpoint inoperative, but does not clear it; you can ENABLE it later without typing the entire command again.

```
>>> DISABLE—AT_command——;
```

**AT_command**

An enabled AT command. The AT command must be complete except that the every_clause and command are omitted. Valid forms are the same as those allowed with CLEAR AT.

Usage notes
• You can use the DISABLE command to disable either active or suspended breakpoints. However, you cannot use it to disable suspended label breakpoints.
• If you want to disable a suspended breakpoint, you must specify both the load module and CU name.
• To reenable a disabled AT command, use the ENABLE command.
• Disabling an AT command does not affect its replacement by a new (enabled) version if an overlapping AT command is later specified. It also does not prevent removal by a CLEAR AT command.
• Breakpoints already disabled within the range(s) specified in the specific AT command are unaffected; however, a warning message is issued for any specified range found to contain no enabled breakpoints.
The DISABLE command cannot be used while you replay recorded statements by using the PLAYBACK commands.

Examples
- Disable the breakpoint that was set by the command AT ENTRY myprog CALL proc1;
  DISABLE AT ENTRY myprog;
- If statement 25 is in a loop and you set the following breakpoint:
  AT EVERY 5 FROM 1 TO 100 STATEMENT 25 LIST x;
  to disable it, enter:
  DISABLE AT STATEMENT 25;

You do not need to reenter the every_clause or the command list. To restore the breakpoint, enter:
  ENABLE AT STATEMENT 25;

Refer to the following sections for more information related to the material discussed in this section.

Related references
- "ENABLE command" on page 101
- "DISABLE prefix (full-screen mode)"

DISABLE prefix (full-screen mode)
Disables a statement breakpoint or offset breakpoint when you issue this command through the Source window prefix area.

```
DISABLE [integer];
```

**integer**
Selects a relative statement (for C and C++ or PL/I) or a relative verb (for COBOL) within the line. The default value is 1.

Example
Disable the breakpoint at the third statement or verb in the line by entering the following command in the prefix area of the line where the statement is found.

```
DIS 3
```

You do not need to enter a space between the keyword and the integer: DIS 3 is equivalent to DIS3.

Refer to the following sections for more information related to the material discussed in this section.

Related tasks
- Debug Tool User’s Guide
DO command (assembler, disassembly, and OS/VS COBOL)

The DO command performs one or more commands that are collected into a group. The DO and END keywords delimit a group of commands called a DO group. The keywords cannot be abbreviated.

```
  DO ; END ;
```

*command*

A valid Debug Tool command.

do/while command (C and C++)

The do/while command performs a command before evaluating the test expression. Due to this order of execution, the command is performed at least once. The do and while keywords must be lowercase and cannot be abbreviated.

```
  do--command--while(--expression--)--;
```

*command*

A valid Debug Tool command.

*expression*

A valid Debug Tool C and C++ expression.

The body of the loop is performed before the while clause (the controlling part) is evaluated. Further execution of the do/while command depends on the value of the while clause. If the while clause does not evaluate to false, the command is performed again. Otherwise, execution of the command ends.

A break command can cause the execution of a do/while command to end, even when the while clause does not evaluate to false.

Usage note

The do/while command cannot be used while you replay recorded statements by using the PLAYBACK commands by using the PLAYBACK command.

Example

The following command prompts you to enter a 1. If you enter a 1, the command ends execution. Otherwise, the command displays another prompt.

```c
  int reply1;
  do {
    printf("Enter a 1.\n");
    scanf("%d", &reply1);
  } while (reply1 != 1);
```
DO command (PL/I)

The DO command allows one or more commands to be collected into a group that can (optionally) be repeatedly executed. The DO and END keywords delimit a group of commands collectively called a DO group. The keywords cannot be abbreviated.

**Simple**

```
DO; command; END;
```

*command*  
A valid Debug Tool command.

**Repeating**

```
DO WHILE(expression) UNTIL(expression) ;
END;
```

*WHILE*  
 Specifies that *expression* is evaluated before each execution of the command list. If the expression evaluates to true, the commands are executed and the DO group begins another cycle; if it evaluates to false, execution of the DO group ends.

*expression*  
A valid Debug Tool PL/I Boolean expression.

*UNTIL*  
 Specifies that *expression* is evaluated after each execution of the command list. If the expression evaluates to false, the commands are executed and the DO group begins another cycle; if it evaluates to true, execution of the DO group ends.

*A valid Debug Tool command."

**Iterative**

```
DO reference = iteration; command; END;
```

*reference*  
A valid Debug Tool command.
iteration:

```
expression
   BY expression
   TO expression
   REPEAT expression

WHILE (expression)
UNTIL (expression)
UNTIL (expression)
WHILE (expression)
```

reference
  A valid Debug Tool PL/I reference.

expression
  A valid Debug Tool PL/I expression.

BY
  Specifies that *expression* is evaluated at entry to the DO specification and saved. This saved value specifies the increment to be added to the control variable after each execution of the DO group.
  
  If BY *expression* is omitted from a DO specification and if TO *expression* is specified, *expression* defaults to the value of 1.
  
  If BY 0 is specified, the execution of the DO group continues indefinitely unless it is halted by a WHILE or UNTIL option, or control is transferred to a point outside the DO group.
  
  The BY option allows you to vary the control variable in fixed positive or negative increments.

TO
  Specifies that *expression* is evaluated at entry of the DO specification and saved. This saved value specifies the terminating value of the control variable.
  
  If TO *expression* is omitted from a DO specification and if BY *expression* is specified, repetitive execution continues until it is terminated by the WHILE or UNTIL option, or until some statement transfers control to a point outside the DO group.
  
  The TO option allows you to vary the control variable in fixed positive or negative increments.

REPEAT
  Specifies that *expression* is evaluated and assigned to the control variable after each execution of the DO group. Repetitive execution continues until it is terminated by the WHILE or UNTIL option, or until some statement transfers control to a point outside the DO group.
  
  The REPEAT option allows you to vary the control variable nonlinearly. This option can also be used for nonarithmetic control variables, such as pointers.

WHILE
  Specifies that *expression* is evaluated before each execution of the command list. If the expression evaluates to true, the commands are executed and the DO group begins another cycle; if it evaluates to false, execution of the DO group ends.
UNTIL

Specifies that expression is evaluated after each execution of the command list. If the expression evaluates to false, the commands are executed and the DO group begins another cycle; if it evaluates to true, execution of the DO group ends.

command
A valid Debug Tool command.

Usage note
You cannot use the DO command while you replay recorded statements by using the PLAYBACK commands by using the PLAYBACK command.

Examples
• At statement 25, initialize variable a and display the values of variables x, y, and z.
  AT 25 DO; %BLOCK:>a = 0; LIST (x, y, z); END;
• Execute the DO group until ctr is greater than 4 or less than 0.
  DO UNTIL (ctr > 4) WHILE (ctr >= 0); END;
• Execute the DO group with i having the values 1, 2, 4, 8, 16, 32, 64, 128, and 256.
  DO i = 1 REPEAT 2*i UNTIL (i = 256); END;
• Repeat execution of the DO group with j having values 1 through 20, but only if k has the value 1.
  DO j = 1 TO 20 BY 1 WHILE (k = 1); END;

ENABLE command

The ENABLE command makes the AT breakpoints operative after they have been disabled.

 ENABLE—AT_command—;

AT_command
A disabled AT command. The AT command must be complete except that the every_clause and command are omitted. Valid forms are the same as those allowed with CLEAR AT.

Usage notes
• You can use the ENABLE command to enable either active or suspended breakpoints. However, you cannot use it to enable suspended label breakpoints.
• If you want to enable a suspended breakpoint, you must specify both the load module and CU name.
• To disable an AT command, use the DISABLE command.
• Breakpoints already enabled within the range(s) specified in the specific AT command are unaffected; however, a warning message is issued for any specified range found to contain no disabled breakpoints.
• The ENABLE command cannot be used while you replay recorded statements by using the PLAYBACK commands.

Example
• Reenable the previously disabled command AT ENTRY mysub CALL proc1;.
ENABLE AT ENTRY mysub;

Refer to the following sections for more information related to the material discussed in this section.

Related references
“DISABLE prefix (full-screen mode)” on page 97
“ENABLE prefix (full-screen mode)”

ENABLE prefix (full-screen mode)
Enables a disabled statement breakpoint or a disabled offset breakpoint when you issue this command through the Source window prefix area.

ENABLE [integer];

integer
Selects a relative statement (for C and C++ or PL/I) or a relative verb (for COBOL) within the line. The default value is 1. For optimized COBOL programs, the default value is the first executable statement which was not discarded by the optimizer.

Example
Enable the breakpoint at the third statement or verb in the line (typed in the prefix area of the line where the statement is found).

ENABLE 3

No space is needed as a delimiter between the keyword and the integer; hence, ENABLE 3 is equivalent to ENABLE3.

EVALUATE command (COBOL)
The EVALUATE command provides a shorthand notation for a series of nested IF statements. The keywords cannot be abbreviated.

EVALUATE constant expression reference TRUE FALSE WHEN any_clause command END-EVALUATE;

WHEN OTHER command
any_clause:

```
<table>
<thead>
<tr>
<th>ANY</th>
<th>condition</th>
<th>TRUE</th>
<th>FALSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>constant</td>
<td>THROUGH</td>
<td>constant</td>
</tr>
<tr>
<td></td>
<td>reference</td>
<td>THRU</td>
<td>reference</td>
</tr>
</tbody>
</table>
```

constant

A valid Debug Tool COBOL constant.

equation

A valid Debug Tool COBOL arithmetic expression.

reference

A valid Debug Tool COBOL reference.

condition

A simple relation condition.

command

A valid Debug Tool command.

Usage notes

- Only a single subject is supported.
- Consecutive WHENs without associated commands are not supported.
- THROUGH/THRU ranges can be specified as constants or references.
- See [Enterprise COBOL for z/OS Language Reference](#) for an explanation of the following COBOL keywords:
  - ANY
  - FALSE
  - NOT
  - OTHER
  - THROUGH
  - THRU
  - TRUE
  - WHEN

- Debug Tool implements the EVALUATE command as a series of IF commands.
- If the DATA option of the PLAYBACK ENABLE command is in effect for the current compile unit, the EVALUATE command can be used while you replay recorded statements by using the PLAYBACK commands.
- For optimized COBOL programs, the value of reference cannot refer to any variables discarded by the optimizer.
- If a COBOL variable is defined as National and it is an operand in a relation condition with an alphabetic, alphanumeric operand, or National numeric, the operand that is not National is converted to Unicode before that comparison is done, except for Group items. See [Enterprise COBOL for z/OS Language Reference](#) for more information about using COBOL variables in conditional expressions.

Example

The following example shows an EVALUATE command and the equivalent coding for an IF command:
EVALUATE menu-input
  WHEN "0"
    CALL init-proc
  WHEN "1" THRU "9"
    CALL process-proc
  WHEN "R"
    CALL read-parms
  WHEN "X"
    CALL cleanup-proc
  WHEN OTHER
    CALL error-proc
END-EVALUATE;

The equivalent IF command:
IF (menu-input = "0") THEN
  CALL init-proc
ELSE
  IF (menu-input >= "1") AND (menu-input <= "9") THEN
    CALL process-proc
  ELSE
    IF (menu-input = "R") THEN
      CALL read-parms
    ELSE
      IF (menu-input = "X") THEN
        CALL cleanup-proc
      ELSE
        CALL error-proc
      END-IF;
    END-IF;
  END-IF;
END-IF;
END-IF;
END-IF;

Refer to the following sections for more information related to the material discussed in this section.

Related references
- "Allowable comparisons for the IF command (COBOL)" on page 116
- Enterprise COBOL for z/OS Language Reference

Expression command (C and C++)

The Expression command evaluates the given expression. The expression can be used to either assign a value to a variable or to call a function.

expression

expression
  A valid Debug Tool C and C++ expression. Assignment is affected by including one of the C and C++ assignment operators in the expression. No use is made of the value resulting from a stand-alone expression.

Usage notes
- Function invocations in expressions are restricted to functions contained in the currently executing enclave.
- The Expression command cannot be used while you replay recorded statements by using the PLAYBACK commands.

Examples
- Initialize the variables x, y, z. You can use functions to provide values for variables.
\[
\begin{align*}
x &= 3 + 4/5; \\
y &= 7; \\
z &= 8 \times \text{func}(x, y);
\end{align*}
\]
- Increment \(y\) and assign the remainder of the integer division of \(\omega\) by 4 to \(\alpha\).
  \[
  \alpha = (y++, \omega \div 4);
  \]
- To list and assign a new value to \(R1\) in the disassembly view:
  \[
  \text{LIST}(R1); \\
  R1 = 0x0001FAF0;
  \]

**FIND command**

The *FIND* command provides full-screen, line, and batch mode search capability in the source object, and full-screen searching of log and monitor objects.

```
\text{FIND} \quad \text{string} \quad ;
```

**string**

The string you want to find, which conforms to the syntax for a character string constant of the current programming language. The string must comply with the following restrictions:

- If the string contains spaces, or is an asterisk ("\*"), a question mark ("?"), or a semicolon (";"), it must be enclosed in quotes.
- The length of the string cannot exceed 128 bytes.

If no operands are specified, a repeat FIND is performed. The usage notes and [Debug Tool User's Guide](#) describes repeat FIND.

* Use the string from the previous FIND command.

**FIRST**

Starts at the beginning of the object and searches forward to find the first occurrence of the string.

**LAST**

Starts at the end of the object and searches backward to find the last occurrence of the string.

**NEXT**

Starts at the first position after the current cursor location and searches forward to find the next occurrence of the string.

**PREV**

Starts at the current cursor location and searches backward to find the previous occurrence of the string.

**CURSOR (Full-Screen Mode)**

Specifies that the current cursor position selects the object searched.

**LOG (Full-Screen Mode)**

Selects the object in the session log window.

**MONITOR (Full-Screen Mode)**

Selects the object in the monitor window.
**SOURCE (Full-Screen Mode)**

Selects the object in the source listing window.

**Usage notes**

- If no operands are specified, a repeat FIND is performed. A repeat FIND behaves in the following ways:
  - The string from the previous FIND that you entered is used.
  - If no FIND string has been previously specified, Debug Tool displays an error message.
  - If the previous FIND command that you entered specified or implied the FIRST or NEXT parameter, Debug Tool uses the NEXT parameter.
  - If the previous FIND command that you entered specified or implied the LAST or PREV parameter, Debug Tool uses the PREV parameter.
  - If a repeat FIND immediately follows an unsuccessful FIND or repeat FIND, Debug Tool continues searching, wrapping from the last line to the first line.
  - If the original direction of the FIND was backward to the beginning of the object, Debug Tool wraps from the first line to the last line.
  - If the cursor is not in a window, Debug Tool uses the same window that was used for the previous FIND command.
- In full-screen mode, Debug Tool chooses the window it searches through in the following ways:
  - If you specify a string and you do not place the cursor in a window nor specify an object on the command, Debug Tool searches the object in the window specified by the SET DEFAULT WINDOW command or the Default window entry in your Profile Settings panel.
  - If you place the cursor in a window and do not specify a different window on the command, Debug Tool searches the object in the window where you placed the cursor.
  - If you specify a string without a direction keyword, forward is the default direction.
  - FIND can be made immediately effective in full-screen mode with the IMMEDIATE command.
  - If the current programming language setting is C or C++, the search is case-sensitive. Otherwise, the search is not case-sensitive.
- In full-screen mode, searches shows the following behavior:
  - If you specify FIRST, the search begins at the beginning of the first line of the object.
  - If you specify LAST, the search begins at the end of the last line of the object.
  - If you specify NEXT or the command defaults to NEXT and the cursor is within the window for the object being searched, the search begins at the first position after the current cursor location.
  - If you specify NEXT or the command defaults to NEXT and the cursor is outside the window for the object being searched, the search begins at the beginning of the first line displayed in the window.
  - If you specify PREV or the command defaults to PREV and the cursor is within the window for the object being searched, the search begins at the current cursor location.
  - If you specify PREV or the command defaults to PREV and the cursor is outside the window for the object being searched, the search begins at the end of the line preceding the first line displayed in the window of the object being
searched. If the beginning of the object is displayed, Debug Tool wraps to the end of the object and continues from the end of the last line in the object.

- If Debug Tool finds the string, the window for the object being searched is scrolled until the string is visible. If the string is DBCS, it is displayed without alteration. If the string is not DBCS, the string is highlighted as specified by the SET COLOR command and the cursor is placed at the beginning of the string.

- If Debug Tool does not find the string, the screen does not change and the cursor is not moved. If you specified NEXT or PREV or the command defaults to NEXT or PREV and Debug Tool searched only part of the object, then Debug Tool displays the message ‘Bottom of data is reached’ or ‘Top of data reached’, as appropriate. If Debug Tool searched through the entire object, then it displays the message ‘Search target not found’.

- In batch mode, searches show the following behavior:
  - If you specify FIRST, the search begins at the beginning of the first line of the source.
  - If you specify LAST, the search begins at the end of the last line of the source.
  - If you specify NEXT or the command defaults to NEXT, Debug Tool begins searching at the first character of the first line of the source or, if a previous FIND command was done in the same compile unit, at the location after the last string that was successfully found by a FIND command.
  - If you specify PREV or the command defaults to PREV, Debug Tool begins searching at the last character of the last line of the source, or if a previous FIND command was done in the same compile unit, at the location before the last string that was successfully found by a FIND command.
  - If you specify NEXT or PREV or the command defaults to NEXT or PREV and Debug Tool searched only part of the source and did not find the string, then Debug Tool displays the message ‘Bottom of data is reached’ or ‘Top of data is reached’, as appropriate. If Debug Tool searched through the entire source without finding the string, then it displays the message ‘Search target not found’.
  - If Debug Tool finds the string, the line that contains the string is displayed and marked with a vertical bar character (|) beneath the string.

- The full-screen FIND command is not logged; however, the FIND command is logged in batch mode.
- If you are searching for strings with trigraphs in them, the trigraphs or their equivalents can be used as input, and Debug Tool matches them to trigraphs or their equivalents.
- If you are searching in the monitor window and SET MONITOR WRAP OFF is in effect, Debug Tool will search all of the scrolled data.
- If you enclose the string in quotation marks, use the following rules:
  - For PL/I, use single quotes.
  - For C and C++, use double quotes.
  - For COBOL, assembler, or disassembly, use single or double quotes.

Table 4.

<table>
<thead>
<tr>
<th>C</th>
<th>C++</th>
<th>COBOL or OS/VS COBOL</th>
<th>Assembler or disassembly</th>
<th>PL/I</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ABC&quot;</td>
<td>&quot;IntLink:::&quot;</td>
<td>&quot;A5&quot; or 'A5'</td>
<td>'ABC' or &quot;ABC&quot; or C'ABC'</td>
<td>'ABC'</td>
</tr>
</tbody>
</table>
• If you are searching for a string that contains a quote, you might have to balance the quotes. Use the following rules to determine how to specify the search string:
  – For PL/I, if the string you are searching for has a single quote, you must add a single quote immediately following the single quote. If the string contains a space, surround the entire string with single quotes.
  – For C and C++, if the string you are searching for has a double quote, you must add a double quote immediately following the double quote. If the string contains a space, surround the entire string with double quotes.
  – For COBOL, if the string you are searching for has a double quote, you must add a double quote immediately following the double quote. If the string contains a space, you do not have to balance the double quotes; however, you must surround the entire string with a double quotes or single quotes.
  – For assembler or disassembly, if the string you are searching for has a single quote, you must add a single quote immediately following the single quote. If the string contains a space, you do not have to balance the single quote; however, you must surround the entire string with a double quotes or single quotes.

Examples
• Indicate that you want to search the monitor window for the name myvar.
  
  FIND myvar MONITOR;
• If you want to search the Source window for the next occurrence of var1, just enter:
  
  FIND

  You do not need to provide the variable name, because the Debug Tool remembers the string you last searched for. Again, the Source window is scrolled forward, var1 is highlighted, and the cursor points to the variable.
• If you want to find a question mark (?) in the Source window and you are debugging a PL/I program, enter the following command:
  
  FIND '?' ;
• If you want to find the string User's in the Source window and you are debugging a PL/I program, enter the following command:
  
  FIND 'User'\'s ;
• If you want to find the string User's in the Source window and you are debugging a C program, enter the following command:
  
  FIND User's ;
• If you want to find the string User's Guide in the Source window and you are debugging a PL/I program, enter the following command:
  
  FIND 'User\'s Guide' ;
• If you want to find the string User's Guide in the Source window and you are debugging a C program, enter the following command:
  
  FIND "User\'s Guide" ;
• If you entered the command FIND xyz LAST; or FIND xyz PREV; and the cursor is on the found string ("xyz"), then press the PF key assigned to the FIND command to repeat the search. Debug Tool runs the command FIND xyz PREV;.
• If you entered the command FIND xyz; Debug Tool searches in the forward direction. To find the string "xyz" in the backward direction, enter the command FIND * PREV;.
for command (C and C++)

The for command provides iterative looping similar to the C and C++ for statement. It enables you to do the following:
- Evaluate an expression before the first iteration of the command ("initialization").
- Specify an expression to determine whether the command should be performed again ("controlling part").
- Evaluate an expression after each iteration of the command.
- Perform the command, or block, if the controlling part does not evaluate to false.

The for keyword must be lowercase and cannot be abbreviated.

```plaintext
for(expression; expression; expression) {
  command;
}
```

- **expression**
  - A valid Debug Tool C and C++ expression.
- **command**
  - A valid Debug Tool command.

Debug Tool evaluates the first expression only before the command is performed for the first time. You can use this expression to initialize a variable. If you do not want to evaluate an expression before the first iteration of the command, you can omit this expression.

Debug Tool evaluates the second expression before each execution of the command. If this expression evaluates to false, the command does not run and control moves to the command following the for command. Otherwise, the command is performed. If you omit the second expression, it is as if the expression has been replaced by a nonzero constant and the for command is not terminated by failure of this expression.

Debug Tool evaluates the third expression after each execution of the command. You might use this expression to increase, decrease, or reinitialize a variable. If you do not want to evaluate an expression after each iteration of the command, you can omit this expression.

A break command can cause the execution of a for command to end, even when the second expression does not evaluate to false. If you omit the second expression, you must use a break command to stop the execution of the for command.

**Usage notes**
- The for command cannot be used while you replay recorded statements by using the PLAYBACK commands.

**Examples**
- The following for command lists the value of count 20 times. The for command initially sets the value of count to 1. After each execution of the command, count is incremented.
  ```plaintext
  for (count = 1; count <= 20; count++)
    LIST TITLED count;
  ```
Alternatively, the preceding example can be written with the following sequence of commands to accomplish the same task.

```c
count = 1;
while (count <= 20) {
   printf("count = %d\n", count);
   count++;
}
```

- The following for command does not contain an initialization expression.

```c
for (; index > 10; --index) {
   varlist[index] = var1 + var2;
   printf("varlist[%d] = %d\n", index, varlist[index]);
}
```

**FREE command**

The `FREE` command frees a file that is currently allocated.

```c
(1)
FREE FILE-ddname--;
```

**Notes:**

1. Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

**ddname**

Name of the file to free.

**GO command**

The `GO` command causes Debug Tool to start or resume running your program.

```c
GO BYPASS--;
```

**BYPASS**

Bypasses the user or system action for the AT-condition that caused the breakpoint. It is valid only when Debug Tool is entered for an:

- AT CALL breakpoint
- HLL or Language Environment condition
- Condition raised by an MVS or CICS ABEND when running without the Language Environment run time

**Usage notes**

- If `GO` is specified in a command list (for example, as the subject of an IF command or WHEN clause), all subsequent commands in the list are ignored.
- If `GO` is specified within the body of a loop, it causes the execution of the loop to end.
- To suppress the logging of `GO` commands, use the `SET ECHO` command.
- `GO` with no operand specified does not actually resume the program if there are additional AT-conditions that have not yet been processed.
- The `GO` command cannot be used while you replay recorded statements by using the PLAYBACK commands by using the PLAYBACK command.

**Examples**
• Resume execution.
  GO;
• Resume execution and bypass user and system actions for the AT-condition that
  caused the breakpoint.
  GO BYPASS;
• Your application has ended with a protection exception, so an OCCURRENCE
  breakpoint has been triggered. Correct the results of the instruction that caused
  the exception and issue GO BYPASS; to continue processing as if the abend had
  not occurred.

Refer to the following sections for more information related to the material
discussed in this section.

Related references
"AT command" on page 37

GOTO command

The GOTO command causes Debug Tool to resume program execution at the
specified statement id. The GOTO keyword cannot be abbreviated. If you want
Debug Tool to return control to you at a target location, make sure there is a
breakpoint at that location.

```
GOTO statement_id;
GO TO
```

Usage notes
• You cannot use the GOTO command while you debug a disassembled program.
• If GOTO is specified in a command list (for example, as the subject of an IF
  command or WHEN clause), all subsequent commands in the list are ignored.
• Statement GOTO's are not restricted if the program is compiled with minimum
  optimization.
• The GOTO command cannot be used while you replay recorded statements by
  using the PLAYBACK command.
• For C, C++, and PL/I, statements can be removed by the compiler during
  optimization, specify a reference or statement with the GOTO command that can
  be reached during program execution. You can issue the LIST STATEMENT
  NUMBERS command to determine the reachable statements.
• PL/I allows GOTO in a command list on a call to PLTEST or CEETEST.
• In PL/I, out-of-block GOTOs are allowed. However, qualification might be needed.
• For COBOL, the GOTO command follows the COBOL language rules for the GOTO
  statement. You can use the GOTO command only if you compiled your program
  with either PATH or ALL suboption and the SYM suboption of the TEST
  compiler option.
• If you are debugging an optimized COBOL program, you cannot use the GOTO
  command.
• You can use the GOTO command to debug COBOL programs compiled with the
  NONE suboption of the TEST compiler option and the following compilers:
  - Enterprise COBOL for z/OS and OS/390, Version 3 Release 2
  - Enterprise COBOL for z/OS and OS/390, Version 3 Release 1, with APAR
    PQ63235 installed
  - COBOL for OS/390 & VM, Version 2, with APAR PQ63234 installed
Examples
- Resume execution at statement 23, where statement 23 is in a currently active block.
  
  GOTO 23;

  If there’s no breakpoint at statement 23, Debug Tool will run from statement 23 until a breakpoint is hit.
- Resume execution at statement 45, where statement 45 is in a currently active block.
  
  AT 45
  GOTO 45

Refer to the following sections for more information related to the material discussed in this section.

Related tasks
- Debug Tool User’s Guide

Related references
- “statement_id” on page 16

GOTO LABEL command

The GOTO LABEL command causes Debug Tool to resume program execution at the specified statement label. The specified label must be in the same block. If you want Debug Tool to return control to you at the target location, make sure there is a breakpoint at that location.

```
GOTO 'statement_label';
```

*statement_label*

A valid statement label within the currently executing program or, in PL/I, a label variable.

Usage notes
- Use the syntax of *statement_label* enclosed in single-quotes only for OS/VS COBOL programs. It is not supported in any other programming language.
- In PL/I, out-of-block GOTOS are allowed. However, qualification might be needed.
- The LABEL keyword is optional when either the target *statement_label* is nonnumeric or if it is qualified (whether the actual label was nonnumeric or not).
- A COBOL *statement_label* can have either of the following forms:
  - name
    This form can be used in COBOL for reference to a section name or for a COBOL paragraph name that is not within a section or is in only one section of the block.
  - name1 OF name2 or name1 IN name2
    This form must be used for any reference to a COBOL paragraph (name1) that is within a section (name2), if the same name also exists in other sections in the same block. You can specify either OF or IN, but Debug Tool always uses OF for output.

Either form can be prefixed with the usual block, compile unit, and load module qualifiers.
• For C, to be able to use the GOTO LABEL command, you must compile your
  program in one of the following ways:
  – With either the PATH or ALL suboption and the SYM suboption of the TEST
    compiler option. The default is PATH.
  – With either the PATH or ALL suboption and the SYM suboption of the DEBUG
    compiler option.

  There are no restrictions on using labels with the GOTO LABEL command.

• For C++, to be able to use the GOTO LABEL command, you must compile your
  program in one of the following ways:
  – With the TEST compiler option.
  – With either the PATH or ALL suboption and the SYM suboption of the DEBUG
    compiler option.

  There are no restrictions on using labels with the GOTO LABEL command.

• For COBOL, you can use GOTO LABEL only if you compiled your program with
  either PATH or ALL suboption and the SYM suboption of the TEST compiler option.
  The label itself can take either of two forms:
  – name, where name is a section name, or the name of a paragraph not within a
    section or in only one section of the block.
  – name1 OF name2 or name1 IN name2, where name1 is duplicated by one or more
    other paragraphs in one or more other sections in the block. You can use
    either OF or IN, but Debug Tool always logs OF.

• For PL/I, you can use GOTO LABEL only if you compiled your program with
  either the PATH or ALL suboption and the SYM suboption of the TEST compiler option.
  There are no restrictions on using labels with GOTO LABEL and label
  variables are supported.

• GOTO LABEL is not available while debugging Enterprise PL/I programs.

• You cannot use the GOTO LABEL command while you are replaying recorded steps
  by using the PLAYBACK commands.

• You cannot use the GOTO LABEL command while you debug an optimized
  COBOL program.

Examples
• Go to the label constant 1aba in block suba in program prog1.
  GOTO progl:suba:1aba;

• Go to the label constant para 0F sect1. The current programming language
  setting is COBOL.
  GOTO LABEL para 0F sect1;

Refer to the following sections for more information related to the material
  discussed in this section.

  Related tasks
  Debug Tool User's Guide

  Related references
  "statement_label" on page 17

IF command (assembler, disassembly, and OS/VS COBOL)

The IF command lets you conditionally perform a command. You can optionally
specify an ELSE clause on the IF command. If the test expression evaluates to false
and the ELSE clause exists, the command associated with the ELSE clause is
performed. The IF and ELSE keywords cannot be abbreviated.
**if command (C and C++)**

The if command lets you conditionally perform a command. You can optionally specify an else clause on the if command. If the test expression evaluates to false and an else clause exists, the command associated with the else clause is performed. The if and else keywords must be lowercase and cannot be abbreviated.

\[
\text{if} \left\{ \begin{array}{l}
\text{expression} \\
\text{command}
\end{array} \right\} ;
\text{[else} \begin{array}{l}
\text{command}
\end{array} \right\}
\]

**expression**

A valid Debug Tool C and C++ expression.

**command**

A valid Debug Tool command.

When if commands are nested and else clauses are present, a given else is associated with the closest preceding if clause within the same block.

**Usage notes**
• An else clause should always be included if the if clause causes Debug Tool to get more input (for example, an if containing use or other commands that cause Debug Tool to be restarted because an AT-condition occurs).
• The if command cannot be used while you replay recorded statements by using the playback commands by using the playback command.

Examples
• The following example causes grade to receive the value "A" if the value of score is greater than or equal to 90.
  if (score >= 90)
    grade = "A";
• The following example shows a nested if command.
  if (paygrade == 7) {
    if (level >= 0 && level <= 8)
      salary *= 1.05;
    else
      salary *= 1.04;
  }
else
  salary *= 1.06;

IF command (COBOL)
The IF command lets you conditionally perform a command. You can optionally specify an ELSE clause on the IF command. If the test expression evaluates to false and an ELSE clause exists, the command associated with the ELSE clause is performed. The keywords cannot be abbreviated.

condition
  A simple relation condition.
command
  A valid Debug Tool command.

When IF commands are nested and ELSE clauses are present, a given ELSE or END-IF is associated with the closest preceding IF clause within the same block.

Unlike COBOL, Debug Tool requires terminating punctuation (;) after commands. The END-IF keyword is required.

Usage notes
• An ELSE clause should always be included if the IF clause causes Debug Tool to get more input (for example, an IF containing USE or other commands that cause Debug Tool to be restarted because an AT-condition occurs).
• The COBOL NEXT SENTENCE phrase is not supported.
• Comparison combinations with windowed date fields are not supported.
• Comparisons between expanded date fields with different DATE FORMAT clauses are not supported.
If the DATA option of the PLAYBACK ENABLE command is in effect, the IF command can be used while you replay recorded statements by using the PLAYBACK commands.

For optimized COBOL programs, the IF clause cannot reference any variables discarded by the optimizer.

If a COBOL variable is defined as National and it is an operand in a relation condition with an alphabetic, alphanumeric operand, or National numeric, the operand that is not National is converted to Unicode before that comparison is done, except for Group items. See Enterprise COBOL for z/OS Language Reference for more information about using COBOL variables in conditional expressions.

Refer to the following sections for more information related to the material discussed in this section.

Related references

“Allowable comparisons for the IF command (COBOL)”

Allowable comparisons for the IF command (COBOL)

The following table shows the allowable comparisons for the Debug Tool IF command. A description of the codes follows the table.
<table>
<thead>
<tr>
<th>OPERAND</th>
<th>GR</th>
<th>AL</th>
<th>AN</th>
<th>ED</th>
<th>BI</th>
<th>NE</th>
<th>ANE</th>
<th>NDI</th>
<th>NNDI</th>
<th>ID</th>
<th>IN</th>
<th>IDI</th>
<th>PTR</th>
<th>@</th>
<th>IF</th>
<th>EF</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group (GR)</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN(^1)</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alphabetic (AL)</td>
<td>NN</td>
<td>NN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alphanumeric (AN)(^2)</td>
<td>NN</td>
<td>NN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>External Decimal (ED)(^3)</td>
<td>NN</td>
<td></td>
<td>NU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary</td>
<td>NN</td>
<td></td>
<td>NU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Numeric Edited (NE)</td>
<td>NN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alphanumeric Edited (ANE)</td>
<td>NN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIGCON ZERO(^7)</td>
<td>NN</td>
<td></td>
<td>NU</td>
<td>NU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIGCON(^1,7)</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>National Data Item (NDI)</td>
<td>NN(^10)</td>
<td>NN</td>
<td>NN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>National Numeric Data Item (NNDI)</td>
<td>NN(^10)</td>
<td>NN</td>
<td>NN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Numeric Literal(^2)</td>
<td>NN</td>
<td></td>
<td>NU</td>
<td>NU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alphanumeric Literal(^2)</td>
<td>NN</td>
<td>NN(^2)</td>
<td>NN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alphanumeric hex literal(^2)</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal Decimal (ID)(^4)</td>
<td>NN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index Name (IN)</td>
<td>NN</td>
<td></td>
<td></td>
<td>NU(^3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index Data Item (IDI)</td>
<td>NN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pointer Data Item (PTR)</td>
<td>NN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Chapter 5: Debug Tool commands

Page 117
| OPERAND                          | GR | AL | AN | ED | BI | NE | ANE | NDI | NNDI | ID  | IN | IDI | PTR | @ | IF | EF | D |
|----------------------------------|----|----|----|----|----|----|-----|-----|------|-----|----|----|-----|----|---|---|---|---|
| Address of (@)                   |    |    |    |    |    |    |     |     |      |     |    |    |     |    |   |   |   |   |
| Floating Point Literal\(^7\)     | X  |    |    |    |    |    |     |     |      |     |    |    |     |    |   |   |   |   |
| Internal Floating Point (IF)     | NN |    |    |    |    |    |     |     |      |     |    |    |     |    |   |   |   |   |
| External Floating Point (EF)     | NN |    |    |    |    |    |     |     |      |     |    |    |     |    |   |   |   |   |
| DBCS data item (DI)              |    |    |    |    |    |    |     |     |      |     |    |    |     |    |   |   |   |   |
| DBCS Literal\(^7\)              |    |    |    |    |    |    |     |     |      |     |    |    |     |    |   |   |   |   |
| Address hex Literal\(^b\)       |    |    |    |    |    |    |     |     |      |     |    |    |     |    |   |   |   |   |
| National Literal                | NN\(^10\) |    |    |    |    |    |     |     |      |     |    |    |     |    |   |   |   |   |
| National Hex Literal\(^12\)     | NN\(^10\) |    |    |    |    |    |     |     |      |     |    |    |     |    |   |   |   |   |

\(^7\) Floating Point Literal
\(^b\) Address hex Literal
\(^10\) National Literal
\(^12\) National Hex Literal
Notes:
1 FIGCON includes all figurative constants except ZERO and ALL.
2 A alphanumeric literal must be enclosed in quotation marks, and the quotation marks are not valid characters in the literal.
3 Must contain only alphabetic characters.
4 Index name converted to subscript value before compare.
5 Only comparison for equal and not equal can be made.
6 Must be hexadecimal characters only, delimited by either double (”) or single (’) quotation marks and preceded by H.
7 Constants and literals can also be compared against constants and literals of the same type.
8 Comparisons using windowed date fields are not supported.
9 The figurative constants HIGH-VALUES and LOW-VALUES are not allowed in comparisons with national data items.
10 Conversion of internal format is not done before the comparison.
11 Must be hexadecimal characters only, delimited by either double (”) or single (’) quotation marks and preceded by X.
12 Must be hexadecimal characters only, delimited by either double (”) or single (’) quotation marks and preceded by NX.

Allowable comparisons are comparisons as described in *IBM OS Full American National Standard COBOL* for the following:

| NN | Nonnumeric operands |
| NU | Numeric operands    |
| IO | Two index names     |
| IV | Index data items    |
| X  | High potential for user error |

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

*IBM OS Full American National Standard COBOL*

**IF command (PL/I)**

The IF command lets you conditionally perform a command. You can optionally specify an ELSE clause on the IF command. If the test expression evaluates to false and an ELSE clause exists, the command associated with the ELSE clause is performed. The keywords cannot be abbreviated.

```
IF expression THEN command [ELSE command];
```

*expression*

A valid Debug Tool PL/I expression.

If necessary, the expression is converted to a BIT string.

*command*

A valid Debug Tool command.

When IF commands are nested and ELSE clauses are present, a given ELSE is associated with the closest preceding IF clause within the same block.
Usage notes

- An ELSE clause should always be included if the IF clause causes Debug Tool to get more input (for example, an IF containing USE or other commands that cause Debug Tool to be restarted because an AT-condition occurs).
- The IF command cannot be used while you replay recorded statements by using the LAYBACK commands.

Examples

- If the value of array1 is equal to the value of array2, go to the statement with label constant label_1. Execution of the user program continues at label_1. If array1 does not equal array2, the GOTO is not performed and control is passed to the user program.
  
  ```
  IF array1 = array2 THEN GOTO LABEL label_1; ELSE GO;
  ```
- Set a breakpoint at statement 23, which will test if variable j is equal to 10, display the names and values of variables rmdr, totodd, and terms(j). If variable j is not equal to 10, continue program execution.
  
  ```
  AT 23 IF j = 10 THEN LIST TITLED (rmdr, totodd, terms(j)); ELSE GO;
  ```

**IMMEDIATE command (full-screen mode)**

The IMMEDIATE command causes a command within a command list to be performed immediately. It is intended for use with commands assigned to a PF key.

IMMEDIATE can only be entered as an unnested command or within a compound command.

It is recommended that PF key definitions for FIND, RETRIEVE, SCROLL, and WINDOW commands be prefixed with IMMEDIATE. This makes it possible to do things like SCROLL even when entering a group of commands.

```
>>>IMMEDIATE—command—;
```

**command**

One of the following Debug Tool commands:

- FIND
- RETRIEVE
- SCROLL commands
  - BOTTOM
  - DOWN
  - LEFT
  - NEXT
  - RIGHT
  - TO
  - TOP
  - UP
- WINDOW commands
  - CLOSE
  - OPEN
  - SIZE
  - ZOOM

Usage notes

- The IMMEDIATE command is not logged.
Examples

- Specify that the WINDOW OPEN LOG command be immediately effective.
  
  IMMEDIATE WINDOW OPEN LOG;

- Specify that the SCROLL BOTTOM command be immediately effective.
 _IMMEDIATE SCROLL BOTTOM;

**INPUT command (C, C++, and COBOL)**

The INPUT command provides input for an intercepted read and is valid only when there is a read pending for an intercepted file. The INPUT keyword cannot be abbreviated.

```
>>—INPUT—text—;
```

- **text**

  Specifies text input to a pending read.

Usage notes

- The INPUT text consists of everything between the INPUT keyword and the semicolon (or end-of-line). Any leading or trailing blanks are removed by Debug Tool.
- If a semicolon is included as part of the INPUT text, or if the first character of the INPUT text is a quote, the INPUT text must conform to the current programming language syntax for a character string constant (that is, enclosed in quotes, with internal quotes entered according to the rules of that programming language).
- This command is not supported for CICS.
- To set interception to and from a file, use the SET INTERCEPT (C, C++, and COBOL) command.
- The INPUT command cannot be used while you replay recorded statements by using the PLAYBACK commands.

Example

You have used SET INTERCEPT ON to make Debug Tool prompt you for input to a sequential file. The prompt and the file’s name appears in the Command Log.

To substitute the input that would have come from the DD name specified by the SET INTERCEPT ON command with your desired input, enter:

```
INPUT text you want to input;
```

Program input is recorded in your Log window.

A closing semicolon (;) is required for this command. Everything between the INPUT keyword and the semicolon is considered input text. If you want to include a semicolon in your input, or if the first character of your input is a quote, you must enter your input as a valid character string for your programming language.

Indicate that the phrase "quick brown fox" is input to a pending read. The phrase is written to the file.

```
INPUT quick brown fox;
```

Refer to the following sections for more information related to the material discussed in this section.
JUMPTO command

The JUMPTO command moves the point at which the program resumes running to the specified statement but does not resume running the program.

```
JUMPTO statement_id;
```

Notes:
1. Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

Usage notes
- You cannot use the JUMPTO command while you debug a disassembled program.
- If you specify the JUMPTO command in a command list (for example, as the subject of an IF command or WHEN clause), all subsequent commands in the list are ignored.
- If the program is compiled with minimum optimization, the JUMPTO command is not restricted to specific statements.
- You cannot use the JUMPTO command while you replay recorded statements by using the PLAYBACK command.
- For C, C++, and PL/I programs, statements can be removed by the compiler during optimization. Specify a reference or statement for the JUMPTO command that can be reached while the program is running. You can use the LIST STATEMENT NUMBERS command to determine the statements that can be reached.
- For PL/I programs, you can use JUMPTO in a command list on a call to PLITEST or CEETEST.
- For PL/I programs, you cannot specify a statement that is out of the currently active block. However, you might have to qualify the statement.
- For COBOL programs, the JUMPT0 command follows the COBOL language rules that apply to the GOTO statement. You can use the JUMPT0 command only if you compiled your program with either the PATH or ALL suboption and the SYM suboption of the TEST compiler option.
- If you are debugging an optimized COBOL program, you cannot use the JUMPT0 command.
- You can use the JUMPT0 command to debug COBOL programs compiled with the NONE suboption of the TEST compiler option and the following compilers:
  - Enterprise COBOL for z/OS and OS/390, Version 3 Release 2
  - Enterprise COBOL for z/OS and OS/390, Version 3 Release 1, with APAR PQ63235 installed
  - COBOL for OS/390 & VM, Version 2, with APAR PQ63234 installed

Example

You want to jump to statement 24 and then stop there. Enter the following command:

```
JUMPTO 24;
```
Refer to the following sections for more information related to the material discussed in this section.

**Related tasks**
- Debug Tool User’s Guide

**Related references**
- "statement_id" on page 16

---

**JUMPTO LABEL command**

The **JUMPTO LABEL** command moves the point at which the program resumes running to the specified label but does not resume running the program.

```
(1) JUMPTO 'statement_label'
```

**Notes:**

1. Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

**statement_label**

A valid statement label within the currently executing program or, in PL/I, a label variable.

**Usage notes**

- Use the syntax of **statement_label** enclosed in single-quotes only for OS/VS COBOL programs. It is not supported in any other programming language.
- In PL/I, out-of-block JUMPTOs are allowed. However, qualification might be needed.
- The **LABEL** keyword is optional when either the target **statement_label** is nonnumeric or if it is qualified (whether the actual label was nonnumeric or not).
- A COBOL **statement_label** can have either of the following forms:
  - **name**
    - This form can be used in COBOL for reference to a section name or for a COBOL paragraph name that is not within a section or is in only one section of the block.
  - **name1 OF name2** or **name1 IN name2**
    - This form must be used for any reference to a COBOL paragraph (name1) that is within a section (name2), if the same name also exists in other sections in the same block. You can specify either **OF** or **IN**, but Debug Tool always uses **OF** for output.
  
    Either form can be prefixed with the usual block, compile unit, and load module qualifiers.

- For C, to be able to use the **JUMPTO LABEL** command, you must compile your program in one of the following ways:
  - With either the **PATH** or **ALL** suboption and the **SYM** suboption of the **TEST** compiler option. The default is **PATH**.
  - With either the **PATH** or **ALL** suboption and the **SYM** suboption of the **DEBUG** compiler option.

There are no restrictions on using labels with the **JUMPTO LABEL** command.
For C++, to be able to use the JUMPTO LABEL command, you must compile your program in one of the following ways:
- With the TEST compiler option.
- With either the PATH or ALL suboption and the SYM suboption of the DEBUG compiler option.

There are no restrictions on using labels with the JUMPTO LABEL command.

For COBOL, you can use JUMPTO LABEL only if you compiled your program with either PATH or ALL suboption and the SYM suboption of the TEST compiler option. The label itself can take either of two forms:
- name, where name is a section name, or the name of a paragraph not within a section or in only one section of the block.
- name1 OF name2 or name1 IN name2, where name1 is duplicated by one or more other paragraphs in one or more other sections in the block. You can use either OF or IN, but Debug Tool always logs OF.

For PL/I, you can use JUMPTO LABEL only if you compiled your program with either the PATH or ALL suboption and the SYM suboption of the TEST compiler option. There are no restrictions on using labels with JUMPTO LABEL and label variables are supported.

JUMPTO LABEL is not available while debugging Enterprise PL/I programs.

You cannot use the JUMPTO LABEL command while you are replaying recorded steps by using the PLAYBACK commands.

You cannot use the JUMPTO LABEL command while you debug an optimized COBOL program.

**Examples**

- Jump to the label constant laba in block suba in program prog1.
  
  JUMPTO prog1:suba:lab;  

- Jump to the label constant para OF sect1. The current programming language setting is COBOL.
  
  JUMPTO LABEL para OF sect;  

Refer to the following sections for more information related to the material discussed in this section.

**Related tasks**

- [Debug Tool User's Guide](#)

**Related references**

- "statement_label" on page 17

---

**LIST command**

The LIST command displays information about a program such as values of specified variables, structures, arrays, registers, statement numbers, frequency information, and the flow of program execution. The LIST command can be used to display information in any enclave. All information displayed will be saved in the log file.

The following table summarizes the forms of the LIST command.

<table>
<thead>
<tr>
<th>LIST (blank)&quot; on page 125</th>
<th>Displays Source Identification panel</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;LIST AT&quot; on page 125</td>
<td>Lists the currently defined breakpoints.</td>
</tr>
<tr>
<td>&quot;LIST CALLS&quot; on page 128</td>
<td>Displays the dynamic chain of active blocks.</td>
</tr>
</tbody>
</table>
**LIST CURSOR (full-screen mode)** on page 128
Displays the variable pointed to by the cursor.

**LIST expression** on page 129
Displays values of expressions.

**LIST FREQUENCY** on page 132
Lists statement execution counts.

**LIST LAST** on page 132
Displays a list of recent entries in the history table.

**LIST LINE NUMBERS** on page 133
Lists all line numbers that are valid locations for an AT LINE breakpoint.

**LIST LINES** on page 133
Lists one or more lines from the current listing or source file displayed in the Source window.

**LIST MONITOR** on page 134
Lists the current set of MONITOR commands.

**LIST NAMES** on page 134
Lists the names of variables, programs, or Debug Tool procedures.

**LIST ON (PL/I)** on page 136
Lists the action (if any) currently defined for the specified PL/I conditions.

**LIST PROCEDURES** on page 136
Lists the commands contained in the specified Debug Tool procedure.

**LIST REGISTERS** on page 137
Displays the current register contents.

**LIST STATEMENT NUMBERS** on page 137
Lists all statement numbers that are valid locations for an AT STATEMENT breakpoint.

**LIST STATEMENTS** on page 138
Lists one or more statements from the current listing or source file displayed in the Source window.

**LIST STORAGE** on page 139
Provides a dump-format display of storage.

---

**LIST (blank)**
Displays the Source Identification panel, where associations are made between source listings or source files shown in the source window and their program units. LIST is equivalent to PANEL LISTINGS and PANEL SOURCES.

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

"PANEL command (full-screen mode)" on page 151

**LIST AT**
Lists the currently defined breakpoints, including the action taken when the specified breakpoint is activated. If no action is defined, Debug Tool displays the NULL command.
**AT_command**
A valid AT command that includes at least one operand. The AT command must be complete except that the every_clause and command are omitted.

**ENABLED**
Restricts the list to enabled breakpoints. The default is to list both enabled and disabled breakpoints.

**DISABLED**
Restricts the list to disabled breakpoints. The default is to list both enabled and disabled breakpoints.

**ALLOCATE**
Lists currently defined AT ALLOCATE breakpoints.

**APPEARANCE**
Lists currently defined AT APPEARANCE breakpoints.

**CALL**
Lists currently defined AT CALL breakpoints.

**CHANGE**
Lists currently defined AT CHANGE breakpoints. This displays the storage address and length for all AT CHANGE subjects, and shows how they were specified (if other than by the %STORAGE function).

**DATE**
Lists currently defined AT DATE breakpoints.
DELETE
Lists currently defined AT DELETE breakpoints.

ENTRY
Lists currently defined AT ENTRY breakpoints.

EXIT
Lists currently defined AT EXIT breakpoints.

GLOBAL
Lists currently defined AT GLOBAL breakpoints for the specified AT-condition.

LABEL
Lists currently defined AT LABEL breakpoints.

LINE
Lists currently defined AT LINE or AT STATEMENT breakpoints. LINE is equivalent to STATEMENT.

LOAD
Lists currently defined AT LOAD breakpoints.

OCCURRENCE
Lists currently defined AT OCCURRENCE breakpoints.

OFFSET
Lists currently defined AT OFFSET breakpoints.

PATH
Lists currently defined AT PATH breakpoints.

STATEMENT
Is equivalent to LINE.

SUSPENDED
Lists all suspended breakpoints.

TERMINATION
Lists currently defined AT TERMINATION breakpoint.

If the AT command type (for example, LOAD) is not specified, LIST AT lists all currently defined breakpoints (both disabled and enabled).

Usage notes
• To display a global breakpoint, you can specify an asterisk (*) with the LIST AT command or you can specify a LIST AT GLOBAL command. For example, if you want to display an AT ENTRY * breakpoint, specify:
  LIST AT ENTRY *;
  or
  LIST AT GLOBAL ENTRY;
  If you have only a global breakpoint set and you specify LIST AT ENTRY without the asterisk (*) or GLOBAL keyword, you get a message saying there are no such breakpoints.
• The LIST AT command cannot be used while you replay recorded statements by using the PLAYBACK commands.

Examples
• Display information about enabled breakpoints defined at block entries.
  LIST AT ENABLED ENTRY;
• Display information about global DATE breakpoint entries.
  LIST AT DATE *;
- Display breakpoint information for all disabled AT CHANGE breakpoints within the currently executing program.
  LIST AT DISABLED CHANGE;
- The current programming language setting is C. Here are some assorted LIST AT commands.
  LIST AT LINE 22;
  or
  LIST AT OCCURRENCE SIGSEGV;
  or
  LIST AT CHANGE structure.un.m;

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

[“AT command” on page 37]

**LIST CALLS**

Displays the dynamic chain of active blocks. For languages without block structure, this is the CALL chain. Under z/OS batch and TSO, LIST CALLS lists the call chain of every active enclave in the process.

```
LIST CALLS;
```

**Usage notes**

- For programs containing interlanguage communication (ILC), routines from previous enclaves are only listed if they are written in a language that is active in the current enclave.
- If the enclave was created with the `system()` function, compile units in parent enclaves are not listed.
- If you are debugging a program that does not follow the standard linkage conventions for R13, R14, and R15, the output of the LIST CALLS command can be incorrect or incomplete.
- If you are debugging a disassembled program and you encounter one of the following situations:
  - The registers' save area has not been created.
  - The registers are not chained to the other save areas.

Some of the programs or CSECTs in the call chain are not displayed.
- The LIST CALLS command cannot be used while you replay recorded statements by using the PLAYBACK commands.

**Example**

Display the current dynamic chain of active blocks.

```
LIST CALLS;
```

**LIST CURSOR (full-screen mode)**

Provides a cursor controlled method for displaying variables, structures, and arrays. It is most useful when assigned to a PF key.
Usage notes

- Cursor pointing can be used by typing the LIST CURSOR command on the command line and moving the cursor to a variable in the source window before pressing Enter, or by moving the cursor and pressing a PF key with the LIST CURSOR command assigned to it.
- When you use the LIST CURSOR command for a variable that is located by the cursor position, the variable’s name nor its full qualification cannot be split across different lines of the source listing.
- If the DATA option of the PLAYBACK ENABLE command is in effect for the current compile unit, the LIST CURSOR command can be used while you replay recorded statements by using the PLAYBACK commands.
- For optimized COBOL programs, you cannot use the LIST CURSOR command to display the value of variables discarded by the optimizer.

Examples

- Display the value of the variable at the current cursor position.
  
  ```plaintext
  LIST CURSOR
  ```

- A COBOL program has a statement of the form:
  
  ```plaintext
  MOVE a TO b OF c
  ```

  You cannot use the LIST CURSOR on the variable b because part of its qualification (OF c) is on the next line.

LIST expression

Displays values of expressions.

Notes:

1. Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

TITLED

Displays each expression in the list with its value. For PL/I, this is the default. For C and C++, this is the default for expressions that are `values`. For COBOL, this is the default except for expressions consisting of only a single constant.
For assembler, disassembly, and OS/VS COBOL, this is the default for expressions that are valid as receivers of a Debug Tool assembler assignment statement.

If you specify TITLED with no keyword, all variables in the currently qualified block are listed. If you specify TITLED with an asterisk (*) and you are debugging a C, C++, or COBOL program, all variables in the currently qualified compile unit are listed.

If you are debugging a COBOL program, the following additional options are available with TITLED:

- **FS** Lists all variables defined in the COBOL File Section in the currently qualified compile unit.

- **WSS** Lists all variables defined in the COBOL Working-Storage Section in the currently qualified compile unit.

- **LS** Lists all variables defined in the COBOL Linkage Section in the currently qualified compile unit.

- **LOS** List all variables defined in the COBOL Local-Storage Section in the currently qualified compile unit.

* (C, C++, and COBOL)
Lists all variables in the currently qualified compile unit.

**UNTITLED**
Lists expression values without displaying the expressions themselves. For C and C++, this is the default for expressions that are not valid. For COBOL, this is the default for expressions consisting of only a single constant. For assembler, disassembly, and OS/VS COBOL, this is the default for expressions that are not valid as receivers of a Debug Tool assembler assignment statement.

For the LIST command, an expression also includes character strings enclosed in either double (") or single (') quotes, depending on the current programming language.

In C and COBOL, expressions containing parentheses () must be enclosed in another set of parentheses when used with the LIST command as in example LIST ((x + y) / z);

In COBOL, an expression can be the GROUP keyword followed by a reference. If specified, the GROUP keyword causes the reference to be displayed as if it were an elementary item. If GROUP is specified for an elementary item, it has no effect. The operand of a GROUP keyword can only be a reference (expressions are not allowed) as in example LIST TITLED GROUP y;

**expression**
An expression valid in the current programming language other than OS/VS COBOL.

'**expression**'
A valid OS/VS COBOL expression enclosed in single quotes.

**Usage notes**
- If you want to use the LIST TITLED with the parameters FS, WSS, LS or LOS, the PTF for Language Environment APAR PK12834 must be installed on z/OS Version 1 Release 4, Version 1 Release 5, Version 1 Release 6 and Version 1 Release 7.
• For COBOL programs, if you want to use the LIST TITLED command with a variable that is named FS, WSS, LS, or LOS, you must enclose the name of the variable in parenthesis. For example, the command LIST TITLED (FS) lists the variable FS; the command LIST TITLED FS lists the variables in the File Section.

• Debug Tool allows you to abbreviate many commands. This might result in unexpected results when you use the LIST command with a single-letter expression. For example, LIST A can be interpreted as the LIST AT command, which lists all breakpoints. However, if you wanted to display the value of a variable labeled A in your program, you need to use parenthesis: LIST (A).

• If LIST TITLED * is specified and your compile unit is large, slow performance might result.

• For COBOL, if LIST TITLED * is specified and your compile unit is large, you might receive an out of storage error message.

• For COBOL, the LIST command can reference a condition name, a file name, or an expression.

• For optimized COBOL programs, the LIST command cannot reference a variable that was discarded by the optimizer.

• When using LIST TITLED with no parameters within the PL/I compile unit, only the first element of any array will be listed. If the entire array needs to be listed, use LIST and specify the array name (i.e., LIST array where array is the name of an array).

• Currently, Debug Tool only supports two character sets: English and Japanese. If a variable contains unprintable characters, an error message is displayed.

• If the DATA option of the PLAYBACK ENABLE command is in effect for the current compile unit, the LIST expression command can be used while you replay recorded statements by using the PLAYBACK commands.

• If you are trying to display a scalar item, the maximum length that LIST can display is 65,535 bytes.

• If a character variable contains data that cannot be displayed, Debug Tool displays those characters in hexadecimal, which requires twice as many bytes. The maximum number of bytes that can be displayed is 65,535.

• You can use the LIST prefix command to list a member of the current set of MONITOR commands.

Examples

• Display the values for variables size and r and the expression c + r, with their respective names.

  LIST TITLED (size, r, c + r);

• Display the COBOL references as if they were elementary items. The current programming language setting is COBOL.

  LIST (GROUP x OF z(1,2), GROUP a, w);

• Display the value of the Debug Tool variable %ADDRESS.

  LIST %ADDRESS;

• In the disassembly view, display the value of register 1 (R1), which is the value of Debug Tool variable %R1.

  LIST R1 ;

• In COBOL, display the names and values of variables defined in the File Section.

  LIST TITLED FS;

Refer to the following sections for more information related to the material discussed in this section.
Related references
“expression” on page 14
“SET LIST TABULAR” on page 193

**LIST FREQUENCY**
Lists statement execution counts.

```plaintext
LIST FREQUENCY statement_id_range;
```

* Lists frequency for all statements in the currently qualified compile unit. If currently executing at the AT TERMINATION breakpoint where there is no qualification available, it will list frequency for all statements in all the compile units in the terminating enclave where frequency data exists.

**LINES**
Displays the source line after the frequency count.

**STATEMENT**
Equivalent to LINES.

**Usage notes**
- In the disassembly view, LIST FREQUENCY and LIST FREQUENCY * are not supported.
- When you replay recorded statements by using the PLAYBACK commands, the frequency count is not updated.

**Examples**
- List frequency for statements 1-20.
  ```plaintext
  LIST FREQUENCY 1 - 20;
  ```
- List frequency and statement for statements 18 - 19:
  ```plaintext
  LIST FREQUENCY LINES 18-19;
  ```
- List frequency for all statements in the currently qualified compile unit.
  ```plaintext
  LIST FREQUENCY *;
  ```
- List frequency for all statements in all compile units.
  ```plaintext
  AT TERMINATION LIST FREQUENCY *;
  ```

Refer to the following sections for more information related to the material discussed in this section.

**Related references**
“statement_id_range and stmt_id_spec” on page 16
“SET FREQUENCY” on page 190

**LIST LAST**
Displays a list of recent entries in the history table.

```plaintext
LIST LAST integer HISTORY LINES PATHS STATEMENTS;
```
integer
   Specifies the number of most recently processed breakpoints and conditions displayed.

HISTORY
   Displays all processed breakpoints and conditions.

LINES
   Displays processed statement or line breakpoints. LINES is equivalent to STATEMENTS.

PATHS
   Displays processed path breakpoints.

STATEMENTS
   Is equivalent to LINES.

Usage notes
   • The LAST keyword is provided to make the LIST command readable. It does not perform any function.
   • In the disassembly view, LIST LAST is not supported.

Examples
   • Display all processed path breakpoints in the history table.
     LIST PATHS;
   • Display all program breakpoints and conditions for the last five times Debug Tool gained control.
     LIST LAST 5 HISTORY;

Refer to the following sections for more information related to the material discussed in this section.
   Related references
   "SET HISTORY" on page 191

LIST LINE NUMBERS

Equivalent to LIST STATEMENT NUMBERS.

Refer to the following sections for more information related to the material discussed in this section.
   Related references
   "LIST STATEMENT NUMBERS" on page 137

LIST LINES

Equivalent to LIST STATEMENTS.

Refer to the following sections for more information related to the material discussed in this section.
   Related references
   "LIST STATEMENTS" on page 138
**LIST MONITOR**

Lists all or selected members of the current set of MONITOR commands.

```
LIST MONITOR [integer - integer];
```

*integer*

An unsigned integer identifying a MONITOR command. If two integers are specified, the first must not be greater than or equal to the second. If omitted, all MONITOR commands are displayed.

**Usage notes**

- You can enter LIST in the prefix area of the monitor window to list the monitor command of the selected line.
- When the current programming language setting is COBOL, blanks are required around the hyphen (-). Blanks are optional for C.

**Example**

List the fifth through the seventh commands currently being monitored.

```
LIST MONITOR 5 - 7;
```

**LIST NAMES**

Lists the names of variables, programs, or Debug Tool procedures. If LIST NAMES is issued with no keyword specified, the names of all program and session variables that can be referenced in the current programming language and that are visible to the currently qualified block are displayed. A subset of the names can be specified by supplying a pattern to be matched.

```
LIST NAMES [pattern] [BLOCK block_spec, cu_spec];
```

*pattern*

The pattern searched for, conforming to the current programming language syntax for a character string constant. The pattern length cannot exceed 128 bytes, excluding the quotes.

If the DBCS setting is ON, the pattern can contain DBCS characters. DBCS shift codes are not considered significant characters in the pattern. Within the pattern, an SBCS or DBCS asterisk represents a string of zero or more insignificant SBCS or DBCS characters. As many as eight asterisks can be included in the pattern, but adjacent asterisks are equivalent to a single asterisk.
Some examples of possible strings follow:

<table>
<thead>
<tr>
<th>C</th>
<th>Assembler</th>
<th>PL/I</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>COBOL</td>
<td>COBOL</td>
</tr>
<tr>
<td>OS/VS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>&quot;ABC&quot;</th>
<th>&quot;A5&quot;</th>
<th>'MY'</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>'A5'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pattern matching is not case-sensitive outside of DBCS. Both the pattern and potential names outside of shift codes are effectively uppercased, except when the current programming language setting is C. Letters in the pattern must be the correct case when the current programming language setting is C.

**BLOCK**
Displays variable names that are defined within one or more specified blocks.

**CUS**
Displays the compile unit names. CUS is equivalent to PROGRAMS.

**PROCEDURES**
Displays the Debug Tool procedure names.

**PROGRAMS**
Is equivalent to CUS.

**TEST**
Displays the Debug Tool session variable names.

**Usage notes**
- LIST NAMES CUS applies to compile unit names.
- LIST NAMES TEST shows only those session variable names that can be referenced in the current programming language.
- The output of LIST NAMES without any options depends on both the current qualification and the current programming language setting. If the current programming language differs from the programming language of the current qualification, the output of the command shows only those session variable names that can be referenced in the current programming language.
- For structures, the pattern is tested against the complete name, hence "B" is not satisfied by "C OF B OF A:" (COBOL).
- If the DATA option of the PLAYBACK ENABLE command is in effect for the current compile unit, you can use the LIST NAMES command while you replay recorded statements by using the PLAYBACK commands.
- For optimized COBOL programs, the LIST NAMES command does not display variables discarded by the optimizer.

**Examples**
- Display all compile unit names that begin with the letters "MY" and end with "5". The current programming language setting is either C or COBOL.
  LIST NAMES "MY*5" PROGRAMS;
- Display the names of all the Debug Tool procedures that can be called.
  LIST NAMES PROCEDURES;
- Display the names of variables whose names begin with 'R' and are in the mainprog block. The current programming language setting is COBOL.
  LIST NAMES 'R*' BLOCK (mainprog);
Refer to the following sections for more information related to the material discussed in this section.

**Related references**
- "block_spec" on page 11
- "cu_spec" on page 13

**LIST ON (PL/I)**
Lists the action (if any) currently defined for the specified PL/I conditions.

```
LIST—ON pli_condition;
```

**pli_condition**
A valid PL/I condition specification. If omitted, all currently defined ON command actions are listed.

**Usage notes**
- You cannot use the LIST ON command while you replay recorded statements by using the PLAYBACK commands.

**Example**

List the action for the ON ZERODIVIDE command.

```
LIST ON ZERODIVIDE;
```

Refer to the following sections for more information related to the material discussed in this section.

**LIST PROCEDURES**
Lists the commands contained in the specified Debug Tool PROCEDURE definitions.

```
LIST—PROCEDURES name [(name)];
```

**name**
A valid Debug Tool procedure name. If no procedure name is specified, the commands contained in the currently running procedure are displayed. If no procedure is currently running, an error message is issued.

**Usage note**

**Examples**
- Display the commands in the Debug Tool procedure p2.
  
  ```
  LIST PROC p2;
  ```
- List the procedures abc and proc7.
  
  ```
  LIST PROCEDURES (abc, proc7);
  ```
LIST REGISTERS
Displays the current register contents.

REGISTERS
Displays the general-purpose registers (%GPRn). When this command is issued
when you are qualified to an Assembler or Disassembly CU other than the CU
where execution was suspended, it also displays the values of the %Rn
symbols.

LONG
Displays the decimal value of the long-precision floating-point registers.

SHORT
Displays the decimal value of the short-precision floating-point registers.

FLOATING
Displays the long-precision floating-point registers.

Examples
• Display the general-purpose registers at the point of a program interruption:
  LIST REGISTERS;
• Display the floating-point registers.
  LIST FLOATING REGISTERS;

LIST STATEMENT NUMBERS
Lists all statement or line numbers that are valid locations for an AT LINE or AT
STATEMENT breakpoint.

NUMBERS
Displays the statement numbers that can be used to set STATEMENT breakpoints,
assuming the compile options used to generate statement hooks were specified
at compile time. The list can also be used for the GOTO command, however, you
might not be able to GOTO all of the statement numbers listed.

block_spec
A valid block specification. This operand lists all statement or line numbers in
the specified block.

cu_spec
A valid compile unit specification. For C programs, cu_spec can be used to list
the statement numbers that are defined within the specified compile unit
before the first function definition.

statement_id_range
A valid range of statement ids, separated by a hyphen (-).
Usage notes
- In the disassembly view, LIST STATEMENT NUMBERS is not supported.

Examples
- List the statement or line numbers in the currently qualified block.
  LIST STATEMENT NUMBERS;
- Display the statement or line number of every statement in block earnings.
  LIST STATEMENT NUMBERS earnings;

Refer to the following sections for more information related to the material discussed in this section.

Related references
- "block_spec" on page 11
- "cu_spec" on page 13
- "statement_id_range and stmt_id_spec" on page 16

LIST STATEMENTS
Lists one or more statements or lines from a file. It is primarily intended for viewing portions of the source listing or source file in line mode, but can also be used in full-screen mode to copy a portion of a source listing or source file to the log.

Listing syntax:

```plaintext
>>> LIST LINE statement_id_range;
```

Usage notes
- The specified lines are displayed in the same format as they would appear in the full-screen Source window, except that wide lines are truncated.
- You might need to specify a range of line numbers to ensure that continued statements are completely displayed.
- This command is not to be confused with the LIST LAST STATEMENTS command.
- In the disassembly view, LIST STATEMENTS is not supported.

Examples
- List lines 25 through 30 in the source file associated with the currently qualified compile unit.
  LIST LINES 25 - 30;
- List statement 100 from the current program listing file.
  LIST STATEMENT 100;

Refer to the following sections for more information related to the material discussed in this section.

Related references
- "statement_id_range and stmt_id_spec" on page 16
LIST STORAGE

Displays the contents of storage at a particular address in hex format.

\[\text{LIST STORAGE} - \text{(address)} - \text{(reference)} - \text{('reference')} - \text{('offset')} - \text{('length')};\]

address
The starting address of storage to be watched for changes. This must be a hex constant in one of the following formats:
- \(0x\) in C
- \(H\) in COBOL (using either double ‘(‘) or single (‘) quotes)
- \(PX\) in PL/I
- \(X\) with single quotes (‘) or double quotes (‘) surrounding the hex constant in assembler or disassembly
- \(X\) with double quotes (‘) surrounding the hex constant in OS/VS COBOL

reference
A variable whose storage location is to be watched for changes.

In assembler or disassembly, this operand might be specified as any assembler expression that represents a storage location. If the assembler expression does not have an implied length (for example, R3->+10), you must specify the number of bytes to display by using the integer operand.

‘reference’
An OS/VS COBOL variable whose storage location is to be watched for changes. An OS/VS COBOL reference must be enclosed in single quotes.

offset
The decimal or hexadecimal number of bytes indicating the starting offset from the memory location pointed to by the reference’s address or the address provided by the user. offset can be a negative number. If offset is a hex constant it must follow the language rules for a hex constant, as described for the address above. The default is 0.

length
The decimal number of bytes of storage displayed. The default is 16 bytes. The length must be an integer number.

Usage notes
- For C and C++, if the referenced variable is an array, Debug Tool displays the storage at the address of that array. However, if the referenced variable is a pointer, Debug Tool displays the storage at the address given by that pointer.
- Using Debug Tool, cursor pointing can be used by typing the LIST STORAGE command on the command line and moving the cursor to a variable in the Source window before pressing Enter, or by moving the cursor and pressing a PF key with the LIST STORAGE command assigned to it.
- When using the LIST STORAGE command in Debug Tool for a variable that is located by the cursor position, the variable’s name cannot be split across different lines of the source listing.
- If the referenced variable is a General Purpose Register (GPR) such as %GPR1, the result depends on the programming language that is in effect:
  - For all languages except assembler and disassembly, Debug Tool displays the storage at the address contained in the referenced GPR.
- For assembler and disassembly, you must use the indirection notation (%GPR1->) to instruct Debug Tool to display the storage at the address contained in the referenced register.

- If no operand is specified with LIST STORAGE, the command is cursor-sensitive.
- If you are replaying recorded statements by using PLAYBACK commands, the LIST STORAGE command displays the contents of storage at the point where you entered the PLAYBACK START command.
- For optimized COBOL programs, LIST STORAGE cannot display variables that were discarded by the optimizer.

Examples
- Display the first 64 bytes of storage beginning at the address of variable table.
  `LIST STORAGE (table, 64);`
- Display 16 bytes of storage at the address given by pointer table(1).
  `LIST STORAGE (table(1));`
- Display the 16 bytes contained at locations 20CD0-20CDF. The current programming language setting is COBOL.
  `LIST STORAGE ('H'20CD0');`
- Display the 16 bytes contained at locations 20CD0-20CDF. The current programming language setting is PL/I.
  `LIST STORAGE ('20CD0'PX);`
- In the disassembly view, display the storage at the address given by register R13.
  `LIST STORAGE (R13->);`
- Display 10 characters starting at offset 2 for variable MYVAR. MYVAR is declared as CHAR (20).
  `LIST STORAGE (MYVAR, 2, 10);`
- Display 20 bytes starting at offset 10 from address '20ACD0'PX. The current programming language setting is PL/I.
  `LIST STORAGE ('20ACD0'PX, 10, 20);`
- Display 10 bytes starting at offset -5 from address '20ACD0'PX. The current programming language setting is PL/I.
  `LIST STORAGE ('20ACD0'PX, -5, 10);`

Refer to the following sections for more information related to the material discussed in this section.

Related references
"references" on page 15

LOAD command

Specifies that the named module should be loaded for debugging purposes. The LOAD command enables you to debug preloaded load modules.

If you are running in Language Environment, the enclave-level load service is used to load the load module or modules. The load module or modules remain active until the current enclave terminates or you enter the CLEAR LOAD command for those load modules.

If you are not running in Language Environment, the load module or modules remain active until the debugging task terminates or you enter the CLEAR LOAD
command for those load modules. If you are debugging CICS programs, the load is done by EXEC CICS LOAD. For all other programs, the load is done by MVS LOAD services.

![Diagram](LNML0010.png)

Notes:
1 Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

module_name
   The name of one or more load modules to be loaded by Debug Tool.

LE Use the Language Environment enclave-level load service to load the load module or modules. The load module or modules remain active until the current enclave terminates or you enter a CLEAR LOAD command for the load module or modules.

NONLE
   Use non-Language Environment services to load the load module or modules. The load module or modules remain active until the debugging task terminates or you enter a CLEAR LOAD command for the load module or modules. For CICS programs, the load module or modules are loaded by using EXEC CICS LOAD. For all other programs, the load module or modules are loaded by using the MVS LOAD services.

Usage notes
   • You can use this command in remote debug mode when you use one of the following remote debuggers:
     – Compiled Language Debugger component of WebSphere Studio Enterprise Developer
     – Compiled Language Debugger component of WebSphere Developer for zSeries
     – WebSphere Developer Debugger for zSeries
   • You can enter the QUALIFY CU command for a program or CSECT in the load module or load modules that you just loaded unless the program is COBOL.
   • If you set breakpoints in the programs or CSECTS in the module and then the same load module is loaded again, the breakpoints might not work because location of the load module has changed.
   • If the module to be debugged is RESIDENT or was loaded before Debug Tool was started, you can use the LOAD command to make the module known to Language Environment.

LOADDEBUGDATA
   Specifies that a compile unit (CU) is an assembler or OS/VS COBOL CU and loads debug data from the default data set name, userid.EQALANGX(cu_name). If the debug data is stored in a different data set, you can specify that data set name by using the SET SOURCE, SET DEFAULT LISTINGS command, or the EQADEBUG DD statement. In remote debug mode, you can specify the data set name by using the EQADEBUG DD statement or let the remote debugger prompt you for the data set name.
You can generate the required debug data by using the EQALANGX program or, if you are debugging an assembler program, by assembling your program through Debug Tool Utilities. Both methods are described in "Debug Tool User’s Guide".

You can use the `LOADDEBUGDATA` command to load debug information into a compile unit. If the compile unit is currently known to Debug Tool, the command is run immediately. If it is not currently known to Debug Tool, the command is deferred until the compile unit is known and the session is active. You can use the `LOADDEBUGDATA` command from a terminal or any z/OS terminal session. The command is deferred when the session is active and the compile unit is not currently known to Debug Tool.

```
LOADDEBUGDATA (1) {'cu_name'}
LDD (2)
(3) {'cu_name'}
```

Notes:

1. Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

`cu_name`

The name of the assembler or OS/VS COBOL compile unit. If this compile unit is currently known to Debug Tool, it must be a disassembly compile unit. If it is not currently known to Debug Tool, the `LOADDEBUGDATA` command is deferred until a disassembly compile unit by the specified name becomes known to Debug Tool.

Usage notes

- When you use the `SET SAVE` command to save breakpoints or monitor specifications or you use the `RESTORE` command to restore breakpoints or monitor specifications, all LDD settings including the data set name of the data set from which the debug data was loaded is saved and restored.
- **For CICS only**: When a DTCN profile is active for a full screen mode debugging session, Debug Tool preserves all LDD settings, including the data set name of the data set from which the debug data was loaded, until the DTCN profile is deleted or the terminal session is terminated.
- You can use this command for assembler CUs (but not OS/VS COBOL CUs) in remote debug mode when you use one of the following remote debuggers:
  - Compiled Language Debugger component of WebSphere Studio Enterprise Developer
  - Compiled Language Debugger component of WebSphere Developer for zSeries
  - WebSphere Developer Debugger for zSeries
- After Debug Tool successfully processes a `LOADDEBUGDATA` command for a CU, if the CU is deleted and then appears later, an implicit `LDD` command is run for the CU using the same EQALANGX data set that was used initially.
- You cannot enter the `LDD` command for the same compile unit more than once.

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

"SET LDD" on page 193

**MONITOR command**

The MONITOR command defines or redefines a command and then displays the output in the monitor window (full-screen mode) or log file (batch mode). The following commands are the only commands you can use with the MONITOR command:

- `DESCRIBE`
- `LIST`
Null
QUERY

Debug Tool maintains a list of your most recently entered MONITOR commands. Each command entered is assigned a number between 1 and 99 or you can assign it a number. Use these numbers to indicate to Debug Tool which MONITOR command you want to redefine.

GLOBAL
Specifies that the monitor definition is global. That is, it is not associated with a particular compile unit.

LOCAL
Specifies that the monitor definition is local to a specific compile unit. Using Debug Tool, the specified output is displayed only when the current qualification is within the associated compile unit.

cu_spec
A valid compile unit specification. This specifies the compile unit associated with the monitor definition.

integer
An integer in the range 1 to 99, indicating what command in the list is replaced with the specified command and the order that the monitored commands are evaluated. If omitted, the next monitor integer is assigned. An error message is displayed if the maximum number of monitoring commands already exists.

command
A DESCRIBE, LIST, Null, or QUERY command whose output is displayed in the monitor window or log file.

HEX
Specifies that the value of the variable be displayed in hexadecimal format. You can specify the HEX parameter only with a MONITOR LIST expression command or the MONITOR n command where n is the nth command in the MONITOR list and it must be a LIST expression command.

You cannot use the HEX parameter with PL/I programs.

DEFAULT
Specifies that the value of the variable be displayed in its declared data type. You can specify the HEX parameter only with a MONITOR LIST expression command or the MONITOR n command where n is the nth command in the MONITOR list and it must be a LIST expression command.

You cannot use the DEFAULT parameter with PL/I programs.

Usage notes
• You can enter HEX or DEF in the prefix area of the monitor window to display the selected line in hexadecimal or the default representation, respectively.
The HEX and DEFAULT parameters cannot be used with PL/I programs. To display the value of a PL/I variable in hexadecimal format, use the PL/I HEX built-in function.

A monitor number identifies a global monitor command, a local monitor command, or neither.

Using Debug Tool, monitor output is presented in monitor number sequence.

If a number is provided and a command omitted, a Null command is inserted on the line corresponding to the number in the monitor window. This reserves the monitor number.

You can only specify a monitor number that is at most one greater than the highest existing monitor number.

To clear a command from the monitor, use the CLEAR MONITOR command.

The MONITOR command displays up to a maximum of 1000 lines of output in the monitor window.

Replacement only occurs if the command identified by the monitor number already exists.

The MONITOR LIST command does not allow the POPUP, TITLED, and UNTITLED options.

When using the MONITOR LIST command, simple references (or C1 values) display identifying information with the values, whereas expressions and literals do not.

The GLOBAL and LOCAL keywords also affect the default qualification for evaluation of an expression. GLOBAL indicates that the default qualification is the currently executing point in the program. LOCAL indicates that the default qualification is to the compile unit specified.

LOCAL monitors are implicitly cleared when the enclave containing the CU terminates or when the load module containing the CU is deleted.

If the DATA option of the PLAYBACK ENABLE command is in effect for the current compile unit, you can use the MONITOR command while you replay recorded statements by using the PLAYBACK commands.

A MONITOR LIST command can be evaluated only when the programming language currently in effect is the same as it was when the MONITOR LIST command was issued. Therefore, if the programming language is changed by one of the following actions, the evaluation of the MONITOR LIST command fails, and a message is displayed:

- SET PROGRAMMING LANGUAGE
- SET QUALIFY
- LOADDEBUGDATA

**Examples**

- Replace the 10th command in the monitor list with QUERY LOCATION. This is a global definition; therefore, it is always present in the monitor output.

  MONITOR 10 QUERY LOCATION;

- Add a monitor command that displays the variable abc and is local to compile unit myprog. The monitor number is the next available number.

  MONITOR LOCAL myprog LIST abc;

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

- "cu_spec" on page 13
MOVE command (COBOL)

The MOVE command transfers data from one area of storage to another. The keywords cannot be abbreviated.

```move reference to reference; literal```

**reference**
A valid Debug Tool COBOL reference.

**literal**
A valid COBOL literal.

**Usage notes**
- If Debug Tool was started because of a computational condition or an attention interrupt, using an assignment to set a variable might not give expected results. This is due to the uncertainty of variable values within statements as opposed to their values at statement boundaries.
- MOVE assigns a value only to a single receiver; unlike COBOL, multiple receiver variables are not supported.
- The COBOL CORRESPONDING phrase is not supported.
- MOVE does not support date windowing. Therefore, you cannot use the MOVE command to assign the value of a windowed date field to an expanded date field or to a non-date field.
- You cannot use the MOVE command to assign the value of one expanded date field to another expanded date field with a different DATE FORMAT clause, or to assign the value of one windowed date field to another windowed date field with a different DATE FORMAT clause.
- If the DATA parameter of the PLAYBACK ENABLE command is in effect for the current compile unit, the MOVE command can be used while you replay recorded statements by using the PLAYBACK commands. The target of the MOVE command must be a session variable, not a program variable.
- If you are debugging an optimized COBOL program, you can use the MOVE command to assign a value to a program variable only if you first enter the SET WARNING OFF command.
- If you are debugging a COBOL program that was compiled with the OPTIMIZE compiler option, neither operand of the MOVE command can be a variable that was discarded by the optimizer.
- If a COBOL variable defined as National is used as the receiving field in a MOVE command with an alphabetic or alphanumeric operand, the operand that is not National is converted to Unicode before that move is done, except for Group items. See the Enterprise COBOL for z/OS Language Reference for more information about using COBOL variables with the MOVE statement.

Chapter 5. Debug Tool commands 145
• Literals with an N or NX prefix are always treated as National data and can be moved only to other National Data Items or Group items.

Examples
• Move the string constant "Hi There" to the variable field.
  MOVE "Hi There" TO field;
• Move the value of session variable temp to the variable b.
  MOVE temp TO b;
• To assign a new value to a DBCS variable when the current programming language is COBOL, enter the following command in the Command/Log window.
  MOVE G"D B C S V A L U E"
• Assign to the program variable c, found in structure d, the value of the program variable a, found in structure b.
  MOVE a OF b TO c OF d;

Note the qualification used in this example.
• Assign the value of 123 to the first table element of itm-2.
  MOVE 123 TO itm-2(1,1);
• You can also use reference modification to assign values to variables as shown in the following two examples.
  MOVE aa(2:3) TO bb;
  and
  MOVE aa TO bb(1:4);

Refer to the following sections for more information related to the material discussed in this section.

Related tasks
Enterprise COBOL for z/OS Programming Guide

Related references
• "Allowable moves for the MOVE command (COBOL)"
• “SET WARNING (C, C++, COBOL, and PL/I)” on page 215

Allowable moves for the MOVE command (COBOL)
The following table shows the allowable moves for the Debug Tool MOVE command.
<table>
<thead>
<tr>
<th>Source field</th>
<th>GR</th>
<th>AL</th>
<th>AN</th>
<th>ED</th>
<th>BI</th>
<th>NE</th>
<th>ANE</th>
<th>NDI</th>
<th>NNDI</th>
<th>ID</th>
<th>IF</th>
<th>EF</th>
<th>DI</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP (GR)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y1</td>
<td>Y1</td>
<td>Y1</td>
<td>Y1</td>
<td>Y1</td>
<td>Y1</td>
<td>Y1</td>
<td>Y1</td>
<td>Y1</td>
<td>Y1</td>
</tr>
<tr>
<td>ALPHABETIC (AL)</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALPHANUMERIC (AN)4,5</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXTERNAL DECIMAL (ED)4,5</td>
<td>Y1</td>
<td></td>
<td>Y1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BINARY (BI)</td>
<td>Y1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMERIC EDITED (NE)</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALPHANUMERIC EDITED (ANE)</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIGCON ZERO</td>
<td>Y</td>
<td>Y</td>
<td>Y2</td>
<td>Y2</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>FIGCON ZERO, SPACE, or QUOTE</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPACES (AL)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HIGH-VALUE, LOW-VALUE, QUOTES</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NATIONAL DATA ITEM (NDI)</td>
<td>Y1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NN</td>
</tr>
<tr>
<td>NATIONAL NUMERIC DATA ITEM (NNDI)</td>
<td>Y1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NN</td>
</tr>
<tr>
<td>NUMERIC LITERAL</td>
<td>Y1</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>ALPHANUMERIC LITERAL</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>ALPHANUMERIC HEX LITERAL6</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>INTERNAL DECIMAL (ID)4,5</td>
<td>Y1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>FLOATING POINT LITERAL</td>
<td>Y1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>INTERNAL FLOATING POINT (IF)</td>
<td>Y1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>EXTERNAL FLOATING POINT (EF)</td>
<td>Y1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>DBCS DATA ITEM (DI)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>Source field</td>
<td>GR</td>
<td>AL</td>
<td>AN</td>
<td>ED</td>
<td>BI</td>
<td>NE</td>
<td>ANE</td>
<td>NDI</td>
<td>NNDI</td>
<td>ID</td>
<td>IF</td>
<td>EF</td>
<td>D1</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>DBCS LITERAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>NATIONAL LITERAL (NL)</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NATIONAL HEX LITERAL (NHL)</td>
<td>Y¹</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Notes:
1 Move without conversion (like AN to AN)
2 Numeric move
3 Decimal-aligned and truncated, if necessary
4 MOVE does not support date windowing. For example, the MOVE statement cannot be used to move a windowed date field to an expanded date field, or to a nondate field.
5 The MOVE command cannot be used to move one windowed date field to another windowed date field with a different DATE FORMAT clause, or to move one expanded date field to another expanded date field with a different DATE FORMAT clause.
6 Must be hexadecimal characters only, delimited by either double ("') or single ('') quotation marks and preceded by X.
7 Must be hexadecimal characters only, delimited by either double ("') or single ('') quotation marks and preceded by NX.

Refer to the following sections for more information related to the material discussed in this section.

- Related tasks
  - Enterprise COBOL for z/OS Programming Guide
- Related references
  - “MOVE command (COBOL)” on page 145

Null command

The Null command is a semicolon written where a command is expected. It is used for such things as an IF command with no action in its THEN clause.

Example

Do nothing if array[x] > 0; otherwise, set a to 1. The current programming language setting is C.

```c
if (array[x] > 0); else a = 1;
```

ON command (PL/I)

The ON command establishes the actions to be executed when the specified PL/I condition is raised. This command is equivalent to AT OCCURRENCE.
ON condition (condition_name) command;

ENDFILE (file_reference)

ENDPAGE
KEY
NAME
 PENDING
RECORD
TRANSMIT
UNDEFINEDFILE

AREA
ATTENTION
CONVERSION
ERROR
FINISH
FIXEDOVERFLOW
OVERFLOW
SIZE
STRINGRANGE
STRINGSIZE
SUBSCRIPTRANGE
UNDERFLOW
ZERODIVIDE

condition_name
A valid PL/I CONDITION condition name.

file_reference
A valid PL/I file constant or file variable (can be qualified).

command
A valid Debug Tool command.

Usage notes
• You must abide by the PL/I restrictions for the particular condition.
• An ON action for a specified PL/I condition remains established until:
  – Another ON command establishes a new action for the same condition. In
    other words, the breakpoint is replaced.
  – A CLEAR command removes the ON definition.
• The ON command occurs before any existing ON-unit in your application program.
The ON-unit is processed after Debug Tool returns control to the language.
• The following are accepted PL/I abbreviations for the PL/I condition constants:
  ATTENTION or ATTN
  FIXEDOVERFLOW or FOFL
  OVERFLOW or OFL
  STRINGRANGE or STRG
  STRINGSIZE or STRZ
  SUBSCRIPTRANGE or SUBRG
  UNDEFINEDFILE[[file_reference]] or UNDF[[file_reference]]
  UNDERFLOW or UFL
  ZERODIVIDE or ZDIV
• The preferred form of the ON command is AT OCCURRENCE. For compatibility with
  PLITEST and INSPECT, however, it is recognized and processed. ON should be
  considered a synonym of AT OCCURRENCE. Any ON commands entered are logged
  as AT OCCURRENCE commands.
• The ON command cannot be used while you replay recorded statements by using
  the PLAYBACK commands.
Examples

- Display a message if a division by zero is detected.
  
  ```pli
  ON ZERODIVIDE BEGIN;
  LIST 'A zero divide has been detected';
  END;
  ```

- Display and patch the error character when converting character data to numeric.
  
  Given a PL/I program that contains the following statements:
  
  ```pli
  DECLARE i FIXED BINARY(31,0);
  .
  ..
  i = '1s3';
  ```

  The following Debug Tool command would display and patch the error character when converting the character data to numeric:

  ```pli
  ON CONVERSION BEGIN;
  LIST (%STATEMENT, ONCHAR);
  ONCHAR = '0';
  GO;
  END;
  ```

  '1s3' cannot be converted to a binary number so CONVERSION is raised. The ON CONVERSION command lists the offending statement number and the offending character: 's'. The data will be patched by replacing the 's' with a character zero, 0, and processing will continue.

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

- "AT OCCURRENCE" on page 60
- [Enterprise PL/I for z/OS Language Reference](#)
the three windows are evenly divided. For other configurations, the point where the three windows meet is approximately the center of the screen.

**LISTINGS**

Displays the Source Identification panel, where associations are made between source listings or source files shown in the Source window and their program units. LISTINGS is equivalent to SOURCES.

Debug Tool provides the Source Identification panel to maintain a record of compile units associated with your program, as well as their associated source or listing.

You can also make source or listings available to Debug Tool by entering their names on the Source Identification panel.

The Source Identification panel associates compile units with the names of their respective listing or source files and controls what appears in the Source window. To explicitly name the compile units being displayed in the source window, access the Source Identification panel (shown below) by entering the PANEL LISTINGS or PANEL SOURCES command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Source Identification Panel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile Unit</td>
<td>Listings/Source File</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>DBKPS15</td>
<td>T564081.TESTLISTING(IBME73)</td>
</tr>
</tbody>
</table>

Enter QUIT to return with current settings saved.
CANCEL to return without current settings saved.
UP/DOWN to scroll up and down.

**Compile Unit**

Is the name of a valid compile unit currently known to Debug Tool. New compile units are added to the list as they become known.

**Listing/Source File**

Is the name of the listing or source file containing the compilation unit to be displayed in the Source window. If the file is a listing, only source program statements are shown. The minimum required is the compile unit name. The default file specification is pgmname LISTING * (COBOL and PL/I), where pgmname is the name of your program. For TSO, the default file specification is userid.pgmname.C (C and C++), userid.pgmname.list (COBOL), or userid.pgmname.list (PL/I) for sequential data sets and userid.dsnname.C(membername) (C and C++), userid.dsnname.Listing(membername) (COBOL), or userid.dsnname.List(membername) (PL/I) for partitioned data sets.

**Display**

Is a flag that specifies whether the listing or source is to be displayed in the Source window.

To display a listing view, take the following steps:

- Compile the program with the proper option to generate a source or source listing file.
- Make sure the file is available and accessible on your host operating system.
- Set the Display field on the Source Identification panel to Y for the compile unit. To save time and avoid displaying listings or source you do not want to see, specify N.
If any of these conditions are not satisfied, the Source window remains empty until control reaches a compile unit where the conditions are satisfied.

You can change the source or source listing associated with a compile unit by entering the new name over the source or source listing file displayed in the LISTING/SOURCE FILE field.

Note: The new name must be followed by at least one blank.

After you modify the panel, return to the Debug Tool session panel either by issuing the QUIT command, or by pressing the QUIT PF key.

**PROFILE**
Displays the Profile Settings panel, where parameters of a full-screen Debug Tool session can be set.

**SOURCES**
Is equivalent to LISTINGS.

**Usage notes**
- All information on the panels displayed by the PANEL command is saved when QUIT is used to leave them. Saving the changes to the specified panels in this manner returns you to your Debug Tool session with the current settings in effect. In addition, CANCEL can be used to leave the panels without saving the changes.
- The PANEL command is not logged.

**Examples**
- Display the color and attribute panel.
  
  PANEL COLORS;

- Reset the relative sizes of the windows for the current layout configuration.
  
  PANEL LAYOUT RESET;

Refer to the following sections for more information related to the material discussed in this section.

**Related tasks**
[Debug Tool User's Guide]

---

**PERFORM command (COBOL)**

The PERFORM command transfers control explicitly to one or more statements and implicitly returns control to the next executable statement after execution of the specified statements is completed. The keywords cannot be abbreviated.

**Simple:**

```
PERFORM command END-PERFORM;
```

*command*
  
  A valid Debug Tool command.

**Repeating:**

---
reference
A valid Debug Tool COBOL reference.

condition
A simple relation condition.

command
A valid Debug Tool command.

Usage notes
• A constant as a reference is allowed only on the right side of the FROM and BY keywords.
• Index-names and floating point variables cannot be used as the VARYING references.
• Index-names are not supported in the BY phrase.
• Only inline PERFORMs are supported (but the performed command can be a Debug Tool procedure invocation).
• The COBOL AFTER phrase is not supported.
• Windowed date fields cannot be used as the VARYING reference, the FROM reference, or the BY reference.
• See Enterprise COBOL for z/OS Language Reference for an explanation of the following COBOL keywords:
  
  ```COBOL
  AFTER
  BEFORE
  BY
  FROM
  TEST
  UNTIL
  VARYING
  WITH
  ```
  
  • For optimized COBOL programs, the PERFORM command cannot reference any variable that was discarded by the optimizer.
  • For optimized COBOL programs, if the VARYING phrase is specified, the first reference can only refer to a session variable.
  • If you entered the PLAYBACK ENABLED with the DATA parameter and the compile unit supports the DATA parameter, the PERFORM command can reference a program variable and the VARYING operand (if specified) must reference a session variable. For example:
PERFORM VARYING session-var-1 FROM program-var-1 BY program-var-2
  UNTIL program-var-3 = program-var-4

Examples
• Set a breakpoint at statement number 10 to move the value of variable a to the
  variable b and then list the value of x.
  AT 10 PERFORM
    MOVE a TO b;
    LIST (x);
  END-PERFORM;
• List the value of height for each even value between 2 and 30, including 2 and
  30.
  PERFORM WITH TEST AFTER
    VARYING height FROM 2 BY 2
      UNTIL height = 30
    LIST height;
  END-PERFORM;
• Position the cursor at the start of a COBOL performed paragraph and press PF5.

Refer to the following sections for more information related to the material
discussed in this section.

Related references
Enterprise COBOL for z/OS Language Reference

PLAYBACK commands

The PLAYBACK commands help you record and replay:
• Statements that you have run.
• Information about your program. For example, the value of variables and
  registers and the status of files.

The following table summarizes the forms of the PLAYBACK commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;PLAYBACK ENABLE command&quot; on page 156</td>
<td>Informs Debug Tool to record all subsequent statements that you run and other information about your program.</td>
</tr>
<tr>
<td>&quot;PLAYBACK START command&quot; on page 157</td>
<td>Informs Debug Tool to suspend normal debugging and to prepare to replay recorded statements.</td>
</tr>
<tr>
<td>&quot;PLAYBACK FORWARD command&quot; on page 158</td>
<td>Informs Debug Tool to replay recorded statements in forward direction.</td>
</tr>
<tr>
<td>&quot;PLAYBACK BACKWARD command&quot; on page 158</td>
<td>Informs Debug Tool to replay recorded statements in backward direction.</td>
</tr>
<tr>
<td>&quot;PLAYBACK STOP command&quot; on page 159</td>
<td>Informs Debug Tool to stop replaying statements, resume normal debugging, and continue recording the statements that you run and other information about your program.</td>
</tr>
<tr>
<td>&quot;PLAYBACK DISABLE command&quot; on page 159</td>
<td>Informs Debug Tool to stop recording the statements that you run and discard the information about your program that it recorded.</td>
</tr>
</tbody>
</table>

Usage note

You cannot use the PLAYBACK commands while you debug a disassembled program.
PLAYBACK ENABLE command

The PLAYBACK ENABLE command informs Debug Tool to begin recording the statements that you run and information about your program. If Debug Tool is already recording the statements that you run, you can use the PLAYBACK ENABLE command to inform Debug Tool to record the statements that you run in other compile units or to change the effect of the DATA option.

```
PLAYBACK ENABLE options ;
```

**options:**

- **cuname**
  - Name of the compile unit or compile units where Debug Tool is to record the statements that you run. You can specify only the names of the compile units currently known.
  - * Specifies that Debug Tool is to record the statements that you run in all compile units. This is the default.

- **integer**
  - Specifies the maximum amount of memory to use to store data that is collected. The integer value specifies a unit of K (1024) bytes. For example, an integer value of 2000 indicates 2,048,000 bytes. The default value is 8000.

**DATA**

Specifies that Debug Tool is to save information about your program, such as the value of variables and registers. Debug Tool saves this information for the compile units that you specify in the cuname parameter or, if you specified the * parameter, for all compile units. The DATA parameter is effective only for compile units compiled with the SYM suboption of the TEST compiler option and with one of the following compilers:
- Enterprise COBOL for z/OS and OS/390, Version 3 Release 2
- Enterprise COBOL for z/OS and OS/390, Version 3 Release 1, with APAR PQ63235
- COBOL for OS/390 & VM, Version 2, with APAR PQ63234

You also need to have the following Language Environment APAR installed:
- z/OS, Version 1 Release 4, with APAR PQ65176

DATA is the default.

**NODATA**

Specifies that Debug Tool does not save information about your program.

**Notes:**

1. Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

**Usage notes**
• For COBOL only: If you enter the PLAYBACK ENABLE DATA command, and a compile unit supports the DATA parameter, the following information is recorded:
  - FILE SECTION
  - WORKING-STORAGE SECTION
  - LOCAL-STORAGE SECTION
  - LINKAGE SECTION
  - All special registers except: ADDRESS OF, LENGTH OF, and WHEN-DEFINED

**PLAYBACK START command**

The PLAYBACK START command suspends normal debugging and informs Debug Tool to prepare to replay the statements it recorded. When normal debugging is suspended, all breakpoints are disabled and many commands are unavailable. Use the STEP and RUNTO commands to navigate through recorded statements in a forward or backward direction. Backward is the initial direction of the navigation.

(1)

![PLAYBACK—START](image)

**Notes:**

1 Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

**Usage notes**

The following commands are available while you replay recorded statements:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ALLOCATE command&quot; on page 31</td>
<td>FIND command</td>
</tr>
<tr>
<td>CALL procedure</td>
<td>&quot;FREE command&quot; on page 110</td>
</tr>
<tr>
<td>CLEAR EQUATE</td>
<td>IMMEDIATE command (full-screen mode)</td>
</tr>
<tr>
<td>CLEAR LOG</td>
<td>null (full-screen mode)</td>
</tr>
<tr>
<td>CLEAR MONITOR</td>
<td>PANEL command (full-screen mode)</td>
</tr>
<tr>
<td>CLEAR PROCEDURE</td>
<td>&quot;PERFORM command (COBOL)&quot; on page 155</td>
</tr>
<tr>
<td>COMMENT command</td>
<td>&quot;PLAYBACK commands&quot; on page 155</td>
</tr>
<tr>
<td>CURSOR command (full-screen mode)</td>
<td>Prefix commands (full-screen mode)</td>
</tr>
<tr>
<td>Declarations (COBOL)</td>
<td>&quot;PROCEDURE command&quot; on page 160</td>
</tr>
<tr>
<td>DESCRIBE CUS</td>
<td>QUERY command</td>
</tr>
<tr>
<td>DESCRIBE PROGRAMS</td>
<td>QUIT command</td>
</tr>
<tr>
<td>SYSTEM command (z/OS)</td>
<td>SET (most forms)</td>
</tr>
<tr>
<td>TSO command (z/OS)</td>
<td>TSO command (z/OS)</td>
</tr>
</tbody>
</table>

1Refer to “PERFORM command (COBOL)” on page 153 for restrictions.
If the DATA option is in effect and the compile unit supports the DATA option, the following commands are available:

<table>
<thead>
<tr>
<th>Command/Command (COBOL)</th>
<th>Command/Command (COBOL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPUTE</td>
<td>LIST</td>
</tr>
<tr>
<td>DESCRIBE ATTRIBUTES</td>
<td>&quot;MOVE command (COBOL)&quot; on page 145</td>
</tr>
<tr>
<td>DESCRIBE CURSOR</td>
<td>MONITOR</td>
</tr>
<tr>
<td>EVALUATE</td>
<td>&quot;SET command (COBOL)&quot; on page 216</td>
</tr>
<tr>
<td>IF</td>
<td>&quot;SET AUTOMONITOR&quot; on page 177</td>
</tr>
</tbody>
</table>

2 The target must be session variable.

The following commands are not available while you replay recorded statements:

<table>
<thead>
<tr>
<th>Command/Command (PL/I)</th>
<th>Command/Command (C and C++)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALYZE</td>
<td>&quot;Declarations (C and C++)&quot; on page 87</td>
</tr>
<tr>
<td>DECLARE command (PL/I)</td>
<td>IF command (C and C++)</td>
</tr>
<tr>
<td>&quot;Assignment command (assembler and disassembly)&quot; on page 35</td>
<td></td>
</tr>
<tr>
<td>ASSIGNMENT command (PL/I)</td>
<td>DESCRIBE ENVIRONMENT</td>
</tr>
<tr>
<td>AT</td>
<td>DISABLE command</td>
</tr>
<tr>
<td>&quot;break command (C and C++)&quot; on 69</td>
<td>ON command (PL/I)</td>
</tr>
<tr>
<td>CALL %DUMP</td>
<td>&quot;DO command (PL/I)&quot; on page 99</td>
</tr>
<tr>
<td>CALL entry_name (COBOL)</td>
<td>ENABLE command</td>
</tr>
<tr>
<td>CLEAR AT</td>
<td>SET INTERCEPT</td>
</tr>
<tr>
<td>CLEAR DECLARE</td>
<td>Expression command (C and C++)</td>
</tr>
<tr>
<td>CLEAR ON</td>
<td>switch command (C and C++)</td>
</tr>
<tr>
<td>CLEAR VARIABLES</td>
<td>TRIGGER command</td>
</tr>
<tr>
<td>CLEAR GO</td>
<td>while command (C and C++)</td>
</tr>
<tr>
<td>CLEAR VTO</td>
<td>SET command (PL/I)</td>
</tr>
</tbody>
</table>

**PLAYBACK FORWARD command**

The PLAYBACK FORWARD command informs Debug Tool to perform STEP and RUNTO commands forward, starting from the current statement and going to the next statement.

```
(1)
  >>> PLAYBACK—FORWARD ;
```

**Notes:**
1 Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

**PLAYBACK BACKWARD command**

The PLAYBACK BACKWARD command informs Debug Tool to perform STEP and RUNTO commands backward, starting from the current statement and going to previous statements. Backward is the initial direction when you enter the PLAYBACK START command.
Notes:
1 Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

PLAYBACK STOP command
The PLAYBACK STOP command resumes normal debugging at the statement where you entered the PLAYBACK START command. All suspended breakpoints are enabled and all commands are available. Debug Tool continues to record the statements you run and, if you specified the DATA option, information about your program.

Notes:
1 Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

PLAYBACK DISABLE command
The PLAYBACK DISABLE command informs Debug Tool to stop recording the statements that you run and, if you specified the DATA option, information about your program. The information about the program that Debug Tool collected while recording is discarded. You can instruct Debug Tool to stop recording for one or more compile units. If you stop recording for one compile unit and continue recording for other compile units, the information that you collected for the one compile unit is discarded.

Notes:
1 Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

cuname
Indicates to Debug Tool to stop recording for the compile unit or compile units specified. Only the names of currently known compile units can be specified.

* Indicates to Debug Tool to stop recording for all compile units. This is the default.

Prefix commands (full-screen mode)
The prefix commands apply to source listing lines and monitor lines. Prefix commands are commands that are typed into the prefix area of the source window or monitor window. For more information about the commands, see the section corresponding to the command name.
The following tables summarize the forms of the prefix commands.

Table 5. Source window prefix commands

<table>
<thead>
<tr>
<th>Prefix Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;AT Prefix (full-screen mode)&quot; on page 64</td>
<td>Defines a statement breakpoint through the Source window prefix area.</td>
</tr>
<tr>
<td>&quot;CLEAR prefix (full-screen mode)&quot; on page 83</td>
<td>Clears a breakpoint through the Source window prefix area.</td>
</tr>
<tr>
<td>&quot;DISABLE prefix (full-screen mode)&quot; on page 97</td>
<td>Disables a breakpoint through the Source window prefix area.</td>
</tr>
<tr>
<td>&quot;ENABLE prefix (full-screen mode)&quot; on page 102</td>
<td>Enables a disabled breakpoint through the Source window prefix area.</td>
</tr>
<tr>
<td>&quot;QUERY prefix (full-screen mode)&quot; on page 166</td>
<td>Queries what statements have breakpoints through the Source window prefix area.</td>
</tr>
<tr>
<td>&quot;RUNTO prefix command (full-screen mode)&quot; on page 171</td>
<td>Runs the program to the location that the cursor or statement identifier indicate in the Source window prefix area.</td>
</tr>
<tr>
<td>&quot;SHOW prefix command (full-screen mode)&quot; on page 220</td>
<td>Specifies what relative statement or verb within the line is to have its frequency count shown in the suffix area.</td>
</tr>
</tbody>
</table>

Table 6. Monitor window prefix commands

<table>
<thead>
<tr>
<th>Prefix Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEX (MONITOR n HEX)&quot;MONITOR command&quot; on page 142</td>
<td>Displays selected member of the current set of MONITOR commands in hexadecimal representation.</td>
</tr>
<tr>
<td>DEF (MONITOR n DEFAULT)&quot;MONITOR command&quot; on page 142</td>
<td>Displays selected member of the current set of MONITOR commands in default representation.</td>
</tr>
<tr>
<td>CL (CLEAR MONITOR n)&quot;CLEAR command&quot; on page 79</td>
<td>Clears selected member of the current set of MONITOR commands.</td>
</tr>
<tr>
<td>LIST (LIST MONITOR n) &quot;LIST MONITOR&quot; on page 134</td>
<td>Lists selected member of the current set of MONITOR commands.</td>
</tr>
</tbody>
</table>

**PROCEDURE command**

The PROCEDURE command allows the definition of a group of commands that can be accessed by using the CALL procedure command. The CALL command is the only way to perform the commands within the PROCEDURE. PROCEDURE definitions remain in effect for the entire debug session.

The PROCEDURE keyword can be abbreviated only as PROC. PROCEDURE definitions can be subcommands of other PROCEDURE definitions. The name of a nested procedure has the scope of only the containing procedure. Session variables cannot be declared within a PROCEDURE definition.

In addition, a procedure must be defined before it is called on a CALL statement.
name
A valid Debug Tool procedure name. It must be a valid identifier in the current programming language. The maximum length is 31 characters.

command
A valid Debug Tool command other than a declaration or PANEL command.

Usage notes
• Because the Debug Tool procedure names are always uppercase, the procedure names are converted to uppercase even for programming languages that have mixed-case symbols.
• If a GO or STEP command is issued within a procedure or a nested procedure, any statements following the GO or STEP in that procedure and the containing procedure are ignored. If control returns to Debug Tool, it returns to the statement following the CALL of the containing PROCEDURE.
• It is recommended that procedure names be chosen so that they are valid for all possible programming language settings throughout the entire Debug Tool debug session.

Examples
• When procedure proc1 is called, the values of variables x, y, and z are displayed.
  proc1: PROCEDURE; LIST (x, y, z); END;

  Define a procedure named setat34 that sets a breakpoint at statement 34.
  Procedure setat34 contains a nested procedure lister that lists current statement breakpoints. Procedure lister can be called only from within setat34.
  setat34: PROCEDURE;
      AT 34;
      lister: PROCEDURE;
          LIST AT STATEMENT;
      END;
      CALL lister;
      END;

QUALIFY RESET command
The QUALIFY RESET command is equivalent to the SET QUALIFY RESET command.

QUERY command
The QUERY command displays the current value of the specified Debug Tool setting, the current setting of all the Debug Tool settings, or the current location in the suspended program.

For an explanation of the Debug Tool settings, see the SET command.
Notes:
1. Available only with Debug Tool Utilities and Advanced Functions (5655-R45).
2. You can use this command in remote debug mode when you use one of the following remote debuggers:
   - Compiled Language Debugger component of WebSphere Studio Enterprise Developer
   - Compiled Language Debugger component of WebSphere Developer for zSeries
   - WebSphere Developer Debugger for zSeries
3. Available only if the Dynamic Debug facility is installed.
4. Only for PL/I.

**ASSEMBLER**
Displays the current ASSEMBLER setting.

**AUTOMONITOR**
Displays the current AUTOMONITOR setting.

**CHANGE**
Displays the current CHANGE setting.

**COLORS (full-screen mode)**
Displays the current COLOR setting.

**COUNTRY**
Displays the current COUNTRY setting.

**CURRENT VIEW**
Displays the name of the view being used for the currently qualified CU.

**DBCS**
Displays the current DBCS setting.

**DEFAULT LISTINGS**
Displays the current DEFAULT LISTINGS setting.

**DEFAULT SCROLL (full-screen mode)**
Displays the current DEFAULT SCROLL setting.

**DEFAULT VIEW**
Displays the name of the view that will be used as the initial view when you enter the LOADDEBUGDATA command for an assembler CU.

**DEFAULT WINDOW (full-screen mode)**
Displays the current DEFAULT WINDOW setting.

**DISASSEMBLY**
Displays the current DISASSEMBLY setting.

**DYNDDEBUG**
Displays the current DYNDDEBUG setting.

**ECHO**
Displays the current ECHO setting.

**EQUATES**
Displays the current EQUATE definitions.

**EXECUTE**
Displays the current EXECUTE setting.
FREQUENCY
 Displays the current FREQUENCY setting.

HISTORY
 Displays the current HISTORY setting and size.

INTERCEPT
 Displays the current INTERCEPT setting.

KEYS (full-screen mode)
 Displays the current KEYS setting.

LDD
 Displays the current LDD setting.

LIST TABULAR
 Displays the current LIST TABULAR setting.

LOCATION
 Displays the statement identifier where execution is suspended. The current statement identified by QUERY LOCATION has not yet executed. If suspended at a breakpoint, the description of the breakpoint is also displayed.

LOG
 Displays the current LOG setting.

LOG NUMBERS (full-screen mode)
 Displays the current LOG NUMBERS setting.

LONGCUNAME
 Displays the current LONGCUNAME setting.

MONITOR COLUMN
 Displays the current MONITOR COLUMN setting. SET MONITOR COLUMN is accepted in batch mode, but has no effect.

MONITOR DATATYPE
 Displays the current MONITOR DATATYPE setting.

MONITOR NUMBERS (full-screen mode)
 Displays the current MONITOR NUMBERS setting.

MONITOR WRAP
 Displays the current MONITOR WRAP setting. SET MONITOR WRAP is accepted in batch mode, but has no effect.

MSGID
 Displays the current MSGID setting.

NATIONAL LANGUAGE
 Displays the current NATIONAL LANGUAGE setting.

PACE
 Displays the current PACE setting. This setting is not supported in batch mode.

PFKEYS
 Displays the current PFKEY definitions. This setting is not supported in batch mode.

PLAYBACK
 Displays the current status of PLAYBACK.

PLAYBACK LOCATION
 Displays the statement identifier of the statement being replayed.
PROGRAMMING LANGUAGE
Displays the current PROGRAMMING LANGUAGE setting. Debug Tool does not differentiate between C and C++, use this option for C++ as well a C programs.

PROMPT (full-screen mode)
Displays the current PROMPT setting.

QUALIFY
Displays the current QUALIFY BLOCK setting.

REFRESH (full-screen mode)
Displays the current REFRESH setting.

RESTORE
Displays the current RESTORE setting.

REWRITE
Displays the current REWRITE setting. This setting is not supported in batch mode.

SAVE
Displays the current SAVE setting.

SCREEN (full-screen mode)
Displays the current SCREEN setting.

SCROLL DISPLAY (full-screen mode)
Displays the current SCROLL DISPLAY setting.

SEQUENCE (PL/I)
Displays current SEQUENCE setting.

SETS
Displays all settings that are controlled by the SET command.

SOURCE
Displays the current SOURCE setting.

SUFFIX (full-screen mode)
Displays the current SUFFIX setting.

TEST
Displays the current TEST setting.

WARNING (C)
Displays the current WARNING setting.

WINDOW SIZES
Displays the current WINDOW SIZE values and WINDOW CLOSE information. The window sizes are the values that apply when all windows are open.

Examples
• Display the current ECHO setting.
  QUERY ECHO;
• Display all current settings.
  QUERY SETS;

Refer to the following sections for more information related to the material discussed in this section.

Related references
"QUERY prefix (full-screen mode)" on page 166
QUERY prefix (full-screen mode)

Queries what statements on a particular line have statement breakpoints when you issue this command through the Source window prefix area.

Usage notes

• When the QUERY prefix command is issued, a sequence of characters corresponding to the statements is displayed in the prefix area of the Source window. If the statement contains a breakpoint, "*" is used, or ".", if it does not. If there are more than eight statements or verbs on the line, and one or more past the eighth statement have breakpoints, the eighth character of the map is replaced by a "+".
  
For example, a display of ".*." indicates that four statements or verbs begin on the line and the third one has a breakpoint defined.

• The QUERY prefix command is not logged.

Related references

"LIST command" on page 124

QUIT command

The QUIT command ends a Debug Tool session and, if an expression is specified, sets the return code. It also displays a prompt panel (full-screen) that asks if you really want to quit the debug session. In line and batch mode, the QUIT command ends the session without prompting.

expression

A valid Debug Tool expression in the current programming language.

If expression is specified, this value is used as the application return code value. The actual return code for the run is determined by the execution environment.

ABEND

If you specify ABEND, Debug Tool raises a CEE2F1 exception to terminate each active enclave.

DEBUG

If you specify DEBUG, Debug Tool ends and your program keeps running. Any calls to restart Debug Tool are ignored. By default, when running under CICS, a pseudo-conversational application will run until the end of the conversation (until EXEC CICS RETURN without TRANSID is issued to return to CICS).

TASK

TASK applies to CICS pseudo-conversational applications. If you specify TASK, Debug Tool processing will be terminated until the end of the current CICS pseudo-conversational task (EXEC CICS RETURN TRANSID). When a new task is started in the pseudo-conversation, Debug Tool debugging will resume.
Usage notes
- Debug Tool will only resume in a new pseudo-conversational task if CADP or DTCN successfully match on a pattern.
- QUIT is always logged in a comment line except where it appears in a command list. This enables you to reuse the log file as a primary commands file.
- If QUIT is entered from a Debug Tool commands file, no prompt is displayed. This behavior applies to the Debug Tool preferences files, primary commands files, and USE files.
- For PL/I, the expression will be converted to FIXED BINARY (31,0), if necessary. In addition, if an expression is specified, it is used as if your program called the PLIRETC built-in subroutine.
- For PL/I, the value of the expression must be nonnegative and less than 1000.
- If you enter the QUIT DEBUG command and then want to restart Debug Tool, you must first restart your program.
- If you enter the QUIT or QQUIT command while you are debugging a non-Language Environment assembler or OS/VS COBOL program running under CICS, Debug Tool behaves the same as if you entered a QUIT ABEND command and a U4038 abend occurs.

Examples
- End a Debug Tool session.
  QUIT;
- End a Debug Tool session and use the value in variable x as the application return code.
  QUIT (x);
- End a Debug Tool session without ending the program.
  QUIT DEBUG;

Refer to the following sections for more information related to the material discussed in this section.

Related references
"expression" on page 14

QQUIT command
The QQUIT command ends a Debug Tool session without further prompting.

QUIT;

Usage notes
- In full-screen mode, the QQUIT command does not display a prompt panel to verify that you want to quit the debug session.
- If you enter the QQUIT command while you are debugging a non-Language Environment assembler or OS/VS COBOL program running under CICS, Debug Tool behaves the same as if you had entered the QUIT ABEND command and a U4038 abend occurs.

Example
End a Debug Tool session.
QUIT;
RESTORE command

The RESTORE command enables you to explicitly restore the settings, breakpoints, and monitor specifications that were previously saved by the SET SAVE AUTO command when Debug Tool terminated.

**SETTINGS**
Indicates that all SET values except the following values are to be restored:
- SET DBCS
- SET FREQUENCY
- SET NATIONAL LANGUAGE
- SET PROGRAMMING LANGUAGE
- FILE operand of SET RESTORE SETTINGS
- SET QUALIFY
- SET SOURCE
- SET TEST

**BPS**
Indicates that breakpoints and LOADDEBUGDATA (LDD) specifications are to be restored. The following breakpoints are restored:
- APPEARANCE breakpoints
- CALL breakpoints
- DELETE breakpoints
- ENTRY breakpoints
- EXIT breakpoints
- GLOBAL APPEARANCE breakpoints
- GLOBALCALL breakpoints
- GLOBAL DELETE breakpoints
- GLOBAL ENTRY breakpoints
- GLOBAL EXIT breakpoints
- GLOBAL LABEL breakpoints
- GLOBAL LOAD breakpoints
- GLOBAL STATEMENT and GLOBAL LINE breakpoints
- LABEL breakpoints
- LOAD breakpoints
- OCCURRENCE breakpoints
- STATEMENT and LINE breakpoints
- TERMINATION breakpoint
If a deferred AT ENTRY breakpoint has not been encountered, it is not saved nor restored.

**MONITORS**
Indicates that monitor and LOADDEBUGDATA (LDD) specifications are to be restored.

**Usage notes**
- The data restored by this command is retrieved from the default data set or the data set specified by the SET RESTORE SETTINGS, SET RESTORE BPS, or SET RESTORE MONITORS commands.
- The member name used to restore the breakpoints or monitor specifications is the name of the initial load module for the current enclave.
- Do not use the RESTORE command after you have entered any of the following commands:
  - AT
  - MONITOR
  - GO
  - STEP

**Example**
- Restore the settings:
  RESTORE SETTINGS;
- Restore the breakpoints and monitor specifications:
  RESTORE BPS MONITORS;

Refer to the following sections for more information related to the material discussed in this section.

**Related tasks**
- [Debug Tool User’s Guide](#)

**Related references**
- “SET RESTORE” on page 205
- “SET SAVE” on page 207

---

**RETRIEVE command (full-screen mode)**

The RETRIEVE command displays the last command entered on the command line.
For long commands this might be only the last line of the command.

```
RETRIEVE COMMAND ;
```

**COMMAND**
Retrieves commands. Any command retrieved to the command line can be performed by pressing Enter. The retrieved command can also be modified before it is performed. Successive RETRIEVE commands continue to display up to 12 commands previously entered on the command line. This operand is most useful when assigned to a PF key.

**Usage notes**
- The RETRIEVE command is not logged.

**Example**
Retrieve the last line so that it can be reissued or modified.

RETrieve COMmand;

**RUN command**

The **RUN** command is synonymous to the **GO** command.

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

“**GO command**” on page 110

**RUNTO command**

The **RUNTO** command runs your program to a valid executable statement without setting a breakpoint. You can indicate at which statement to stop by specifying the statement id or by positioning the cursor on a statement.

```
  RUNTO [statement_id];
```

**statement_id**
A valid statement identifier. If you are debugging a disassembled program, specify the statement identifier as an offset in hexadecimal form (X'offset').

**Usage notes**

- If you indicate a statement by positioning the cursor on the statement, the cursor must be in the Source window and positioned on a line where an executable statement begins.
- If you indicate a statement by positioning the cursor on the statement and there are multiple statements on the same line, the target of the **RUNTO** command is the first relative statement on the line. For optimized COBOL programs, the target of the command is the first executable command which was not discarded by the optimizer.
- If you indicate a statement by providing a statement id, the statement id must be an executable statement.
- Execution continues until one of the following conditions occurs:
  - The location indicated by the cursor position or the statement id is reached.
  - A previously set breakpoint is encountered.
  - The end of the job is reached.
- For optimized COBOL programs, the **RUNTO** command remains in effect until the statement you indicated is reached. For example, if your program encounters a breakpoint and then you enter the **GO** or **RUN** command, the program runs until the next breakpoint is encountered or the statement you indicated is reached.

**Examples**

- Run to statement 67, where statement 67 is in a currently active block.
  
  ```
  RUNTO 67;
  ```

- Run to the statement 11 in the block IPL111A, where IPL111A is known in the current enclave.
  
  ```
  RUNTO IPL111A :> 11
  ```

- Run to statement 36, where statement 36 is located in the Source window.
1. Type RUNTO in the command line.
2. Place the cursor on statement 36.
3. Press Enter.

- Run to the statement 74, using a PF key.
  1. Define a PF key to run to the cursor position.
     
     ```
     SET PF13 = RUNTO;
     ```
  2. Place the cursor at the statement 74 and hit shift+PF1 key.

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

- “RUN command” on page 170

### RUNTO prefix command (full-screen mode)

Runs to the statement when you issue this command through the Source window prefix area.

#### Usage notes

- For RUNTO prefix, no space is needed as a delimiter between the keyword and the integer; RUNTO 67 is equivalent to RUNTO67.
- For optimized COBOL programs, if there are multiple statements on a line, the RUNTO prefix runs to the first executable statement which was not discarded by the optimizer.

#### Example

Run to the statement 67, where statement 67 is located in the Source window.

- Type RUNTO in the prefix area of statement 67, then press Enter.

### SCROLL command (full-screen mode)

The SCROLL command provides horizontal and vertical scrolling in full-screen mode. Scroll commands can be made immediately effective with the IMMEDIATE command. The SCROLL keyword is optional.

The Log, Monitor, or Source window will not wrap around when scrolled.

- `DOWN` Scrolls the specified number of lines in a window toward the top margin of that window. `DOWN` is equivalent to NEXT.
LEFT
Scrolls the specified number of columns in a window toward the right margin of that window. If SET MONITOR WRAP OFF is in effect, using LEFT allows you to scroll toward the right the specified number of characters in the monitor value area so data that is not visible to the left becomes visible.

NEXT
Is equivalent to DOWN.

RIGHT
Scrolls the specified number of columns in a window toward the left margin of that window. If SET MONITOR WRAP OFF is in effect, using RIGHT allows you to scroll toward the left the specified number of characters in the monitor value area so data that is not visible to the right becomes visible.

UP
Scrolls the specified number of lines in a window toward the bottom margin of that window.

CSR
Specifies scrolling based on the current position of the cursor in a selected window. The window scrolls up, down, left, or right of the cursor position until the character where the cursor is positioned reaches the edge of the window. If the cursor is not in a window or if it is already positioned at the edge of a window, a full-page scroll occurs. If the cursor is in the monitor value area then the monitor value area is scrolled left or right to the position of the cursor.

DATA
Scrolls by one line less than the window size or by one character less than the window size (if moving left or right). If the cursor is in the monitor value area then the monitor value area scrolls left or right by one character less than the monitor value area width.

HALF
Scrolls by half the window size or by half the monitor value area.

integer
Scrolls the specified number of lines (up or down) or the specified number of characters (left or right). Maximum value is 9999.

MAX
Scrolls in the specified direction until the limit of the data is reached. To scroll the maximum amount, you must use the MAX keyword. You cannot scroll the maximum amount by filling in the scroll amount field. If the cursor is placed in the monitor value area then the monitor value area is scrolled left or right until the limit of the data is reached.

PAGE
Scrolls by the window size or by the monitor value area size.

BOTTOM
Scrolls to the bottom of the data.

TO integer
Specifies that the selected window is to scroll to the given line (as indicated in the prefix area of the selected window). This can be in either the UP or DOWN direction (for example, if you are line 30 and issue TO 20, it will return to line 20). Maximum value is 999999.

TOP
Scrolls to the top of the data.
CURSOR
   Selects the window where the cursor is currently positioned.

LOG
   Selects the session log window.

MONITOR
   Selects the monitor window.

SOURCE
   Selects the source listing window.

Usage notes
   • If you do not specify an operand with the DOWN, LEFT, NEXT, RIGHT, or UP keywords, and the cursor is outside the window areas, the window scrolled is determined by the current default window setting (if the window is open) and the scroll amount is determined by the current default scroll setting, shown in the SCROLL field on the Debug Tool session panel. Default scroll and default window settings are controlled by SET DEFAULT SCROLL and SET DEFAULT WINDOW commands.
   • When the SCROLL field on the Debug Tool session panel is typed over with a new value, the equivalent SET DEFAULT SCROLL command is issued just as if you had typed the command into the command line (that is, it is logged and retrievable).
   • The SCROLL command is not logged.
   • To scroll the monitor value area left or right, SET MONITOR WRAP OFF must be in effect and the cursor must be in the monitor value area.

Examples
   • Scroll one page down in the window containing the cursor.
     SCROLL DOWN PAGE CURSOR;
   • Scroll the monitor window 12 columns to the left.
     SCROLL LEFT 12 MONITOR;
   • Scroll the monitor value window 15 columns to the right.
     SET MONITOR WRAP OFF; SCROLL RIGHT 15;

     (Do not press Enter.) Place cursor in the monitor value area. Press Enter.

Refer to the following sections for more information related to the material discussed in this section.

   Related references
   “SET DEFAULT SCROLL (full-screen mode)” on page 184

SELECT command (PL/I)

The SELECT command chooses one of a set of alternate commands.

If the reference can be satisfied by more than one of the WHEN clauses, only the first one is performed. If there is no reference, the first WHEN clause containing an expression that is true is executed. If none of the WHEN clauses are satisfied, the command specified on the OTHERWISE clause, if present, is performed. If the OTHERWISE clause should be executed and it is not present, a Debug Tool message is issued.
SELECT

\[\text{reference}\];

WHEN

\[\text{expression}\] \text{command}

OTHERWISE \text{command} \text{END};

\text{reference}
A valid Debug Tool PL/I scalar reference. An aggregate (array or structure) cannot be used as a reference.

\text{WHEN}
Specifies that an expression or a group of expressions be evaluated and either compared with the reference immediately following the SELECT keyword, or evaluated to true or false (if \text{reference} is omitted).

\text{expression}
A valid Debug Tool PL/I expression.

\text{command}
A valid Debug Tool command.

\text{OTHERWISE}
Specifies the command to be executed when every test of the preceding \text{WHEN} statements fails.

\text{Usage notes}
- You cannot use the SELECT command while you replay recorded statements by using the PLAYBACK commands.

\text{Example}

When sum is equal to the value of c+ev, display a message. When sum is equal to either fv or 0, display a message. If sum is not equal to the value of either c+ev, fv, or 0, a Debug Tool error message is issued.

```plaintext
SELECT (sum);
WHEN (c + ev) LIST ('Match on when group number 1');
WHEN (fv, 0) LIST ('Match on when group number 2');
END;
```

\textbf{SET command}

The SET command sets various switches that affect the operation of Debug Tool. Except where otherwise specified, settings remain in effect for the entire debug session.

The following table summarizes the forms of the SET command.

<table>
<thead>
<tr>
<th>SET command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SET ASSEMBLER&quot;</td>
<td>Controls the enablement of assembler debugging.</td>
</tr>
<tr>
<td>&quot;SET AUTOMONITOR&quot;</td>
<td>Controls the addition of data items to the Monitor window.</td>
</tr>
<tr>
<td>&quot;SET CHANGE&quot;</td>
<td>Controls the frequency of checking the AT CHANGE breakpoints.</td>
</tr>
<tr>
<td>&quot;SET COLOR (full-screen and line mode)&quot;</td>
<td>Provides control of the color, highlighting, and intensity attributes.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&quot;SET COUNTRY&quot; on page 182</td>
<td>Changes the current national country setting.</td>
</tr>
<tr>
<td>&quot;SET DBCS&quot; on page 182</td>
<td>Controls whether DBCS shift-in and shift-out codes are recognized.</td>
</tr>
<tr>
<td>&quot;SET DEFAULT LISTINGS&quot; on page 183</td>
<td>Defines a default partitioned data set (PDS) ddname or dname searched for program source listings or source files.</td>
</tr>
<tr>
<td>&quot;SET DEFAULT SCROLL (full-screen mode)&quot; on page 184</td>
<td>Sets the default scroll amount.</td>
</tr>
<tr>
<td>&quot;SET DEFAULT VIEW&quot; on page 189</td>
<td>Controls the default view for assembler compile units.</td>
</tr>
<tr>
<td>&quot;SET DEFAULT WINDOW (full-screen mode)&quot; on page 185</td>
<td>Sets the window that is affected by a window referencing command.</td>
</tr>
<tr>
<td>&quot;SET DISASSEMBLY&quot; on page 186</td>
<td>Controls whether the disassembly view is displayed in the Source window.</td>
</tr>
<tr>
<td>&quot;SET DYNDEBUG&quot; on page 186</td>
<td>Controls whether the Dynamic Debug facility is activated.</td>
</tr>
<tr>
<td>&quot;SET ECHO&quot; on page 188</td>
<td>Controls whether GO and STEP commands are recorded in the log window.</td>
</tr>
<tr>
<td>&quot;SET EQUATE&quot; on page 188</td>
<td>Equates a symbol to a string of characters.</td>
</tr>
<tr>
<td>&quot;SET EXECUTE&quot; on page 189</td>
<td>Controls whether commands are performed or just syntax checked.</td>
</tr>
<tr>
<td>&quot;SET FREQUENCY&quot; on page 190</td>
<td>Controls whether statement executions are counted.</td>
</tr>
<tr>
<td>&quot;SET HISTORY&quot; on page 191</td>
<td>Specifies whether entries to Debug Tool are recorded in the history table.</td>
</tr>
<tr>
<td>&quot;SET INTERCEPT (C, C++, and COBOL)&quot; on page 191</td>
<td>Intercepts input to and output from specified files, displaying prompts and output in the log.</td>
</tr>
<tr>
<td>&quot;SET KEYS (full-screen mode)&quot; on page 192</td>
<td>Controls whether PF key definitions are displayed.</td>
</tr>
<tr>
<td>&quot;SET LDD&quot; on page 193</td>
<td>Controls how debug data is loaded for assemblies containing multiple CSECTs.</td>
</tr>
<tr>
<td>&quot;SET LIST TABULAR&quot; on page 193</td>
<td>Controls the formatting of the output of the LIST command.</td>
</tr>
<tr>
<td>&quot;SET LOG&quot; on page 194</td>
<td>Controls the logging of output and assignment to the log file.</td>
</tr>
<tr>
<td>&quot;SET LOG NUMBERS (full-screen mode)&quot; on page 195</td>
<td>Controls whether line numbers are shown in the log window.</td>
</tr>
<tr>
<td>&quot;SET LONGCUNAME&quot; on page 195</td>
<td>Controls whether a long or short CU name is shown.</td>
</tr>
<tr>
<td>&quot;SET MONITOR&quot; on page 196</td>
<td>Controls the format and layout of variable names and values displayed in the Monitor window.</td>
</tr>
<tr>
<td>&quot;SET MSGID&quot; on page 198</td>
<td>Controls whether message identifiers are shown.</td>
</tr>
<tr>
<td>&quot;SET NATIONAL LANGUAGE&quot; on page 198</td>
<td>Switches your application to a different run-time national language.</td>
</tr>
<tr>
<td>&quot;SET PACE&quot; on page 199</td>
<td>Specifies the maximum pace of animated execution.</td>
</tr>
<tr>
<td>&quot;SET PFKEY&quot; on page 200</td>
<td>Associates a Debug Tool command with a PF key.</td>
</tr>
<tr>
<td>&quot;SET PROGRAMMING LANGUAGE&quot; on page 201</td>
<td>Sets the current programming language.</td>
</tr>
<tr>
<td>&quot;SET PROMPT (full-screen mode)&quot; on page 202</td>
<td>Controls the display of the current program location.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&quot;SET QUALIFY&quot; on page 203</td>
<td>Simplifies the identification of references and statement numbers by resetting the point of view.</td>
</tr>
<tr>
<td>&quot;SET REFRESH (full-screen mode)&quot; on page 205</td>
<td>Controls screen refreshing when the SCREEN setting is ON.</td>
</tr>
<tr>
<td>&quot;SET RESTORE&quot; on page 205</td>
<td>Controls the automatic and manual restoring of settings, breakpoints, and monitor specifications.</td>
</tr>
<tr>
<td>&quot;SET REWRITE&quot; on page 207</td>
<td>Forces a periodic screen rewrite.</td>
</tr>
<tr>
<td>&quot;SET SAVE&quot; on page 207</td>
<td>Controls the automatic saving of settings, breakpoints, and monitor specifications.</td>
</tr>
<tr>
<td>&quot;SET SCREEN (full-screen and line mode)&quot; on page 210</td>
<td>Controls how information is displayed on the screen.</td>
</tr>
<tr>
<td>&quot;SET SCROLL DISPLAY (full-screen mode)&quot; on page 211</td>
<td>Controls whether the scroll field is displayed.</td>
</tr>
<tr>
<td>&quot;SET SEQUENCE (PL/I)&quot; on page 211</td>
<td>Controls whether Debug Tool interprets data after column 72 as a sequence number.</td>
</tr>
<tr>
<td>&quot;SET SOURCE&quot; on page 211</td>
<td>Associates a source listing or source file with one or more compile units.</td>
</tr>
<tr>
<td>&quot;SET SUFFIX (full-screen mode)&quot; on page 214</td>
<td>Controls the display of the Source window suffix area.</td>
</tr>
<tr>
<td>&quot;SET TEST&quot; on page 214</td>
<td>Overrides the initial TEST run-time options specified at invocation.</td>
</tr>
<tr>
<td>&quot;SET WARNING (C, C++, COBOL, and PL/I)&quot; on page 215</td>
<td>Controls display of the Debug Tool warning messages and whether exceptions are reflected to the application program.</td>
</tr>
</tbody>
</table>

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

- "SET command (COBOL)" on page 216

**SET ASSEMBLER**

A disassembled compilation unit is a CU that was not compiled with the TEST compiler option and has not been used as the operand of a LOADDEBUGDATA command. The SET ASSEMBLER ON command enables a subset of the functions enabled by the SET DISASSEMBLY ON command. The following behavior is enabled for disassembled compilation units by the SET ASSEMBLER ON command:

- You can stop in a disassembly CU by using the commands:
  - AT APPEARANCE *
  - AT APPEARANCE name
  - AT ENTRY name
- You can display the names of disassembled CUs by using the following commands:
  - DESCRIBE CUS
  - LIST
  - LIST NAMES CUS
  - QUERY SOURCE
Notes:
1 Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

OFF
Disables the display of data that is useful while you debug an assembler program.

ON Enables the display of data that is useful while you debug an assembler program.

Usage notes
• You can also use the SET DISASSEMBLY ON to control the display of information that is useful while you debug an assembler program.
• You can use this command in remote debug mode when you use one of the following remote debuggers:
  – Compiled Language Debugger component of WebSphere Studio Enterprise Developer
  – Compiled Language Debugger component of WebSphere Developer for zSeries
  – WebSphere Developer Debugger for zSeries

Example
To include disassembly compile units in the list of compile units displayed by the LIST NAMES CUS and DESCRIBE CUS commands, enter the following command:
SET ASSEMBLER ON ;

The next time you enter the LIST NAMES CUS or DESCRIBE CUS command, the disassembly compile units are displayed in the list of compile units.

Refer to the following sections for more information related to the material discussed in this section.

Related references
"SET DISASSEMBLY" on page 186

SET AUTOMONITOR
Controls the monitoring of data items at the currently executing statement. The initial setting is OFF.

AUTOMONITOR works only for the following compile units:
• compiled with COBOL or PL/I compilers with the SYM suboption of the TEST compiler option
• assembler, disassembly, or OS/VS COBOL compile units

The SET AUTOMONITOR command does not work for compile units written in any other language. In addition, the compile unit must be compiled or assembled with one of the following compilers or assemblers:
• Enterprise COBOL for z/OS and OS/390, Version 3 Release 2
• Enterprise COBOL for z/OS and OS/390, Version 3 Release 1, with APAR PQ63235 installed
• COBOL for OS/390 & VM, Version 2, with APAR PQ63234 installed
• OS/VS COBOL, Version 1 Release 2.4
• Enterprise PL/I for z/OS and OS/390, Version 3 Release 2
• High Level Assembler for MVS & VM & VSE, Version 1 Release 4

You must also have the following APARs installed on your system:
• For COBOL:
  – z/OS, Version 1 Release 5, with APAR PQ92282
  – z/OS, Version 1 Release 4, with APAR PQ65176 and PQ92282

Notes:
1  Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

ON  Enables monitoring of data items on currently executing line. Specify the LOG suboption to save information in the log file.

OFF  Disables monitoring of all data items. Information is not saved in the log file.

LOG    Saves information in the log file.

NOLOG  Does not save information in the log file.

Usage notes
• You can use this command in remote debug mode when you use one of the following remote debuggers:
  – Compiled Language Debugger component of WebSphere Studio Enterprise Developer
  – Compiled Language Debugger component of WebSphere Developer for zSeries
  – WebSphere Developer Debugger for zSeries
• If the DATA option of the PLAYBACK ENABLE command is in effect for the current compile unit, you can use the SET AUTOMONITOR command while you replay recorded statements by using the PLAYBACK commands.
• If you enter the SET AUTOMONITOR ON LOG command for a compile unit that was compiled with a compiler that does not support aut-monitoring, then Debug Tool writes the breakpoint location into the log. This provides a record of the breakpoints encountered (breakpoint trace). No variable information is displayed.
• To record the breakpoints encountered (breakpoint trace) in the log file, enter the following commands: SET AUTOMONITOR ON LOG; AT * GO; For compile units compiled with a compiler that supports aut-monitoring, the statement location, the variable names, and the value of the variables are saved into the log. For other compile units, the statement location is saved into the log.
• If you are debugging programs compiled with a PL/I compiler earlier than Enterprise PL/I for z/OS Version 3 Release 5, target variables are not listed. For example, in the following PL/I statement only J and its value is displayed:
  \[ I = J + 1 \]
• For assembler and disassembly, only 32-bit general registers, floating-point registers, and storage operands are displayed. Register operands are displayed in numeric order. Storage operands are displayed in the order S1, S2, and S4 and are displayed using the \texttt{\_STORAGE} function (for example, \texttt{\_STORAGE(X'1F3C8':4)} is used to display a four-byte storage operand at address \texttt{X'1F3C8')}.
• For OS/VS COBOL, array references are not included in the AUTOMONITOR output.
• To disable monitoring of all data items, you can enter the \texttt{SET AUTOMONITOR OFF} or \texttt{CLEAR MONITOR n} commands, where \( n \) is the monitor number of an automonitor entry. You can also use \texttt{CL} prefix command on an entry in Monitor window.

**SET CHANGE**

Controls the frequency of checking the AT CHANGE breakpoints. The initial setting is \texttt{STATEMENT/LINE}.

\[
\begin{array}{c}
\text{SET CHANGE} \quad \text{STATEMENT} \\
\text{ ALL} \\
\text{ BLOCK} \\
\text{ LINE} \\
\text{ PATH} \\
\end{array}
\]

**STATEMENT**

Specifies that the AT CHANGE breakpoints are checked at all statements. \texttt{STATEMENT} is equivalent to \texttt{LINE}.

**ALL**

Specifies that the AT CHANGE breakpoints are checked at all statements, block entry and exits, and path points.

**BLOCK**

Specifies that the AT CHANGE breakpoints are checked at all block entry and exits.

**LINE**

Is equivalent to \texttt{STATEMENT}.

**PATH**

Specifies that the AT CHANGE breakpoints are checked at all path points.

**Examples**

• Specify that AT CHANGE breakpoints are checked at all statements.
  \[
  \text{SET CHANGE;} \\
  \]

• Specify that AT CHANGE breakpoints are checked at all path points.
  \[
  \text{SET CHANGE PATH;} \\
  \]
**SET COLOR** (full-screen and line mode)

Provides control of the color, highlighting, and intensity attributes when the SCREEN setting is ON. The color, highlighting, and intensity keywords can be specified in any order.

```
>> SET COLOR color_attributes UI_elements ::
```

**color_attributes:**

- **CYCLE**
  - Causes the color to change to the next one in the sequence of colors. The sequence follows the order shown in the syntax diagram.
  - **BLUE**
  - **GREEN**
  - **PINK**
  - **RED**
  - **TURQUOISE**
  - **WHITE**
  - **YELLOW**

- **BLINK**
  - Causes the characters to blink (if supported by the terminal).

- **NONE**
  - Causes the characters to appear in normal type.

- **REVERSE**
  - Transforms the characters to reverse video (if supported by the terminal).

- **UNDERLINE**
  - Causes the characters to be underlined (if supported by the terminal).

**UI_elements:**

- **CURSOR**
  - **COMMAND-LINE**
  - **LOG-LINES**
  - **MONITOR-AREA**
  - **MONITOR-LINES**
  - **PROGRAM-OUTPUT**
  - **SOURCE-AREA**
  - **BREAKPOINTS**
  - **CURRENT**
  - **PREFIX**
  - **SUFFIX**
  - **TARGET**
  - **FIELD**
  - **INPUT**
  - **OUTPUT**
  - **TITLE**
  - **FIELDS**
  - **HEADERS**
  - **TOFEOF**
  - **MARKER**
  - **WINDOW**
  - **HEADERS**

---

Debug Tool V7.1 Reference and Messages
HIGH
Causes screen colors to be high intensity (if supported by the terminal).

LOW
Causes screen colors to be low intensity (if supported by the terminal).

CURSOR
Specifies that cursor pointing is used to select the field. Optionally, you can
type in the field name (for example, COMMAND LINE) as shown in the syntax
diagram.

COMMAND LINE
Selects the command input line (preceded by ===>).

LOG LINES
Selects the line number portion of the log window.

MONITOR AREA
Selects the primary area of the monitor window.

MONITOR LINES
Selects the line number portion of the monitor window.

PROGRAM OUTPUT
Selects the application program output displayed in the log window.

SOURCE AREA
Selects the primary area of the Source window.

SOURCE BREAKPOINTS
Selects the source prefix fields next to statements where breakpoints are set.

SOURCE CURRENT
Selects the line containing the source statement that is about to be performed.

SOURCE PREFIX
Selects the statement identifier column at the left of the source window.

SOURCE SUFFIX
Selects the frequency column at the right of the Source window.

TARGET FIELD
Selects the target of a FIND command in full-screen mode, if found.

TEST INPUT
Selects the Debug Tool input displayed in the log window.

TEST OUTPUT
Selects the Debug Tool output displayed in the log window.

TITLE FIELDS
Selects the information fields in the top line of the screen, such as current
programming language setting or the current location within the program.

TITLE HEADERS
Selects the descriptive headers in the top line of the screen, such as location.

TOFEOF MARKER
Selects the top-of-file and end-of-file lines in the session panel windows.

WINDOW HEADERS
Selects the header lines for the windows in the main session panel.

Examples
• Set the Source window display area to yellow reverse video.
SET COLOR YELLOW REVERSE SOURCE AREA;

- Set the monitor window display area to high intensity green.

SET COLOR HIGH GREEN MONITOR AREA;

**SET COUNTRY**

Changes the current national country setting for the application program. It is available only where supported by Language Environment or when running without the Language Environment run time. The IBM-supplied initial country code is US.

```plaintext
SET COUNTRY country_code;
```

- `country_code`:
  A valid two-letter set that identifies the country code used. The country code can have one of the following values:
  - United States: US
  - Japanese: JP

Country codes cannot be truncated.

**Usage notes**

- This setting affects both your application and Debug Tool.
- At the beginning of an enclave, the settings are those provided by Language Environment, your operating system, or the Debug Tool run-time options. For nested enclaves, the parent’s settings are restored upon return from a child enclave.

**Example**

Change the current country code to correspond to Japan.

```plaintext
SET COUNTRY JP;
```

**SET DBCS**

Controls whether shift-in and shift-out codes are interpreted on input and supplied on DBCS output. SET DBCS is valid for all programming languages. The initial setting is OFF.

```plaintext
SET DBCS [ON|OFF];
```

- **ON**:
  Interprets shift-in and shift-out codes. If you debugging in full-screen mode and your terminal is not capable of displaying DBCS characters, this option is not available.

- **OFF**:
  Ignores shift-in and shift-out codes.

**Usage notes**

- If you enter the commands SET NATIONAL LANGUAGE ENU and then SET DBCS ON, Debug Tool resets the national language to UEN to remain compatible with DBCS characters.

**Example**
Specify that shift-in and shift-out codes are interpreted.

```
SET DBCS ON;
```

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

“SET NATIONAL LANGUAGE” on page 198

**SET DEFAULT LISTINGS**

Defines a default partitioned data set DD name or DS name whose members are searched for program source, listings, or separate debug files.

```
SET DEFAULT LISTINGS ddname
    dsn
    (dsn, dsn, ...)
```

**ddname**

Specifies a valid z/OS DD name. If the operand is less than nine characters long and does not contain a period, it is interpreted as a DD name.

The **ddname** form cannot be used if the data set allocated to it is C, C++ or Enterprise PL/I source and the EQAOPTS SUBSYS=ssss option is being used to access a source file in a library system.

**dsn**

Specifies a valid z/OS partitioned data set name.

**(dsn, dsn, ...)**

Specifies a list of valid z/OS partitioned data set names.

**Usage notes**

- The LISTINGS keyword cannot be abbreviated.
- If you do not specify a **ddname** or **dsn**, any previous default listing setting is cleared.
- If the data set name is too long to be typed on one line, suffix it with a trailing hyphen.
- The SET SOURCE ON command has a higher precedence than the SET DEFAULT LISTINGS command.
- The SET DEFAULT LISTINGS command has no effect on a disassembly compile unit. However, it is saved and it might apply later if the compile unit is specified as the operand of the LOADDEBGDATA command.
- If you are debugging in a CICS environment, you can not use the **ddname** parameter.
- If you compiled your C or C++ program with the FORMAT(DWARF) suboption of the DEBUG compiler option, you cannot use the SET DEFAULT LISTINGS command to specify the new location of the file generated by the FILE suboption of the DEBUG compiler option. Use the EQADBUG statement to specify the new location of the file.

**Examples**

- Indicate that the default listings file is allocated to DS name SVTRSAMP.TS99992.MYP org.
SET DEFAULT LISTINGS SVTRSAMP.TS99992.MYPROG;

- The listing for the program MYPROG is in SVTRSAMP.TS99992.MYPROG, which was allocated by using the following command:
  ALLOC DDNAME(ITEM1) DSNAME('SVTRSAMP.TS99992.MYPROG') SHR

To specify the location, enter the following command:
SET DEFAULT LISTINGS ITEM1;

- The listing for the program MYPROG is in JSMITH.COBPGMS.LISTING, which was allocated by using the following command:
  ALLOC FI(CBLIST) DAT('MJONES.OTHER.LISTING' 'JSMITH.COBPGMS.LISTING')

To specify the location, enter the following command:
SET DEFAULT LISTINGS CBLIST

- The listing for the program AVER is in myid.source.listing(AVERLIST). If you enter the command SET DEFAULT LISTINGS myid.source.listing, Debug Tool looks for a member named AVER in the PDS myid.source.listing. Because the member is called AVERLIST, the listing is not found. To specify the location, enter the following command:
  SET SOURCE ON (AVER) myid.source.listing(AVERLIST);

Refer to the following sections for more information related to the material discussed in this section.

**Related tasks**

- [Debug Tool User’s Guide](#)
- “SET SOURCE” on page 211

**SET DEFAULT SCROLL (full-screen mode)**

Sets the default scroll amount that is used when a SCROLL command is issued without the amount specified. The initial setting is PAGE.

```
SET DEFAULT SCROLL
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSR</td>
<td>Scrolls in the specified direction until the character where the cursor is positioned reaches the edge of the window.</td>
</tr>
<tr>
<td>DATA</td>
<td>Scrolls by one line less than the window size or by one character less than the window size (if moving left or right).</td>
</tr>
<tr>
<td>HALF</td>
<td>Scrolls by half the window size.</td>
</tr>
<tr>
<td>integer</td>
<td>Scrolls the specified number of lines (up or down) or the specified number of characters (left or right). Maximum value is 9999.</td>
</tr>
<tr>
<td>MAX</td>
<td>Scrolls in the specified direction until the limit of the data is reached.</td>
</tr>
</tbody>
</table>
PAGE
   Scrolls by the window size.

Example
   Set the default amount to half the size of the window.
   SET DEFAULT SCROLL HALF;

SET DEFAULT VIEW
   Controls the default view for assembler compile units.

   SET DEFAULT VIEW STANDARD; NOMACGEN

STANDARD
   Indicates that whenever a LOADDEBUGDATA (LDD) command is issued for an assembler CU, the initial view is to contain all source statements.

NOMACGEN
   Indicates that whenever a LOADDEBUGDATA (LDD) command is issued for an assembler CU, the initial view is to contain only source statements that were not generated via macro expansion (similar to the assembler listing when PRINT NOGEN is in effect).

Usage notes
   • SET DEFAULT VIEW applies only to assembler compile units.
   • You can use this command in remote debug mode when you use one of the following remote debuggers:
     – Compiled Language Debugger component of WebSphere Studio Enterprise Developer
     – Compiled Language Debugger component of WebSphere Developer for zSeries
       – WebSphere Developer Debugger for zSeries

SET DEFAULT WINDOW (full-screen mode)
   Specifies what window is selected when a window referencing command (for example, FIND, SCROLL, or WINDOW) is issued without explicit window identification and the cursor is outside the window areas. The initial setting is SOURCE.

   SET DEFAULT WINDOW LOG; MONITOR SOURCE

LOG
   Selects the session log window.

MONITOR
   Selects the monitor window.

SOURCE
   Selects the source listing window.

Example
Set the default to the monitor window for use with scrolling commands.

```
SET DEFAULT WINDOW MONITOR;
```

**SET DISASSEMBLY**

A disassembled compilation unit is a CU that was not compiled with the TEST compiler option and has not been used as the operand of a LOADDEBUGDATA command. The SET DISASSEMBLY ON command enables the following behavior for disassembled compilation units:

- A disassembly view appears in the source window whenever you qualify a disassembled compilation unit. You can set breakpoints in the CU using the AT OFFSET command and you can step within the CU using the STEP command.
- You can stop in a disassembly CU by using the following commands:
  - AT APPEARANCE *
  - AT APPEARANCE name
  - AT ENTRY *
  - AT ENTRY name
  - STEP INTO

- You can display the names of disassembled CUs by using the following commands:
  - DESCRIBE CUS
  - LIST
  - LIST NAMES CUS
  - QUERY SOURCE

```
ON  Specifies that the disassembly view is displayed in the Source window.
OFF Turns off the disassembly view. This is the default setting.
```

**Usage notes**

- The disassembly view is provided only for disassembled programs or programs written in supported languages that do not have debug information.
- You can use this command in remote debug mode.

**SET DYNDEBUG**

Controls the activation or deactivation of the Dynamic Debug facility.

The Dynamic Debug facility must be installed and activated in order to debug the following types of programs:

- COBOL programs compiled with the NONE suboption of the TEST compiler option
- Enterprise PL/I for z/OS Version 3 Release 4 programs compiled with the NOHOOK suboption of the TEST compiler option
- assembler programs
• disassembled programs (using the disassembly view)
• OS/VS COBOL programs
• programs that run without the Language Environment run time

You can use the Dynamic Debug facility to improve the performance of programs with compiled-in hooks (compiled with COBOL, C/C++, and PL/I compilers) while you debug them.

If the Dynamic Debug facility has been installed, the initial setting is ON. If it was not installed, the initial setting is OFF and you cannot activate the Dynamic Debug facility.

ON  Activates the Dynamic Debug facility.
OFF  Deactivates the Dynamic Debug facility.

Usage notes
• After a dynamic debug hook has been inserted, either explicitly or implicitly, into any program during a debugging session, you cannot use the SET DYNDEBUG OFF command.
• You can use this command in remote debug mode.
• To debug COBOL programs compiled with the TEST(NONE) compiler option and use the Dynamic Debug facility, you must compile with one of the following compilers:
  – Enterprise COBOL for z/OS and OS/390, Version 3
  – COBOL for OS/390 & VM, Version 2 Release 2
  – COBOL for OS/390 & VM, Version 2 Release 1, with APAR PQ40298
• You can use the GOTO command if you compile your program with one of the following compilers and specify the TEST(NONE,SYM) compiler option:
  – Enterprise COBOL for z/OS and OS/390, Version 3 Release 2
  – Enterprise COBOL for z/OS and OS/390, Version 3 Release 1, with APAR PQ63235 installed
  – COBOL for OS/390 & VM, Version 2, with APAR PQ63234 installed
• The Dynamic Debug facility does not support attention interrupts with programs compiled using TEST(NONE,SYM).
• When the following compilers are used with the suboption of the TEST compiler option that adds compiled-in hooks, the Dynamic Debug facility can be used to add hooks at runtime, which Debug Tool uses instead of the compiled-in hooks. This can improve the performance of the program while running under the control of Debug Tool.
  – Any COBOL compiler supported by Debug Tool
  – Any C/C++ compiler supported by Debug Tool
  – Any PL/I compiler supported by Debug Tool

1. In non-CICS environments, SVC screening must be enabled to debug OS/VS COBOL programs, programs that run without the Language Environment runtime, or programs that are loaded by using the MVS LOAD and LINK macros. See Debug Tool Customization Guide for instructions on how to manage SVC screening.
• Refer to your system administrator to determine if the Dynamic Debug facility is installed on your system.

• The same program compiled with different TEST options may halt execution at different locations or the same scenarios. For instance, if you compile a program with TEST(ALL,...) and step through the first three lines, execution is halted on line four. However, if you compile the same program with TEST(NONE,SYM,...) and step through the first three lines, execution is halted on line five. The difference is due to optimization techniques used by the compiler.

A small arrowhead indicates where a Debug Tool would stop if the same program were compiled in two different ways.

<table>
<thead>
<tr>
<th>Program compiled with TEST(ALL)</th>
<th>Program compiled with TEST(NONE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>000001 MOVE…</td>
<td>000001 MOVE…</td>
</tr>
<tr>
<td>000002 ADD…</td>
<td>000002 ADD…</td>
</tr>
<tr>
<td>➖000003 LABEL: ...</td>
<td>000003 LABEL: ...</td>
</tr>
<tr>
<td>000004 MOVE…</td>
<td>➖000004 MOVE…</td>
</tr>
</tbody>
</table>

**SET ECHO**

Controls whether GO and STEP commands are recorded in the log window when they are not subcommands. The presence of long sequences of GO and STEP commands clutters the log window and provides little additional information. SET ECHO makes it possible to suppress the display of these commands. The contents of the log file are unaffected. The initial setting is ON.

```plaintext
SET ECHO [ON|OFF] [keyword];
```

**ON**  Shows given commands in the log window.

**OFF**  Suppresses given commands in the log window.

**keyword**  Can be GO (with no operand) or STEP.

*  Specifies that the command is applied to the GO and STEP commands. This is the default.

**Examples**

• Specify that the display of GO and STEP commands is suppressed.

  ```plaintext
  SET ECHO OFF;
  ```

• Specify that GO and STEP commands are displayed.

  ```plaintext
  SET ECHO ON *;
  ```

**SET EQUATE**

Equates a symbol to a string of characters. The equated symbol can be used anywhere a keyword, identifier, or punctuation is used in a Debug Tool command. When an equated symbol is found in a Debug Tool command (other than the *identifier operand in SET EQUATE and CLEAR EQUATE), the equated symbol is replaced by the specified string before parsing continues.
---SET-EQUATE-identifier—string---;

**identifier**
An identifier that is valid in the current programming language. The maximum length of the identifier is:
- For C, 32 SBCS characters
- For COBOL and OS/VS COBOL, 30 SBCS characters
- For PL/I, 31 SBCS characters

The identifier can contain DBCS characters.

**string**
A string constant in the current programming language. The maximum length of the replacement string is 255 SBCS characters.

**Usage notes**
- Operands of the following commands are for environments other than the standard Debug Tool environment (that is, TSO DS name, and so forth) and are not scanned for EQUATEd symbol substitution:
  
  ```
  COMMENT
  INPUT
  SET DEFAULT LISTINGS
  SET INTERCEPT ON/OFF FILE
  SET LOG ON FILE
  SET SOURCE (cu_spec)
  SYSTEM/SYS
  TSO
  USE
  ```

- To remove an EQUATE definition, use the CLEAR EQUATE command.
- To remain accessible when the current programming language setting is changed, symbols that are equated when the current programming language setting is C must be entered in uppercase and must be valid in the other programming languages.
- If an EQUATE identifier coincides with an existing keyword or keyword abbreviation, EQUATE takes precedence. If the EQUATE identifier is already defined, the new definition replaces the old.
- The equate string is not scanned for, or substituted with, symbols previously set with a SET EQUATE command.

**Examples**
- Specify that the symbol INFO is equated to "ABC, DEF (H+1)". The current programming language setting is either C or COBOL.
  ```
  SET EQUATE INFO = "ABC, DEF (H+1)";
  ```
- Specify that the symbol tslen is equated to the equivalent of a define for structure pointing. The current programming language setting is C. If the programming language changes, this lowercase symbol might not be accessible.
  ```
  SET EQUATE tslen = "struct1->member.b->c.len";
  ```
- Specify that the symbol VARVALUE is equated to the command LIST x.
  ```
  SET EQUATE VARVALUE = "LIST x";
  ```

**SET EXECUTE**
Controls whether commands from all input sources are performed or just syntax checked (primarily for checking USE files). The initial setting is ON.
ON  Specifies that commands are accepted and performed.

OFF  Specifies that commands are accepted and parsed; however, only the following
      commands are performed: END, GO, SET EXECUTE ON, QUIT, and USE.

Example

Specify that all commands are accepted and performed.
SET EXECUTE ON;

SET FREQUENCY

Controls whether statement executions are counted. The initial setting is OFF.

ON  Specifies that statement executions are counted.

OFF  Specifies that statement executions are not counted.

cu_spec  A valid compile unit specification. If omitted, all compile units with statement
          information are processed.

Usage notes

• In the disassembly view, SET FREQUENCY is not supported.
• Because the collection of frequency data can add a substantial amount of
  overhead, set the SET FREQUENCY command to ON only when you intend to make
  use of this data. Do not routinely set the SET FREQUENCY command to ON in
  debug sessions in which you do not intend to make use of this data.
• If the DATA option of the PLAYBACK ENABLE command is in effect for the current
  compile unit, you can use the SET FREQUENCY command while you replay
  recorded statements by using the PLAYBACK commands.

Example

Specify that statement executions are counted in compile units main and subr1.
SET FREQUENCY ON (main, subr1);

Refer to the following sections for more information related to the material
discussed in this section.

Related references
  "cu_spec" on page 13
  "LIST FREQUENCY" on page 132
  "SET SUFFIX (full-screen mode)" on page 214
SET HISTORY

Specifies whether entries to Debug Tool are recorded in the history table and optionally adjusts the size of the table. The history table contains information about the most recently processed breakpoints and conditions. The initial setting is ON; the initial size is 100.

```plaintext
SET HISTORY [ON OFF] integer ;
```

**ON** Maintains the history of invocations.

**OFF**

Suppresses the history of invocations.

*integer* The number of entries kept in the history table.

**Usage notes**

- History is not collected for disassembly compile units.

**Examples**

- Adjust the history table size to 50 lines.
  ```plaintext
  SET HISTORY 50;
  ```

- Turn off history recording.
  ```plaintext
  SET HISTORY OFF;
  ```

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

"LIST LAST" on page 132

SET INTERCEPT (C, C++, and COBOL)

Intercepts input to and output from specified files. Output and prompts for input are displayed in the log.

Only sequential I/O can be intercepted. I/O intercepts remain in effect for the entire debug session, unless you terminate them by selecting SET INTERCEPT OFF. The initial setting is OFF.

```plaintext
SET INTERCEPT [ON OFF] FILE file_spec ;
```

**ON** Turns on I/O interception for the specified file. Output appears in the log, preceded by the file specifier for identification. Input causes a prompt entry in the log, with the file specifier identified. You can then enter input for the specified file on the command line by using the INPUT command.

**OFF**

Turns off I/O interception for the specified file.

**FILE file_spec**

A valid file specification that is interpreted by each supported language. The FILE keyword cannot be abbreviated.
In C, this can be any valid fopen() file specifier including stdin, stdout, or stderr.

**CONSOLE (COBOL)**

Turns on I/O interception for the console.

This consists of:
- Job log output from DISPLAY UPON CONSOLE
- Screen output (and confirming input) from STOP 'literal'
- Terminal input for ACCEPT FROM CONSOLE or ACCEPT FROM SYSIN.

**Usage notes**
- COBOL supports only the CONSOLE command.
- For C, intercepted streams or files cannot be part of any C I/O redirection during the execution of a nested enclave.
- For PL/I, SET INTERCEPT is not supported.
- For CICS, SET INTERCEPT is not supported.
- You cannot use the SET INTERCEPT command while you replay recorded statements by using the PLAYBACK commands.

**Examples**
- Turn on the I/O interception for the console. The current programming language setting is COBOL.
  
  ```
  SET INTERCEPT CONSOLE;
  ```
- Turn on the I/O interception for the fopen() file specifier dd:mydd. The current programming language setting is C.
  
  ```
  SET INTERCEPT ON FILE dd:mydd;
  ```

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

- ["INPUT command (C, C++, and COBOL)" on page 121](#)
- ["SET REFRESH (full-screen mode)" on page 205](#)

**SET KEYS (full-screen mode)**

Controls whether PF key definitions are displayed when the SCREEN setting is ON.

The initial setting is ON.

```plaintext
SET KEYS OFF
```

**ON**

Displays PF key definitions.

**OFF**

Suppresses the display of the PF key definitions.

**12**

Shows PF1-PF12 on the screen bottom.

**24**

Shows PF13-PF24 on the screen bottom.

**Example**

Specify that the display of the PF key definitions is suppressed.

```plaintext
SET KEYS OFF;
```
Refer to the following sections for more information related to the material discussed in this section.

**Related references**

“SET PFKEY” on page 200

---

**SET LDD**

Controls how debug data is loaded for assemblies containing multiple CSECTs. The initial setting is SINGLE.

```
SET LDD SINGLE
```

**SINGLE**

Indicates that subsequent LOADDEBUGDATA (LDD) commands that load debug data for a CU that was assembled with other CSECTs are to load the debug data for the specified CU only.

**ALL**

Indicates that subsequent LOADDEBUGDATA (LDD) commands that load debug data for a CU that was assembled with other CSECTs are to load the debug data for all CUs in the assembly.

**Usage notes**

- This command affects both deferred and non-deferred LDD commands.
- If the target of the LDD is an OS/VS COBOL CU, the command has no effect.
- If SET LDD ALL is in effect and you do the following tasks, you must enter a separate SET SOURCE command for each CU in the assembly for which you previously entered an LDD command:
  - You enter an LDD command for more than one CU in the same assembly.
  - The debug data could not be found for these CUs.
  - Subsequently, you enter a SET SOURCE command for one of these CUs.

**Examples**

- Load debug data for all CSECTs in an assembly that contains CSECTs CS1, CS2, and CS3:
  ```
  SET LDD ALL;
  LDD CS1;
  ```
- Load debug data for CSECT’s CS1 and CS3 in an assembly that contains CSECTs CS1, CS2, and CS3:
  ```
  SET LDD SINGLE;
  LDD (CS1,CS3);
  ```

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

“LOADDEBUGDATA” on page 141

**Related tasks**

Debug Tool User’s Guide

---

**SET LIST TABULAR**

Controls whether to format the output of the LIST command in a tabular format. The default setting is OFF.
**SET LIST**

Controls whether the output of the LIST command is displayed in tabular or linear format. The initial setting is **ON**.

**ON**

Display the output of the LIST command in tabular format.

**OFF**

Display the output of the LIST command in linear format. This is the default setting.

**Notes:**

1. Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

**SET LOG**

Controls whether each performed command and the resulting output is written to the log file and defines (or redefines) the file that is used. The initial setting is **ON**.

**ON**

Specifies that commands and output are written to the log file.

**FILE fileid**

Identifies the log file used. The FILE keyword cannot be abbreviated.

In non-CICS, fileid is a DD name or a fully-qualified data set name. Partitioned data sets cannot be used.

In CICS, fileid is a fully-qualified data set name.

If fileid has the form of a DD name, Debug Tool checks to see if the file is allocated.

In full-screen mode, the log file should not be allocated to the 3270 terminal device.

**KEEP count**

Specifies the number of lines of log output retained for display. The initial setting is 1000; count cannot equal zero (0).

**OFF**

Specifies that commands and output are not written to a log file.

**Usage notes:**

- You can use the SET LOG ON and SET LOG OFF commands in remote debug mode.
- The log output lines retained for display are always the last (that is, the most recent) lines.
- Setting LOG OFF does not suppress the log display.
- In non-CICS, if the same file name already exists, the output log is appended to the existing file. In CICS, if the same file name already exists, the existing file is replaced by the output log.
- If you are debugging in full-screen mode and the log file is allocated to the terminal, issue a SET LOG OFF command before issuing a QUIT command. If you do not issue the SET LOG OFF command, the QUIT command fails.

**Notes:**

You can use the SET LOG ON and SET LOG OFF commands in remote debug mode.
In CICS, the log file (INSPLOG) is not automatically started. You need to use the SET LOG ON command.

Ensure that you allocate a log file big enough to hold all the log output from a debug session, because the log file is truncated after it becomes full. (A warning message is not issued before the log is truncated.)

**Examples**

- Specify that commands and output are written to the log file named mainprog.
  
  ```
  SET LOG ON FILE mainprog;
  ```

  Another example using the data set name thing.
  
  ```
  SET LOG ON FILE userid.thing.log
  ```

- Indicate that 500 lines of log output are retained for display.
  
  ```
  SET LOG KEEP 500;
  ```

Refer to the following sections for more information related to the material discussed in this section.

**Related tasks**

[Debug Tool User’s Guide](#)

**SET LOG NUMBERS (full-screen mode)**

Controls whether line numbers are shown in the log window. The initial setting is ON.

```plaintext
>>-SET-LOG-NUMBERS [ON] [OFF]:
```

- **ON** Shows line numbers in the log window.
- **OFF** Suppresses line numbers in the log window.

**Example**

Specify that log line numbers are not shown.

```plaintext
SET LOG NUMBERS OFF;
```

**SET LONGCUNAME**

Controls whether a short or long CU name is displayed.

```plaintext
>>-SET-LONGCUNAME [ON] [OFF]:
```

- **ON** Specifies that a long CU name is displayed.
- **OFF** Specifies that a short CU name is displayed. The short CU name is displayed in the session panel header, source window header area, and the Source Identification Panel.

**Usage notes**
• You can enter the SET LONGCUNAME at any time, but it applies only to C, C++, and Enterprise PL/I programs. If you compiled your program with the following compiler and it is running in the following environment, this command has no effect.
  – Enterprise PL/I for z/OS, Version 3.5, compiler with the PTFs for APARs PK35230 and PK35489 applied
  – Language Environment Version 1.4 through 1.8 with the PTF for APAR PK33738 applied

• The CU name for programs compiled with C, C++, or Enterprise PL/I (prior to Enterprise PL/I for z/OS, Version 3.5, with the PTFs for APARs PK35230 and PK35489 applied) compilers can have one of the following forms:
  – Fully qualified partitioned data set name and member name
  – A sequential file name
  – A HFS path and file name

These forms can result in long CU names that are truncated in the session panel header, which makes it difficult for you to identify the CU.

For these forms of compile unit names, Debug Tool displays short names in one of the following manners:
  – For PDS file names, the short name is only the member name
  – For sequential file names, the short name is the lowest level qualifier (name segment)
  – For HFS file names, the short name is the file name, without path name

If you compiled your program with the following compiler and it is running in the following environment, this command has no effect.
  – Enterprise PL/I for z/OS, Version 3.5, compiler with the PTFs for APARs PK35230 and PK35489 applied
  – Language Environment Version 1.4 through 1.8 with the PTF for APAR PK33738 applied

• Debug Tool commands affected by the LONGCUNAME setting: QUERY LOCATION, SET SOURCE, and AT ENTRY. All the other commands continue to require the long form of the CU name. For example, if you use the short name with the AT command (AT ARRAY3 ::> 'ARRAY3' ::> 10), Debug Tool displays an error message and does not set the breakpoint. However, if you enter the command AT ENTRY ARRAY3 ::> 'ARRAY3' ::>ARRAY3, Debug Tool sets the breakpoint or defers setting the breakpoint until the entry point is known to Debug Tool.

• You cannot use the SET LONGCUNAME command in remote debug mode.

Examples
• If the CU name is SMITH.TEST.SRC(ARRAY3), the short name is ARRAY3.
• If the CU name is SMITH.TEST.SOURCE.ABCD, the short name is ABCD.
• If the CU name is /testenvir/applications/cicsprograms/project1/prog2.cpp, the short name is prog2.cpp.

SET MONITOR
Controls the format and layout of variable names and values displayed in the Monitor window.
Notes:

1. Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

COLUMN
Controls whether to display the output in the Monitor window in column format. The initial setting is SET MONITOR COLUMN ON. SET MONITOR COLUMN is accepted in batch mode, but has no effect.

DATATYPE
Controls whether to display the data type of the variable in the Monitor window. The initial setting is SET MONITOR DATATYPE OFF.

NUMBERS (full-screen mode)
Controls whether to display line numbers in the Monitor window. The initial setting is SET MONITOR NUMBERS ON.

WRAP
Controls whether to wrap the output in the Monitor window. The initial setting is SET MONITOR WRAP ON. SET MONITOR WRAP is accepted in batch mode, but has no effect.

ON
Sets the corresponding switch to the following values:

COLUMN
Display the Monitor window output in column-aligned format.

DATATYPE
Display the data type attribute for variables in the Monitor window.

NUMBERS
Display line numbers in the Monitor window.

WRAP
Wraps the monitor value area variable in the monitor window.

OFF
Sets the corresponding switch to the following values:

COLUMN
Display the Monitor window output in traditional (non-column-aligned) format.

DATATYPE
Do not display the data type attribute for variables in the Monitor window.

NUMBERS
Do not display line numbers in the Monitor window.

WRAP
Display the variable name and value on the same line in the monitor window. If any values are too long to display in the Monitor window, then the area becomes scrollable.

Usage notes
If you enter the SET MONITOR WRAP OFF command while the SET MONITOR COLUMN switch is set to OFF, the command is rejected because Debug Tool can only display values in one scrollable line when the setting of MONITOR COLUMN is ON. You must first enter the SET MONITOR COLUMN ON command.

If you enter the SET MONITOR COLUMN OFF command while the SET MONITOR WRAP switch is set to OFF, the command is rejected. The Monitor window must be in columnar format to be able to display values in one scrollable line. You must first enter the SET MONITOR WRAP ON command.

Example
• Enter the following command to specify that you do not want line numbers displayed in the Monitor window:
  SET MONITOR NUMBERS OFF;
• Enter the following command to specify that you do not want variable values to wrap to the next line:
  SET MONITOR WRAP OFF;

SET MSGID
Controls whether the Debug Tool messages are displayed with the message prefix identifiers. The initial setting is OFF.

ON  Displays message identifiers. The first 7 characters of the message contain the EQAnnnn message prefix identifier, then a blank, then the original message text, such as: ‘EQA2222 Program does not exist.’

OFF  Displays only the message text.

Example

Specify that message identifiers are suppressed.
SET MSGID OFF;

SET NATIONAL LANGUAGE
Switches your application to a different run-time national language that determines what translation is used when a message is displayed. The switch is effective for the entire run-time environment; it is not restricted to Debug Tool activity only. The initial setting is supplied by Language Environment or the NATLANG Debug Tool run-time option, according to the setting in the current enclave.

language_code
A valid three-letter set that identifies the language used or (for compatibility) one of the two-letter language codes that was accepted in the previous release of INSPECT for C/370™ and PL/I. The language code can have one of the following values:
United States English: ENU
United States English (Uppercase): UEN
Japanese: JPN
Korean: KOR
If you enter the SET DBCS ON command and then you set the national language to ENU, Debug Tool resets the national language to UEN to remain compatible with DBCS characters.

For compatibility with the previous release of INSPECT for C/370 and PL/I:
EN or ENGLISH is mapped to ENU
UE or UENGLISH is mapped to UEN
JA, JAPANESE, NI, or NIHONGO is mapped to JPN

Usage notes
• In order to display DBCS characters correctly in full-screen mode, the high order bit of the Language field in the VTAM Attribute Byte that must be set ON. To verify that this bit is set ON:
  1. In ISPF, select option 0 (Settings).
  2. On the command line, enter: environ.
  3. Tab to the section Terminal Status (TERMSTAT). In the Enable field, enter 2 (Query terminal information).
  4. Several pages of statistics appear. In the section GTTERM Information, note the value of the highest bit in the second byte of the field Attribute Byte. The value of this bit must be 1 (ON). For example, if the value of the Attribute Byte field is x'000000C9', then DBCS characters display correctly because the second byte is x'80'. However, if the value of the Attribute Byte field is x'000000C9', DBCS characters are not displayed properly. Contact the VTAM System Administrator to set the high order bit of the Language Field of the VTAM Attribute Byte to 1 (ON).
• The language you select by using the SET NATIONAL LANGUAGE command affects both your application and Debug Tool.
• At the beginning of an enclave, the settings are those provided by Language Environment, your operating system, or the NATLANG Debug Tool run-time option. For nested enclaves, the parent’s settings are restored upon return from a child enclave.

Examples
• Set the current national language to Japanese.
  SET NATIONAL LANGUAGE JPN;
• Set the current national language to United States English.
  SET LANGUAGE ENU;

Refer to the following sections for more information related to the material discussed in this section.

Related references
“SET DBCS” on page 182

SET PACE

Specifies the maximum pace of animated execution, in steps per second. The initial setting is two steps per second. This setting is not supported in batch mode.

```plaintext
SET PACE number;
```
**Usage notes**

- If you are debugging a CICS program, choose your pace carefully. After animated execution begins, you might not be able to stop it. See the **Debug Tool User's Guide** for information about requesting an attention interrupt during interactive sessions.
- Associated with the SET PACE command is the STEP command. Animated execution is achieved by defining a PACE and then issuing a STEP n command where n is the number of steps to be seen in animated mode. STEP * can be used to see all steps to the next breakpoint in animated mode.
- When PACE is set to 0, no animation occurs.

**Example**

Set the animated execution pace to 1.5 steps per second.

```
SET PACE 1.5;
```

**SET PFKEY**

Associates a Debug Tool command with a Program Function key (PF key). This setting is not supported in batch mode.

```
SET PFn string command;
```

- **PFn**
  A valid program function key specification (PF1 - PF24).

- **string**
  The label shown in the PF key display (if the KEYS setting is ON) that is entered as a string constant. The string is truncated if longer than eight characters. If the string is omitted, the first eight characters of the command are displayed. The string must be surrounded by double quotation marks for C and C++. For COBOL, PL/I, assembler, and disassembly, the string can be surrounded by either single or double quotation marks.

- **command**
  A valid Debug Tool command or partial command.

**Usage notes**

- If you specify the ? as the command, the ? is understood to be the command, not a request for syntax help.
- In Debug Tool, if there is any text on the command line at the time the PF key is pressed, that text is appended to the PF key string, with an intervening blank, for execution.
- The initial settings for PF keys 13-24 are equivalent to PF keys 1-12, respectively. If you change the setting for a PF key in the 1-12 range, the equivalent key in the 13-24 range remains the same.

**Example**

Define the PF5 key to scroll the cursor-selected window forward.

- If the programming language setting is COBOL:
SET PF5 "Down" = IMMEDIATE SCROLL DOWN;

- If the programming language setting is PL/I:
  SET PF5 'Down' = IMMEDIATE SCROLL DOWN;
- If the programming language setting is C++:
  SET PF5 "Down" = IMMEDIATE SCROLL DOWN;

In all cases, the setting for PF17 remains the same.

**SET PROGRAMMING LANGUAGE**

Sets the current programming language. You can only set the current programming language to the selection of languages of the programs currently loaded. For example, if the current load module contains both C and COBOL compile units, but not PL/I, you can set the language only to C or COBOL. However, if you later STEP or GO into another load module that contains C, COBOL, and PL/I compile units, you can set the language to any of the three.

The programming language setting affects the parsing of incoming Debug Tool commands. The execution of a command is always consistent with the current programming language setting that was in effect when the command was parsed. The programming language setting at execution time is ignored.

```
SET PROGRAMMING LANGUAGE
```

<table>
<thead>
<tr>
<th>CYCLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOMATIC</td>
</tr>
<tr>
<td>HOLD</td>
</tr>
<tr>
<td>(1) ASSEMBLER</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>COBOL</td>
</tr>
<tr>
<td>DISASSEMBLY</td>
</tr>
<tr>
<td>OSVSCOBOL</td>
</tr>
<tr>
<td>PL/I</td>
</tr>
</tbody>
</table>

**Notes:**

1. Available only with Debug Tool Utilities and Advanced Functions (5655-R45).

**CYCLE**

Specifies that the programming language is set to the next language in the alphabetic sequence of supported languages.

**AUTOMATIC**

Cancels a HOLD by specifying that the programming language is set according to the current qualification and thereafter changed automatically each time the qualification changes or STEP or GO is issued.

**HOLD**

Specifies that the given language (or the current language, if no language is specified) remains in effect regardless of qualification changes. The language remains in effect until SET PROGRAMMING LANGUAGE changes the language or releases the hold.

**ASSEMBLER**

Sets the current programming language to ASSEMBLER.

**C**

Sets the current programming language to C. Debug Tool does not differentiate between C and C++, use this option for C++ as well as C programs.
COBOL
Sets the current programming language to COBOL.

DISASSEMBLY
Sets the current programming language to disassembly.

OSVSCOBOL
Sets the current programming language to OS/VS COBOL.

PLI
Sets the current programming language to PL/I.

Usage notes
- If CYCLE or one of the explicit programming language names is specified, the current programming language setting is changed regardless of the currently suspended program or the current qualification.
- The current programming language setting affects how commands are parsed, not how they are performed. Commands are always performed according to the programming language setting where they were parsed. For example, it is not possible for a Debug Tool procedure to contain a mixture of C and COBOL commands; there is no way for the programming language setting to be changed while the procedure is being parsed. Also, it is not possible for a command parsed with one programming language setting to reference variables, types, or labels in another programming language.
- If SET PROGRAMMING LANGUAGE AUTOMATIC is in effect (that is, HOLD is not in effect), changing the qualification automatically sets the current programming language to the specified block or compile unit.
- SET PROGRAMMING LANGUAGE can be used to set the programming language to any supported language in the current or parent enclaves.

Example
Specify that C or C++ is the current programming language.
SET PROGRAMMING LANGUAGE C;

SET PROMPT (full-screen mode)
Controls whether the current program location is automatically shown as part of the prompt message in line mode. It has no effect in full-screen mode, because the current location is always shown in the panel header in that case. The initial setting is LONG.

LONG
Uses long form of prompt message.

SHORT
Uses short form of prompt message.

Example
Specify that the long form of prompt message is used.
SET PROMPT LONG;
SET QUALIFY
Simplifies the identification of references and statement numbers by resetting the point of view to a new block, compile unit, or load module. In full-screen mode this affects the contents of the Source window. If you are currently viewing one compile unit in your Source window and you want to view another, issue the SET QUALIFY command to change the qualification. The SET keyword is optional. The QUALIFY keyword can be abbreviated.

```plaintext
SET QUALIFY BLOCK block_spec ;
```

**BLOCK**
Sets the current point of view to the specified block.

`block_spec`
A valid block specification.

**CU**
Sets the current point of view to the specified compile unit. CU is equivalent to PROGRAM.

`cu_spec`
A valid compile unit specification.

**PROGRAM**
Is equivalent to CU.

**LOAD**
Sets the current point of view to the specified load module.

`load_spec`
A valid load module specification. If omitted, the initial (primary) load module qualification is used.

**RESET**
Resets qualification to the block of the suspended program and (if the SCREEN setting is ON) scrolls the source window to display the current statement line.

**RETURN**
Switches qualification to the next higher calling program.

**UP**
Switches qualification up one lexical level to the statically containing block.

**Usage notes**
- If SET PROGRAMMING LANGUAGE AUTOMATIC is in effect (that is, HOLD is not in effect), changing the qualification automatically sets the current programming language to the specified block or compile unit.
- If you are debugging a program that has multiple enclaves, you can issue the SET QUALIFY command only for the following items:
  - Load modules, compile units, and blocks that are known to Debug Tool and are in the current enclave
  - Load modules, compile units, and blocks that are not known to Debug Tool
  - Non-LE assembler compile units in a higher-level enclave
You cannot issue the SET QUALIFY command for a load module that is part of a higher-level enclave. You cannot issue the SET QUALIFY command for compile units in a higher-level enclave unless the compile unit is non-Language Environment.

- The SET QUALIFY command does not imply a change in flow of control when the program is resumed with the G0 command.
- The SET QUALIFY command cannot modify the point of view to a Debug Tool or library block.
- SET QUALIFY LOAD will not change the results of the QUERY QUALIFY command.

- If you specify cu_spec as a CU name without a load module name, Debug Tool searches for the CU in the following order:
  1. CUs in the currently qualified load module.
  2. All known CUs.
  3. A CU by the specified name in a load module of the same name.

If you comply with the following conditions, Debug Tool executes an implicit LOAD command for the load module:
- You enter the SET QUALIFY LOAD command or SET QUALIFY CU command and specify the name of a load module that is not currently known to Debug Tool.
- You install Debug Tool Utilities and Advanced Functions.

If the implicit LOAD is successful, implicit CUs are created for the following types of programs:
- All CUs in the load module except COBOL and disassembly CUs
- If SET DISASSEMBLY ON is in effect, disassembly CUs
- If the entry point of the load module is a disassembly program, regardless of the setting of SET DISASSEMBLY.

With implicit CUs, you can do debugging tasks such as setting breakpoints and browsing the source of the CU. When you run the program by entering a command such as G0 or STEP, the implicitly loaded modules are deleted, any breakpoints created in the implicitly created CUs are suspended, and all implicitly created CUs are destroyed. If the CU is later created during normal program execution, the suspended breakpoints are reactivated.

- If you comply with the following conditions, Debug Tool creates an implicit CU:
  - You enter a SET QUALIFY CU command that specifies the name of a COBOL CU that has not yet been created because the CU has not been run.
  - You install Debug Tool Utilities and Advanced Functions.

With implicit CUs, you can do debugging tasks such as setting breakpoints and browsing the source of the CU. When you run the program by entering a command such as G0 or STEP, any breakpoints created in the implicitly created CUs are suspended and all implicitly created CUs are destroyed. If the CU is later created during normal program execution, the suspended breakpoints are reactivated.

- In order use the SET QUALIFY LOAD or SET QUALIFY CU commands to create implicit CUs for a COBOL program, the PTF for Language Environment APAR PK30521 must be installed on z/OS Version 1 Release 4, Version 1 Release 5, Version 1 Release 6, Version 1 Release 7, and Version 1 Release 8.

- If you stop in an enclave where Language Environment is not yet active, you cannot use SET QUALIFY LOAD or SET QUALIFY CU commands to load a Language Environment load module. You can only use these commands to load a Language Environment load module after Language Environment has been initialized in the current enclave.
Examples

- Indicate to Debug Tool that the load module statmod should be used when no load module is specified.
  ```
  SET QUALIFY LOAD statmod;
  ```

- Set the qualification back to the point of the suspended program.
  ```
  SET QUALIFY RESET;
  ```

- Set the block qualification to blockx. As a result, the load module qualification and compile unit qualification will be updated to the load module and compile unit that contain the block blockx.
  ```
  SET QUALIFY BLOCK blockx;
  ```

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

- "block_spec" on page 11
- "cu_spec" on page 13
- "load_spec" on page 15

**SET REFRESH (full-screen mode)**

Controls screen refreshing. This command is only valid when in full-screen mode, that is the SET SCREEN setting is ON. The initial setting for REFRESH is OFF.

```
SET REFRESH [ON|OFF];
```

**ON**

Clears the screen before each rewrite. This is a required setting if your application handles line mode I/O.

**OFF**

Rewrites without clear.

**Usage note**

SET REFRESH ON is needed for applications that also make use of the screen; for example, applications using ISPF services to display panels.

**Example**

Specify that rewrites only occur on those portions of the screen that have changed. The screen is not cleared before being rewritten.

```
SET REFRESH OFF;
```

**SET RESTORE**

Controls the restoring of settings, breakpoints, and monitor specifications.

```
SET RESTORE SETTINGS [BPS|MONITORS] [NOAUTO|AUTO];
```

**SETTINGS**

- Indicates that SET values and WINDOW SIZE and WINDOW CLOSE settings are to be restored. The following SET values are not restored:
• SET DBCS
• SET FREQUENCY
• SET NATIONAL LANGUAGE
• SET PROGRAMMING LANGUAGE
• FILE operand of SET RESTORE SETTINGS
• SET QUALIFY
• SET SOURCE
• SET TEST

**BPS**
Indicates that breakpoints and LOADDEBUGDATA (LDD) specifications are to be restored. The following breakpoints are restored:
• APPEARANCE breakpoints
• CALL breakpoints
• DELETE breakpoints
• ENTRY breakpoints
• EXIT breakpoints
• GLOBAL APPEARANCE breakpoints
• GLOBALCALL breakpoints
• GLOBAL DELETE breakpoints
• GLOBAL ENTRY breakpoints
• GLOBAL EXIT breakpoints
• GLOBAL LABEL breakpoints
• GLOBAL LOAD breakpoints
• GLOBAL STATEMENT and GLOBAL LINE breakpoints
• LABEL breakpoints
• LOAD breakpoints
• OCCURRENCE breakpoints
• STATEMENT and LINE breakpoints
• TERMINATION breakpoint

**MONITORS**
Indicates that monitor and LOADDEBUGDATA (LDD) specifications are to be restored.

**NOAUTO**
Indicates that the specified data is not to be restored automatically at Debug Tool startup. It will be restored only when you explicitly request it by entering the RESTORE command. NOAUTO is the default until AUTO is specified.

**AUTO**
Indicates that, if possible, the specified data set is to be automatically restored by Debug Tool at startup.

**Usage notes**
• When you use SET RESTORE BPS AUTO or SET RESTORE MONITORS AUTO, you must also use SET RESTORE SETTINGS AUTO. This ensures that Debug Tool knows the values of SET RESTORE BPS and SET RESTORE MONITORS the next time it starts up and then can implement them.
• Monitors are not necessarily restored to the same slot number from which they were saved.
If you are debugging a CICS program and you want to use SET RESTORE parameter_name AUTO, you must log on with a user ID that is different from the default user ID.

If you are debugging DB2 stored procedures or IMS/DC programs, you must do one of the following:
- Ensure that the default data set does not exist.
- Ensure that the name of the default data set is NULLFILE.
- Change the name of the data set by using the SET SAVE SETTINGS command.

Because multiple users share the same default data set, other users can alter the settings in that data set. You can use EQAOPTS to specify NULLFILE as the name of the default data set.

**Related references**

“RESTORE command” on page 168
“SET SAVE”

**SET REWRITE**

Forces a periodic screen rewrite during long sequences of output. This setting is not supported in batch mode.

```plaintext
SET REWRITE EVERY number;
```

`number`

Specifies how many lines of intercepted output are written by the application program before Debug Tool refreshes the screen. The initial setting is 50.

**Example**

Force screen rewrite after each 100 lines of screen output.

```plaintext
SET REWRITE EVERY 100;
```

**SET SAVE**

Controls the saving of settings, breakpoints, and monitor specifications.

```plaintext
SET SAVE SETTINGS NOAUTO FILE setfileid;
```

**SETTINGS**

Indicates that SET values and WINDOW SIZE and WINDOW CLOSE settings are to be saved. The following SET values are not saved:
- `SET DBCS`
- `SET FREQUENCY`
- `SET NATIONAL LANGUAGE`
- `SET PROGRAMMING LANGUAGE`
- `FILE` operand of SET RESTORE SETTINGS
- `SET QUALIFY`
• SET SOURCE
• SET TEST

**BPS**
Indicates that breakpoints and LOADDEBUGDATA (LDD) specifications are to be saved. The following breakpoints are saved:
• APPEARANCE breakpoints
• CALL breakpoints
• DELETE breakpoints
• ENTRY breakpoints
• EXIT breakpoints
• GLOBAL APPEARANCE breakpoints
• GLOBALCALL breakpoints
• GLOBAL DELETE breakpoints
• GLOBAL ENTRY breakpoints
• GLOBAL EXIT breakpoints
• GLOBAL LABEL breakpoints
• GLOBAL LOAD breakpoints
• GLOBAL STATEMENT and GLOBAL LINE breakpoints
• LABEL breakpoints
• LOAD breakpoints
• OCCURRENCE breakpoints
• STATEMENT and LINE breakpoints
• TERMINATION breakpoint

**MONITORS**
Indicates that GLOBAL monitor and LOADDEBUGDATA (LDD) specifications are to be saved. LOCAL monitor specifications are not saved.

**NOAUTO**
Indicates that at Debug Tool termination, the specified settings, breakpoint, or specifications are not to be saved. NOAUTO is the default until AUTO is specified.

**AUTO**
Indicates that, if possible, the specified data is to be saved at Debug Tool termination.

**ONCE**
Indicates that the settings information is to be saved once. The settings information is saved at the termination of the current debugging session but the saved value for SET SAVE SETTINGS is NOAUTO. This enables you save the settings of the current debugging session and not have the settings updated at the termination of subsequent debug sessions.

* Indicates that the default file name is to be used to save settings, breakpoints, and monitor specifications at termination. The default name is userid.DBGTOOL.SAVESETS for settings and userid(DBGTOOL).SAVEBPS for breakpoints and monitor specifications. You can modify the default names by using EQAOPTS.

**FILE setfileid**
Indicates the data set name to be used to save and restore settings. The data set must exist prior to running this command.
In z/OS, setfileid is a DD name, a fully-qualified data set name (without quotes), or an HFS path and file name. In CICS, setfileid is a fully-qualified data set name or an HFS path and file name.

If setfileid is less than nine characters in length and does not contain a period, Debug Tool assumes it is a DD name. Otherwise, it is assumed to be a fully-qualified data set name.

In batch mode, the data set name is ignored. Use the INSPSAFE DD statement to indicate the name of the data set to use to restore and save settings.

This data set must be a sequential data set with a record format (RECFM) of VB and with a record length (LRECL) greater than or equal to 3204.

FILE bpfileid
Indicates the data set to be used to save breakpoints and monitor specifications. The data set must exist prior to running this command.

In z/OS, bpfileid is a DD name, a fully-qualified data set name (without quotes), or an HFS path and file name. In CICS, bpfileid is a fully-qualified data set name or an HFS path and file name.

If bpfileid is less than nine characters in length and does not contain a period, Debug Tool assumes it is a DD name. Otherwise, it is assumed to be a fully-qualified data set name.

In batch mode, the data set name is ignored. Use the INSPBPM DD statement to indicate the name of the data set to use to save breakpoints and monitor specifications.

This data set must be a PDS or PDSE (a PDSE is recommended) and you cannot specify a member name. This data set must have a record format (RECFM) of VB and with a record length (LRECL) greater than or equal to 3204. Debug Tool assigns a member name that is the load module name at enclave start. The breakpoints for each enclave are saved in a separate member of the PDS or PDSE. If you want to discard any saved breakpoints, LDD specifications, and monitor specifications, you can delete the member that has the name of the load module that started the enclave. Do not alter the contents of the member.

Usage notes
• You cannot use AUTO when you are debugging a CICS program and you are logged in with the same user ID as the default user ID.
• When you are debugging a CICS program, the CICS region must have update authorization to the SAVE SETTINGS and SAVE BPS data sets.
• When you enter the QUIT or QQ command from a nested enclave and the SET SAVE BPS AUTO, SET SAVE MONTORS AUTO, or both are in effect, only the data for the lowest level enclave is saved. No data for the higher level enclaves is saved.
• If you are debugging DB2 stored procedures or IMS/DC programs, you must do one of the following:
  – Ensure that the default data set does not exist.
  – Ensure that the name of the default data set is NULLFILE.
  – Change the name of the data set by using the SET SAVE SETTINGS command. Because multiple users share the same default data set, other users can alter the settings in that data set. You can use EQAOPTS to specify NULLFILE as the name of the default data set.
• Specifying setdsn for SAVE SETTINGS does not change the name of the data set from which AUTO RESTORE SETTINGS is done. It only changes the name of...
the data set used by AUTO SAVE SETTINGS and the RESTORE SETTINGS commands. AUTO RESTORE SETTINGS is always done from the default data set or DD name, depending on the environment.

Refer to the following sections for more information related to the material discussed in this section.

Related tasks

Debug Tool User’s Guide

Related references

“SET RESTORE” on page 205
“RESTORE command” on page 168

SET SCREEN (full-screen and line mode)

Controls how information is displayed on the screen. The initial setting for a supported full-screen terminal is ON.

\[
\text{SET SCREEN} \quad \begin{cases} \text{ON} & \text{if neither CYCLE nor integer is specified} \\ \text{OFF} & \text{if SET SCREEN OFF is entered} \end{cases}
\]

CYCLE

Switches to the next window configuration in sequence.

integer

An integer in the range 1 to 6, selecting the window configuration. The initial setting is 1.

LOG or MONITOR or SOURCE

Specifies the sequence of window assignments within the selected configuration (left to right, top to bottom). There must be no more than three objects specified and they must all be different.

ON

Activates the Debug Tool full-screen services.

OFF

Activates line mode. This mode is forced if the terminal is not a supported full-screen device.

Usage notes

- If neither CYCLE nor integer is specified, there is no change in the choice of configuration. If no windows are specified, there is no change in the assignment of windows to the configuration.
- If SET SCREEN OFF is entered while debugging in full-screen mode using a VTAM terminal under TSO, the session enters line mode using the TSO terminal. If SET SCREEN ON is later entered from the TSO terminal, control reverts to full-screen mode using a VTAM terminal.
- SET SCREEN OFF is ignored in CICS full-screen mode and in z/OS batch while debugging in full-screen mode using a VTAM terminal.

Examples

- Indicate that the Debug Tool full-screen services are used.
SET SCREEN ON;

- Indicate that the log window is positioned above the Source window on the left hand side of the screen and the monitor window is to occupy the upper right side portion of the screen.

SET SCREEN 2 LOG MONITOR;

Refer to the following sections for more information related to the material discussed in this section.

**Related tasks**

*Debug Tool User's Guide*

**SET SCROLL DISPLAY (full-screen mode)**

Controls whether the scroll field is displayed when operating in full-screen mode. The initial setting is ON.

```
SET SCROLL DISPLAY [ON/OFF];
```

**ON** Displays scroll field.

**OFF** Suppresses scroll field.

**Example**

Specify that the scroll field is suppressed.

```
SET SCROLL DISPLAY OFF;
```

**SET SEQUENCE (PL/I)**

Controls whether Debug Tool interprets data after column 72 in a commands or preference file as a sequence number.

```
SET SEQUENCE [ON/OFF];
```

**ON** Allows sequence numbers in 73-80 columns in the commands or preferences file.

**OFF** Does not allow sequence numbers in the commands or preferences file.

**Usage note**

If you have sequence numbers placed in 73-80 columns, you have to enter the SET SEQUENCE ON command as the first command of your commands or preferences file. Afterwards, Debug Tool processes 1-72 columns and ignores everything after column 72.

**SET SOURCE**

Associates a source file, compiler listing or separate debug file with one or more compile units.
**ON**  Displays the source or listing for a compile unit when the compile unit is active.

**OFF**  Specifies that the file is not displayed.

**cu_spec**  A valid compile unit specification. Multiple compile units can be associated with the same source, listing or separate debug file.

**fileid**  Identifies the source, listing or separate debug file to be used for the compile unit. The file that you specify must be of fixed block format. You cannot specify concatenated data sets.

In z/OS, **fileid** is a DD name, a fully qualified partitioned data set and member name, a sequential file, or an HFS path and file name.

In CICS, **fileid** is a fully-qualified data set name or an HFS path and file name.

If **fileid** is less than nine characters in length and does not contain a period, Debug Tool assumes it is a DD name. Debug Tool checks to see if it is allocated. If it is not allocated, then **fileid** is assumed to be a data set name.

**Fileid** specifies a file identifier used in place of the default file identifier for the compile unit. A valid **fileid** is required unless it is already known to Debug Tool (by using a previous **SET SOURCE** command) or the default **fileid** is valid.

**Fileid** can not be a DD name if the data set allocated to it is C, C++ or Enterprise PL/I source and the EQAOP TS SUBSYS=ssss option is being used to access a source file in a library system.

**Usage notes**

- If you compiled your C or C++ program with the **FORMAT(DWARF)** suboption of the **DEBUG** compiler option, you cannot use the **SET SOURCE** command to specify the new location of the file generated by the **FILE** suboption of the **DEBUG** compiler option. Use the **EQAD EBUG** statement to specify the new location of the file.
- When **SET SOURCE** is issued for the currently executing compile unit, a test is performed for the existence of the file. If the compile unit is **not** the current compile unit, this test is not performed until the compile unit becomes current. The file associated with the source might not exist and the error message for the nonexistent file does not appear until a function that requires this file is attempted.
- The **SET SOURCE ON** command has a higher precedence than the **SET DEFAULT LISTINGS** command.
- For **COBOL**, if the **cu_spec** includes any names that are case sensitive, enclose the name in single or double quotes.
- The **SET SOURCE** command has no effect on a disassembly compile unit. However, it is saved and might apply later if the compile unit is specified as the operand of the **LOADDEBUGDATA** command.
- If the file name does not fit on one line, suffix it with a trailing hyphen.
Examples

- Indicate that the COBOL listing associated with compile unit prog1 is found in DD name mainprog. In a TSO session, allocate the listing data set:

  ```
  ALLOCATE FI(MAINPROG) DA('JSMITH.COBOL.LISTING(PROG1)') SHR
  ```

  Start Debug Tool and issue:

  ```
  SET SOURCE ON (prog1) mainprog;
  ```

  When prog1 is made current during the debug session, Debug Tool searches for the listing in JSMITH.COBOL.LISTING(PROG1).

- Indicate that the COBOL listing associated with compile unit prog1 is found in DD name mainprog. In a TSO session:

  ```
  SET SOURCE ON (prog1) JSMITH.COBOL.LISTING(PROG1)
  ```

  This accomplishes the same result as the previous example without the execution of the ALLOCATE command.

- Indicate that the source associated with compile unit "/u/userid/code/oefun.c" is found in the HFS under the path and file name "/u/userid/code/oefun.c".

  ```
  SET SOURCE ON ("/u/userid/code/oefun.c") /u/userid/code/oefun.c;
  ```

- Indicate that the PL/I listing file associated with compile unit AVER is found in MYID.PLI.LISTING(AVER)

  ```
  SET SOURCE ON (AVER) myid.pli.listing(AVER);
  ```

- Indicate that the C source associated with compile unit JSMITH.C.SOURCE(myprog) is found in the PDS and member CODE.CLIB.SOURCE(myprog).

  ```
  SET SOURCE ON ("JSMITH.C.SOURCE(myprog)") CODE.CLIB.SOURCE(myprog)
  ```

- Enter the SET LONGCUNAME OFF command to indicate that you want to use short CU names, then indicate that the C source associated with compile unit JSMITH.C.SOURCE(myprog) is found in the PDS and member CODE.CLIB.SOURCE(myprog):

  ```
  SET LONGCUNAME OFF;
  SET SOURCE ON (myprog) CODE.CLIB.SOURCE(myprog)
  ```

- A PL/I program is compiled with the Enterprise PL/I compiler. Indicate that the PL/I source associated with compile unit JSMITH.PLI.SOURCE(myprog) is found in the PDS and member CODE.PLILIB.SOURCE(myprog):

  ```
  SET LONGCUNAME OFF;
  SET SOURCE ON (myprog) CODE.PLILIB.SOURCE(myprog)
  ```

- A PL/I program is compiled with the following compiler and it is running in the following environment:

  ```
  - Enterprise PL/I for z/OS, Version 3.5, compiler with the PTFs for APARs PK35230 and PK35489 applied
  - Language Environment Version 1.4 through 1.8 with the PTF for APAR PK33738 applied
  ```

  Indicate that the PL/I source associated with compile unit JSMITH.PLI.SOURCE(myprog) is found in the PDS and member CODE.PLILIB.SOURCE(myprog):

  ```
  SET SOURCE ON (myprog) CODE.PLILIB.SOURCE(myprog)
  ```

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

- "cu spec" on page 13
- "LIST command" on page 124
**SET SUFFIX (full-screen mode)**

Controls the display of frequency counts at the right edge of the Source window when in full-screen mode. The initial setting is ON.

```
  SET SUFFIX [ON] [OFF] ;
```

**ON** Displays the suffix column.

**OFF**
 Suppresses the suffix column.

**Example**

Specify that the suffix column is displayed.

```
  SET SUFFIX ON;
```

**SET TEST**

Overrides the initial TEST run-time options specified at invocation. The initial setting is ALL.

```
  SET TEST test_level [(-test_level-)] ;
```

*test_level*

Specifies what exception conditions cause Debug Tool to gain control, even though no breakpoint exists. The parentheses are optional.

*Test_level* can include the following:

**ALL**

Specifies that the occurrence of any of the following causes Debug Tool to gain control:

- attention interrupt
- abnormal termination of the Language Environment enclave, or an MVS or CICS ABEND when executing without the Language Environment run time
- normal termination of Language Environment due to a COBOL STOP RUN, PL/I STOP, or EXEC CICS RETURN
- Language Environment condition of severity 1 or above

If a condition occurs and a breakpoint exists for the condition, the commands specified in the breakpoint are executed. If a condition occurs and a breakpoint does not exist for that condition, or if an attention interrupt occurs, Debug Tool:

- In interactive mode, reads commands from a commands file (if it exists) or prompts you for commands, or
- In noninteractive mode, reads commands from the commands file

**ERROR**

Specifies that only the following conditions cause Debug Tool to gain control without a user-defined breakpoint:

- An MVS or CICS ABEND that occurs when you are running without the Language Environment run time

```
  ERROR;
```

---

214 Debug Tool V7.1 Reference and Messages
• For C:
  – An attention interrupt
  – A predefined Language Environment condition of Severity 2 or above
  – Any C condition other than SIGUSR1, SIGUSR2, SIGINT or SIGTERM.
• For COBOL:
  – An attention interrupt
  – A predefined Language Environment condition of Severity 2 or above.
• For PL/I:
  – An attention interrupt, directed at either PL/I or Debug Tool
  – A predefined Language Environment condition of Severity 2 or above.

If a breakpoint exists for one of the above conditions, any commands specified in the breakpoint are executed. If no commands are specified, Debug Tool reads commands from a commands file or prompts you for commands in interactive mode.

NONE
Specifies that Debug Tool gains control only at an attention interrupt, or at a condition if a breakpoint is defined for that condition. If a breakpoint does exist for the condition, the commands specified in the breakpoint are executed.

Examples
• Indicate that only an attention interrupt or exception causes Debug Tool to gain control when no breakpoint exists.
  SET TEST ERROR;
• Indicate that no condition causes Debug Tool to gain control unless a breakpoint exists for that condition.
  SET TEST NONE;

Refer to the following sections for more information related to the material discussed in this section.

Related tasks
  Debug Tool User’s Guide

Related references
  • z/OS Language Environment Debugging Guide

SET WARNING (C, C++, COBOL, and PL/I)
Controls display of the Debug Tool warning messages and whether exceptions are reflected to the C, C++, and PL/I programs. For COBOL programs, controls the ability to modify variables while you debug optimized code. The initial setting is ON.

ON  Displays the Debug Tool warning messages, and conditions such as a divide check result in a diagnostic message. For COBOL programs, prohibits the modification of variables while you debug optimized programs.

OFF  Suppresses the Debug Tool warning messages, and conditions raise an exception in the program. For COBOL programs, allows the modification of variables while you debug optimized programs.
Exceptions that occur due to interaction with you are likely to be due to typing errors and are probably not intended to be passed to the application program. However, you might want to raise a real exception in the program, for example, to test some error recovery code. (TRIGGER is not always appropriate for this because it does not set up the exception information.)

Usage notes
- You can use this command in remote debug mode.
- Debug Tool detects C conditions such as the following:
  - Division by zero
  - Array subscript out of bounds for defined arrays
  - Assignment of an integer value to a variable of enumeration data type where the integer value does not correspond to an integer value of one of the enumeration constants of the enumeration data type.
- Debug Tool detects the following PL/I computational conditions:
  - Invalid decimal data
  - CHARACTER to BIT conversion errors
  - Division by zero
  - Invalid length in varying strings
- You can modify variables in an optimized program that was compiled with one of the following compilers:
  - Enterprise COBOL for z/OS and OS/390, Version 3 Release 2
  - Enterprise COBOL for z/OS and OS/390, Version 3 Release 1 with APAR PQ63235 installed
  - COBOL for OS/390 & VM, Version 2 Release 1 with APAR PQ63234 installed
However, the new value might not be used even if you list the variable and Debug Tool displays the new value. The program uses the value stored in the temporary storage location or register, which you cannot access.

You must apply Language Environment APAR PQ71779 before you can modify variables in an optimized program compiled with the above compilers.
- This command has no effect on Enterprise PL/I programs.

Example

Specify that conditions result in a diagnostic message.
SET WARNING ON;

Refer to the following sections for more information related to the material discussed in this section.

Related tasks
- Debug Tool User's Guide

SET command (COBOL)

The SET command assigns a value to a COBOL reference. The SET keyword cannot be abbreviated.

```
SET reference TO reference literal TRUE
```
reference
A valid Debug Tool COBOL reference.

literal
A valid COBOL numeric literal constant.

TRUE
The value assigned to a level 88 reference.

In order to assign the value TRUE, the PTF for Language Environment APAR PK30521 must be installed on z/OS Version 1 Release 4, Version 1 Release 5, Version 1 Release 6, Version 1 Release 7, and Version 1 Release 8.

Usage notes
• You can assign the value TRUE only to a level 88 reference.
• If Debug Tool was started because of a computational condition or an attention interrupt, using an assignment to set a variable might not give expected results. This is due to the uncertainty of variable values within statements as opposed to their values at statement boundaries.
• SET assigns a value only to a single receiver; unlike COBOL, multiple receiver variables are not supported.
• Only formats 1, 4 and 5 of the COBOL SET command are supported.
• Index-names can only be program variables (because OCCURS is not supported for the Debug Tool session variables).
• COBOL ADDRESS OF identifier is supported only for identifiers that are LINKAGE SECTION variables. In addition, COBOL ADDRESS OF as a receiver must be level 1 or 77, and COBOL ADDRESS OF as a sender can be any level except 66 or 88.
• Debug Tool provides a hexadecimal constant that can be used with the SET command, where the hexadecimal value is denoted by an "H" and delimited by quotation marks or apostrophes.
• If the DATA option of the PLAYBACK ENABLE command is in effect, you can use the SET command to assign a value only to a session variable. You cannot assign a value to a program variable.
• If you are debugging an optimized COBOL program, you can use the SET command to assign a value to a program variable only if you first enter the SET WARNING OFF command. The source or target of the SET command cannot reference a variable that was discarded by the optimizer.

Examples
• Assign the value 3 to inx1, the index to itm-1.
  SET inx1 TO 3;
• Assign the value of inx1 to inx2.
  SET inx2 TO inx1;
• Assign the value of an invalid address (nonnumeric 0) to ptr and:
  SET ptr TO NULL;
• Assign the address of one to ptr.
  SET ptr TO ADDRESS OF one;
• Assigns the hexadecimal value of '20000' to the pointer ptr.
  SET ptr TO H'200000';

Refer to the following sections for more information related to the material discussed in this section.

Related tasks
Refer to the following sections for more information related to the material discussed in this section.

**Related references**

“Allowable moves for the Debug Tool SET command”

### Allowable moves for the Debug Tool SET command

The following table shows the allowable moves for the Debug Tool SET command.
<table>
<thead>
<tr>
<th>Source Field</th>
<th>AO</th>
<th>IN</th>
<th>IDI</th>
<th>ED</th>
<th>BI</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Field</td>
<td>Address Of (AO)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Index Name (IN)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Index Data Item (IDI)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Pointer Data Item (PTR)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Address Hex Literal</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>NULL (NUL)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Integer Literal</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>External Decimal (ED)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Binary (BI)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Internal Decimal (ID)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Object Reference (OR)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
Notes:
1 Must be hexadecimal characters only, delimited by either double ('') or single ('') quotation marks and preceded by H.
2 Index name is converted to index value.

SHOW prefix command (full-screen mode)

The SHOW prefix command specifies what relative statement (for C) or relative verb (for COBOL) within the line is to have its frequency count temporarily shown in the suffix area.

```
SHOW [integer] ;
```

**integer**

Selects a relative statement (for C) or a relative verb (for COBOL) within the line. The default value is 1. For optimized COBOL programs, the default value is the first executable statement which was not discarded by the optimizer.

Usage notes

- If SET SUFFIX is currently OFF, SHOW prefix forces it ON.
- The suffix display returns to normal on the next interaction.
- The SHOW prefix command is not logged.

Example

Display the frequency count of the third statement or verb in the line (typed in the prefix area of the line where the statement is found).

```
SHOW 3
```

No space is needed as a delimiter between the keyword and the integer; hence, SHOW 3 is equivalent to SHOW3.

STEP command

The STEP command causes Debug Tool to dynamically step through a program, executing one or more program statements. In full-screen mode, it provides animated execution.

STEP ends if one or more of the following conditions is reached:

- User attention interrupt
- A breakpoint is encountered
- Normal or unusual termination of the program

```
STEP [integer] INTO OVER RETURN ;
```

**integer**

Indicates the number of statements performed. The default value is 1. If integer is greater than 1, the statement is performed as if it were that many repetitions
of STEP with the same keyword and a count of one. The speed of execution, or the pace of stepping, is set by either the SET PACE command, or with the Pace of visual trace field on the Profile panels.

* Specifies that the program should run until interrupted. STEP * is equivalent to GO.

**INTO**
Steps into any called procedures or functions. This means that stepping continues within called procedures or functions. This is the default except when the called procedure or function is a library or operating system routine.

**OVER**
Steps over any procedure call or function invocations. This operand provides full-speed execution (with no animation) while in called procedures and functions, resuming STEP mode on return. This is the default when the called procedure or function is a library or operating system routine.

If you are debugging a disassembled program, verify that you have set a breakpoint in the calling program. Without the breakpoint, Debug Tool can not resume STEP mode on return and the application continues to run until it ends.

**RETURN**
Steps to the return point the specified number of levels back, halting at the statement following the corresponding procedure call or function invocation. This operand provides full-speed execution (with no animation) for the remainder of the current procedure or function, and for any called procedures or functions, resuming STEP mode on return.

If you are debugging an OS/VS COBOL or disassembled program, do not use the STEP RETURN command because Debug Tool cannot identify the return point. Instead, set a breakpoint in the calling program and enter the GO command.

**Usage notes**
- In the figure below, PGM A calls PGM B.
Assume that the current execution point is on PGM B and, at the line ADD 5 TO MYNUM. At this point, you decide you don’t need to see any more of the code in PGM B. By issuing STEP RETURN on the command line, Debug Tool returns to the first line of code after the CALL command that called PGM B, as indicated by the arrow. You can then continue stepping through PGM A.

- If STEP is specified in a command list (for example, as the subject of an IF command or WHEN clause), all subsequent commands in the list are ignored.
- If STEP is specified within the body of a loop, it causes the execution of the loop to end.
- To suppress the logging of STEP commands, use the SET ECHO command.
- If two operands are given, they can be specified in either order.
- The animation execution timing is set by the SET PACE command.
- The source panel provides a means of suppressing the display of selected listings or files. This gives some control of “debugging scope,” because animated execution does not occur within a load module where the source listing or source file is not displayed.
- If you are debugging a disassembled program and attempt to step out of the current CU, a message appears. The message informs you to set a breakpoint outside the current CU. Without that breakpoint, Debug Tool cannot stop the application. After you have set the breakpoint, you can resume running your application by entering a Debug Tool command like STEP or GO.
• If you are debugging a program that does not use the standard linkage conventions for R13, R14, and R15, and you enter the STEP RETURN or the STEP command on a statement that returns to a higher level CU, Debug Tool does not stop at the expected instruction in the higher-level CU.

• When PLAYBACK ENABLE is in effect, you can use the STEP command to move forward or backward one or more statements. You cannot use the INTO, OVER, and RETURN keywords. Each STEP command moves forward or backward the number of statements specified or implied by the integer parameter.

• If the DATA option of the PLAYBACK ENABLE command is in effect, you can access program variables after each STEP command.

Examples
• Step through the next 25 statements and if an application subroutine or function is called, continue stepping into that subroutine or function.

   STEP 25 INTO;

• Step through the next 25 statements, but if any application subroutines or functions are called, switch to full-speed execution without animation until the subroutine or function returns.

   STEP 25 OVER;

• Return at full speed through three levels of calls.

   STEP 3 RETURN;

**STORAGE command**

The STORAGE command enables you to alter storage. You must be careful when you alter storage because the results can be unpredictable.

```plaintext
   STORAGE (address
              reference
              offset
              length)
             = value;
```

**address**

The address of the first byte of storage that you want to alter. The address must be in one of the following notations:

- For C and C++: 0x followed by a hexadecimal number. For example, 0x1f is the address 1F.
- For COBOL: H followed by a hexadecimal number enclosed in single or double quotes. For example, H'1F' or H*1F*.
- For OS/VS COBOL, X followed by a hexadecimal number enclosed in double quotes.
- For PL/I: a hexadecimal number enclosed in double quotes followed by PX. For example, '1F'PX.
- For assembler and disassembly; X followed by a hexadecimal number enclosed in single or double quotes or an assembler expression that resolves to a storage address. For example, R1->X '120'.

**reference**

A variable whose storage location is to be changed. In assembler or disassembly, this operand may be specified as any assembler expression that represents a storage location.

**'reference'**

An OS/VS COBOL variable whose storage location is to be changed. In OS/VS COBOL, reference must be enclosed in single-quotes.
offset
The decimal or hexadecimal number of bytes indicating the starting offset from
the memory location pointed to by reference’s address or the address provided
by the user. Offset can be a negative number. If offset is a hex constant, it must
follow the language rules for a hex constant, as described for address above.
The default is 0.

length
The decimal number of bytes you want to alter. This must equal the length of

value
The value you want to store. The notation for value must be one of the
following:

- A hexadecimal address. The value must be in one of the following notations:
  - For C and C++: 0x followed by a hexadecimal number. The value can be
    up to four bytes long.
  - For COBOL: $H followed by a hexadecimal number enclosed in single or
double quotes. The value can be up to four bytes long. If the hexadecimal
  address you specify is less than four bytes, it is padded with zeros on the
  left to make it four bytes long. If you specify the integer parameter, its
  must be 4.
  - For OS/VS COBOL: X followed by a hexadecimal number enclosed in
double quotes.
  - For PL/I: a hexadecimal number enclosed in single quotes followed by
PX. The value can be up to four bytes long. If the hexadecimal address
  you specify is less than four bytes, it is padded with zeros on the left to
  make it four bytes long. If you specify the integer parameter, its must be 4.
  - For assembler or disassembly: X followed by a hexadecimal number
enclosed in single or double quotes.
- A hexadecimal string up to eight bytes long. The value must be in one of the
  following notations:
  - For C and C++: 0x followed by a hexadecimal number.
  - For COBOL: $H followed by a hexadecimal number enclosed in single or
double quotes.
  - For OS/VS COBOL: X followed by a hexadecimal number enclosed in
double quotes.
  - For PL/I: a hexadecimal number enclosed in single quotes followed by
PX.
  - For assembler or disassembly: X followed by a hexadecimal number
enclosed in single or double quotes.
- A decimal value. For any decimal value, four bytes are altered. For example,
$STORAGE ($'12345678') = 3 is the same as $STORAGE($'12345678') =
$'00000003'.
- A character string up to 256 bytes long, using the character string notation
appropriate for each language.

Usage notes
- If you specify only two parameters, Debug Tool assumes the second parameter
is the length.
- If you specify only one parameter, Debug Tool assumes the offset is 0 and that
the length is equal to the length of value.
- The STORAGE command can not be used while you replay recorded statements by using the PLAYBACK commands.

Examples
- For C, enter the following command to alter two bytes of storage at address \texttt{X'12345678'}:
  
  ```
  STORAGE (0x12345678) = 0x1234;
  ```

- For COBOL, enter the following command to alter four bytes of storage at address \texttt{X'12345678'}:
  
  ```
  STORAGE (H'12345678') = H'1234' 
  ```

  The command is changed to:

  ```
  STORAGE (H'12345678') = H'00001234' 
  ```

- For COBOL, enter the following command to alter six bytes of storage at address \texttt{X'12345678'}:
  
  ```
  STORAGE (H'12345678') = X'C1C1C1C1C1' 
  ```

- For PL/I, enter the following command to alter two bytes of storage at address \texttt{X'12345678'}:
  
  ```
  STORAGE ('12345678'PX) = 'C1C1'X 
  ```

- For PL/I, enter the following command to alter 23 bytes of storage starting at address \texttt{X'12345678'}:
  
  ```
  STORAGE ('12345678'PX, 10, 20) = 'first and last name '; 
  ```

- Update 20 bytes of storage pointed by an address \texttt{162F0}, language is Cobol, offset is 0:
  
  ```
  STORAGE ( H'162F0', 20 ) = 'clear that string ' ; 
  ```

- For Assembler, update the storage pointed by address \texttt{0002CD0}, starting at offset 16 for 4 bytes, and the offset is specified as a hex constant:
  
  ```
  STORAGE ( X'0002CD0', X'10', 4 ) = 5 ; 
  ```

---

**switch command (C and C++)**

The switch command enables you to transfer control to different commands within the switch body, depending on the value of the switch expression. The switch, case, and default keywords must be lowercase and cannot be abbreviated.

```python
>>>switch---(--expression--)---(---switch_body---)---<---;---
```
switch_body:

<table>
<thead>
<tr>
<th>default_clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>case_clause</td>
</tr>
</tbody>
</table>

case_clause:

<table>
<thead>
<tr>
<th>case_clause</th>
</tr>
</thead>
</table>

default_clause:

<table>
<thead>
<tr>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
</tr>
</tbody>
</table>

expression
A valid Debug Tool C expression.

case_expression
A valid character or optionally signed integer constant.

command
A valid Debug Tool command.

The value of the switch expression is compared with the value of the expression in each case clause. If a matching value is found, control is passed to the command in the case clause that contains the matching value. If a matching value is not found and a default clause appears anywhere in the switch body, control is passed to the command in the default clause. Otherwise, control is passed to the command following the switch body.

If control passes to a command in the switch body, control does not pass from the switch body until a break command is encountered or the last command in the switch body is performed.

Usage notes
- Declarations are not allowed within a switch command.
- The switch command does not end with a semicolon. A semicolon after the closing brace is treated as a Null command.
- Although this command is similar to the switch statement in C, it is subject to Debug Tool restrictions on expressions.
- Duplicate case_expression values are not supported.
- You cannot use the switch command while you replay recorded statements by using the PLAYBACK commands.

Examples
The following switch command contains several case clauses and one default clause. Each clause contains a function call and a break command. The break commands prevent control from passing down through subsequent commands in the switch body.

If key has the value '/', the switch command calls the function divide. On return, control passes to the command following the switch body.

```c
char key;
printf("Enter an arithmetic operator\n");
scanf("%c",&key);
switch (key) {
    case '+':
        add();
        LIST (key);
        break;
    case '-':
        subtract();
        LIST (key);
        break;
    case '*':
        multiply();
        LIST (key);
        break;
    case '/':
        divide();
        LIST (key);
        break;
    default:
        printf("Invalid key\n");
        break;
}
```

In the following example, break commands are not present. If the value of c is equal to 'A', all 3 counters are incremented. If the value of c is equal to 'a', lettera and total are increased. Only total is increased if c is not equal to 'A' or 'a'.

```c
char text[100];
int capa, i, lettera, total;
for (i=0; i < sizeof(text); i++) {
    switch (text[i]) {
        case 'A':
            capa++;
        case 'a':
            lettera++;
        default:
            total++;
    }
}
```

---

**SYSTEM command (z/OS)**

The SYSTEM command lets you issue TSO commands during a Debug Tool session. The SYSTEM keyword can only be abbreviated as SYS.

```
SYS system_command:
```
A valid TSO system command or CLIST name that does not require a parameter.

Usage notes

- No parameters can be specified as part of the system command or CLIST invocation. To execute noninteractively when parameters are required, you must enter the complete invocation in a CLIST and then use a TSO or SYSTEM command to call that CLIST (without parameters).
- You cannot introduce a new Debug Tool session using the SYSTEM command.
- When operating interactively in TSO, there is no provision for entering a mode where commands are accepted repeatedly; however, it is possible to write your own such iterative sequence in a CLIST.
- You cannot issue CICS commands using SYSTEM.

Examples

- List all the data sets in the user catalog.
  
  ```
  SYSTEM LISTCAT;
  ```

- Temporarily places you in ISPF mode.
  
  ```
  SYSTEM PDF;
  ```

Refer to the following sections for more information related to the material discussed in this section.

Related references

"TSO command (z/OS)" on page 231

TRIGGER command

The TRIGGER command raises the specified AT-condition in Debug Tool, or it raises the specified programming language condition in your program.
storage_clause:

\[ \text{STORAGE} \left( \text{address, length} \right) \]

condition

A valid condition or exception. Depending on the current programming language setting, this code can be any one of the following types of codes:

- A Language Environment symbolic feedback code
- A language-oriented keyword or code
- When an application runs without the Language Environment run time, one of the ABEND codes shown below.

If no active condition handler exists for the specified condition, the default condition handler can cause the program to end prematurely.
Following are the C condition constants; they must be uppercase and not abbreviated.

<table>
<thead>
<tr>
<th>Condition Constant</th>
<th>Condition Constant</th>
<th>Condition Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGABND</td>
<td>SIGILL</td>
<td>SIGTERM</td>
</tr>
<tr>
<td>SIGABRT</td>
<td>SIGINT</td>
<td>SIGUSR1</td>
</tr>
<tr>
<td>SIGFPE</td>
<td>SIGIOERR</td>
<td>SIGUSR2</td>
</tr>
<tr>
<td></td>
<td>SIGSEGV</td>
<td></td>
</tr>
</tbody>
</table>

There are no COBOL condition constants. Instead, an Language Environment symbolic feedback code must be used, for example, CEE347.

PL/I condition constants can be used; for syntax and acceptable abbreviations see the ON command.

When you are running without the Language Environment run time, use one of the following codes:

- Codes Sxxx and Uxxx to represent MVS System and User ABENDs. In this case the xxx is three hexadecimal digits representing the ABEND code.
- Any four-character string to represent a CICS ABEND code.

**cu_spec**
A valid compile unit specification.

**entry_name**
A valid external entry point name constant or zero (0); however, 0 can only be specified if the current programming language setting is C or PL/I.

**reference**
A valid Debug Tool reference in the current programming language.

**%STORAGE**
A built-in function that provides an alternative way to select an AT CHANGE subject.

**address**
The starting address of storage to be watched for changes. This must be a hex constant: 0x in C, H in COBOL (using either double (”) or single (‘) quotes), a PX constant in PL/I, or X in assembler or disassembly.

**length**
The number of bytes of storage being watched for changes. This must be a positive integer constant. The default value is 1.

**load_spec**
A valid load module specification.

**block_spec**
A valid block specification.

**statement_label**
A valid source label constant.

**stmt_id_spec**
A valid statement id specification.

**Usage notes**
- AT TERMINATION cannot be raised by the TRIGGER command.
- An enclave cannot be stopped by the TRIGGER command.
- If you are replaying recorded statements by using the PLAYBACK commands, you cannot use the TRIGGER command.
Examples

In the first example, note the following differences

- Triggering a breakpoint (TRIGGER AT OCCURRENCE CEE347), which performs Debug Tool commands associated with the breakpoint. The condition is not raised.
- Triggering a condition (TRIGGER CEE347), which raises the condition and causes a corresponding system action. A corresponding system action can be a condition handler.
- Perform the commands in the AT OCCURRENCE CEE347 breakpoint (the CEE347 condition is not raised). The current programming language setting is COBOL.

```
AT OCCURRENCE CEE347 PERFORM
  SET ix TO 5;
END-PERFORM;
```

```
TRIGGER AT OCCURRENCE CEE347; /* SET ix TO 5 is executed */
```

- Raise the SIGTERM condition in your program. The current programming language setting is C.

```
TRIGGER SIGTERM;
```

- A previously defined STATEMENT breakpoint (for line 13) is triggered.

```
AT 13 LIST "at 13";
TRIGGER AT 13;
/* "at 13" will be the echoed output here */
```

- Assume the following breakpoints exist in a program:

```
AT CHANGE x LIST TITLED (x); AT STATEMENT 10;
```

If Debug Tool is started for the STATEMENT breakpoint and you want to trigger the commands associated with the AT CHANGE breakpoint, enter:

```
TRIGGER AT CHANGE x;
```

Debug Tool displays the value of x.

Refer to the following sections for more information related to the material discussed in this section.

Related references

z/OS Language Environment Programming Guide

Related references

- “ON command (PL/I)” on page 149
- “cu_spec” on page 13
- “references” on page 15
- “load_spec” on page 15
- “block_spec” on page 11
- “statement_label” on page 17
- “statement_id_range and stmt_id_spec” on page 16

TSO command (z/OS)

The TSO command lets you issue TSO commands during a Debug Tool session and is valid only in a TSO environment. The TSO keyword cannot be abbreviated.

```TSO—tso_command—;```
tso_command

A valid TSO system command or CLIST name that does not require a parameter.

Usage notes

- TSO is synonymous to SYSTEM.

Example

List all the data sets in the user catalog.

TSO LISTCAT;

Refer to the following sections for more information related to the material discussed in this section.

Related references

“SYSTEM command (z/OS)” on page 227

USE command

The USE command causes the Debug Tool commands in the specified file or data set to be either performed or syntax checked. This file can be a log file from a previous session. The specified file or data set can itself contain another USE command. The maximum number of USE files open at any time is limited to eight. The USE keyword cannot be abbreviated.

USE ddname dsname;

ddname

A valid ddname in z/OS.

dsnname

A z/OS data set containing the Debug Tool commands to be performed. If dsname is not enclosed in single quotes, Debug Tool assumes it is a partially-qualified data set name and the user ID is prefixed to form the fully-qualified data set name.

Usage notes

- To check the syntax of the commands in a USE file:
  1. Set the EXECUTE setting to 0FF.
  2. Enter a USE command for the file.
- Commands read from a USE file are logged as comments.
- The log file can serve as a USE file in a subsequent Debug Tool session.
- Recursive calls are not allowed; that is, a commands file cannot be used if it is already active. This includes the primary commands and preferences files. If another invocation of Debug Tool occurs during the execution of a USE file (for example, if a condition is raised while executing a command from a USE file), the USE file is not used for command input until control returns from the condition.
- The USE file is closed when the end of the file is reached.
- If a nonreturning command (such as GO) is performed from a USE file, the action taken (as far as closing the USE file) depends on certain things:
  - If the USE file was called directly or indirectly from the primary commands file or preferences file, it has the same characteristics as the primary
commands file or preferences file. That is, it "keeps its place" and the next time Debug Tool requests a command, it reads from the USE file where it left off.

- If the USE file was not called directly or indirectly from the primary commands file or preferences file, the rest of the USE file and the file that called the USE file is skipped.

- If the end of the USE file is reached without encountering a QUIT command, Debug Tool returns to the command source where the USE command was issued. This can be the terminal, a command string, or another commands file.

- A USE file takes on the aspects of whatever command source issued the USE command, relative to its behavior when a GO, GOTO, or STEP is executed. When called from the primary commands file, it continues with its next sequential command at the next breakpoint. If it is called from any other command sequence, the GO, GOTO, or STEP causes any remaining commands in the USE file to be discarded.

Examples

- Perform the Debug Tool commands in the z/OS data set USERID.COMMANDS.FILE. The data set must first be allocated with, for example, ALLOC FI(MYCMDS) DA('USERID.COMMANDS.FILE').
  
  USE MYCMDS;

  Alternatively, perform the commands in the z/OS data set USERID.COMMANDS.FILE.
  USE COMMANDS.FILE

- On z/OS, perform the Debug Tool commands in the partitioned data set member USERID.PDS(CMDS).
  USE PDS(CMDS)

- For CICS, perform Debug Tool commands in the fully-qualified data set TS64081.USE.FILE.
  USE 'TS64081.USE.FILE';

In addition to using sequential files, you can perform Debug Tool commands using partitioned data sets.

  USE 'userid.thing.file(usefile)'

---

**while command (C and C++)**

The while command enables you to repeatedly perform the body of a loop until the specified condition is no longer met or evaluates to false. The while keyword must be lowercase and cannot be abbreviated.

```plaintext
  while(expression) command;
```

**expression**

A valid Debug Tool C expression.

**command**

A valid Debug Tool command.

The expression is evaluated to determine whether the body of the loop should be performed. If the expression evaluates to false, the body of the loop never executes. Otherwise, the body does execute. After the body has been performed,
control is given once again to the evaluation of the expression. Further execution of the action depends on the value of the condition.

A break command can cause the execution of a while command to end, even when the condition does not evaluate to false.

**Usage notes**
- If you are replaying recorded statements by using the PLAYBACK commands, then you cannot use the while command.

**Examples**
- List the values of \( x \) starting at 3 and ending at 9, in increments of 2.

```plaintext
x = 1;
while (x += 2, x < 10)
    LIST x;
```
- While \(--\text{index}\) is greater than or equal to zero (0), triple the value of the expression \( \text{item}[\text{index}] \).

```plaintext
while (--\text{index} >= 0) {
    \text{item}[\text{index}] *= 3;
    printf("\text{item}[\text{index}] = \%d\n", \text{index}, \text{item}[\text{index}]);
}
```

## WINDOW command (full-screen mode)

The WINDOW command provides window manipulation functions. WINDOW commands can be made immediately effective with the IMMEDIATE command. The cursor-sensitive form is most useful when assigned to a PF key. The WINDOW keyword is optional.

The following table summarizes the forms of the WINDOW command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;WINDOW CLOSE&quot;</td>
<td>Closes the specified window in the Debug Tool full-screen session panel.</td>
</tr>
<tr>
<td>&quot;WINDOW OPEN&quot;</td>
<td>Opens a previously-closed window in the Debug Tool full-screen session panel.</td>
</tr>
<tr>
<td>&quot;WINDOW SIZE&quot;</td>
<td>Controls the relative size of currently visible windows in the Debug Tool full-screen session panel.</td>
</tr>
<tr>
<td>&quot;WINDOW ZOOM&quot;</td>
<td>Expands the indicated window to fill the entire screen.</td>
</tr>
</tbody>
</table>

**Usage notes**
- If no operand is specified and the cursor is on the command line, then the default window id set by SET DEFAULT WINDOW is used (if it is open, otherwise the precedence is SOURCE, LOG, MONITOR).

## WINDOW CLOSE

Closes the specified window in the Debug Tool full-screen session panel. The remaining open windows expand to fill the remainder of the screen. Closing a window does not effect the contents of that window. For example, closing the monitor window does not stop the monitoring of variable values assigned by the LIST MONITOR command.

If there is only one window visible, WINDOW CLOSE is invalid.
CURSOR
Selects the window where the cursor is currently positioned unless on the command line.

LOG
Selects the session log window.

MONITOR
Selects the monitor window.

SOURCE
Selects the source listing window.

Example

Close the window containing the cursor.
WINDOW CLOSE CURSOR;

WINDOW OPEN
Opens a previously-closed window in the Debug Tool full-screen session panel. Any existing windows are resized according to the configuration selected with the PANEL LAYOUT command.

If the OPEN command is issued without an operand, Debug Tool opens the last closed window.

LOG
Selects the session log window.

MONITOR
Selects the monitor window.

SOURCE
Selects the source listing window.

Example

Open the monitor window.
WINDOW OPEN MONITOR;

WINDOW SIZE
Controls the relative size of currently visible windows in the Debug Tool full-screen session panel.
integer
Specifies the number of rows or columns, as appropriate for the selected
window and the current window configuration.

CURSOR
Selects the window where the cursor is currently positioned unless on the
command line. The cursor form of WINDOW SIZE applies to that window if
integer is specified. Otherwise, it redraws the configuration of windows so that
the intersection of the windows is at the cursor, or if the configuration does not
have a common intersection, so that the nearest border is at the cursor.

LOG
Selects the session log window.

MONITOR
Selects the monitor window.

SOURCE
Selects the source listing window.

Usage notes
• You cannot use WINDOW SIZE if a window is zoomed or if there is only one
  window open.
• Each window in any configuration has only one adjustable dimension:
  – If one or more windows are as wide as the screen:
    - The number of rows is adjustable for each window as wide as the screen
    - The number of columns is adjustable for the remaining windows
  – If one or more windows are as high as the screen:
    - The number of columns is adjustable for each window as high as the
      screen
    - The number of rows is adjustable for the remaining windows

Examples
• Adjust the size of the Source window to 15 rows.
  WINDOW SIZE 15 SOURCE;
• Adjust the size of the window where the cursor is currently positioned to 20
  rows.
  SIZE 20 CURSOR;

WINDOW ZOOM
Expands the indicated window to fill the entire screen or restores the screen to the
currently defined window configuration.
CURSOR
Selects the window where the cursor is currently positioned unless on the command line.

LOG
Selects the session log window.

MONITOR
Selects the monitor window.

SOURCE
Selects the source listing window.

If the selected window is currently zoomed, the zoom mode is toggled. That is, the currently defined window configuration is restored.

Usage note
The WINDOW ZOOM command is not logged.

Example
Expand the log window.
WINDOW ZOOM LOG;
# Chapter 6. Debug Tool built-in functions

Debug Tool provides you with several built-in functions which allow you to manipulate variables. All Debug Tool built-in function names begin with a percent sign (%).

The table below summarizes the Debug Tool built-in functions. Unless otherwise indicated, the functions can be used with all supported languages.

<table>
<thead>
<tr>
<th>Debug Tool built-in function</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>`%DEC (assembler, disassembly, and OS/VS COBOL)&quot;</td>
<td>Decimal value of an operand.</td>
</tr>
<tr>
<td>`%GENERATION (PL/I)&quot; on page 240</td>
<td>A specific generation of a controlled variable</td>
</tr>
<tr>
<td>`%HEX&quot; on page 240</td>
<td>Hexadecimal value of an operand</td>
</tr>
<tr>
<td>`%INSTANCES (C, C++, and PL/I)&quot; on page 241</td>
<td>Maximum value of %RECURSION for a block</td>
</tr>
<tr>
<td>`%RECURSION (C, C++, and PL/I)&quot; on page 242</td>
<td>An automatic variable or a parameter in a specific instance of a recursive procedure</td>
</tr>
<tr>
<td>%STORAGE</td>
<td>Changed/unchanged status of a range of bytes in storage; this function can only be used in an &quot;AT CHANGE&quot; on page 45 command.</td>
</tr>
<tr>
<td>You cannot use the %STORAGE built-in function while you replay recorded steps.</td>
<td></td>
</tr>
<tr>
<td>`%WHERE (assembler, disassembly, and OS/VS COBOL)&quot; on page 243</td>
<td>A string indicating the address of the operand.</td>
</tr>
</tbody>
</table>

### `%DEC (assembler, disassembly, and OS/VS COBOL)"

Returns the decimal value of an operand.

```plaintext
>>> %DEC(expression);
```

**expression**

A valid assembler, disassembly, or OS/VS COBOL expression.

### Examples

Assuming register R1 contains the value 14, to display the value of the expression R1+2 in decimal, enter the following command:

```plaintext
LIST %DEC(R1+2);
```

The Log window displays the value 16.

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

"LIST expression" on page 129
%GENERATION (PL/I)

Returns a specific generation of a controlled variable in your program.

\[ %\text{GENERATION}(\text{reference}, \text{expression}) \]

- **reference**: A controlled variable.
- **expression**: The generation number \( n \) of a controlled variable \( x \), where:
  \[ 1 \leq n \leq \text{ALLOCATION}(x) \]

To return the oldest instance of \( x \), specify:
\[ %\text{GENERATION}(x, 1) \]

To return the most recent instance of \( x \), specify:
\[ %\text{GENERATION}(x, \text{ALLOCATION}(x)) \]

**Usage notes**
- You cannot use the \%GENERATION built-in function while you replay recorded steps.
- If you want to use the \%GENERATION built-in function while debugging Enterprise PL/I programs, you must do the following task:

Refer to the following sections for more information related to the material discussed in this section.

**Related tasks**
- [Debug Tool User’s Guide](#)

---

%HEX

Returns the hexadecimal value of an operand.

\[ %\text{HEX}(\text{reference}) \]

- **reference**: One of the following:
  - COBOL or PL/I reference
  - C or C++ lvalue
  - assembler, disassembly, or OS/VS COBOL expression

**Examples**

C and C++: To display the internal representation of the packed decimal variable \( z\text{var1} \) whose external representation is 235, enter the following command.

\[ \text{LIST } %\text{HEX}(z\text{var1}) ; \]

The Log window displays the hexadecimal string 235C.
COBOL: To display the external representation of the packed decimal pvar3, defined as PIC 9(9), from 1234 as its hexadecimal (or internal) equivalent, enter the following command.

LIST %HEX (pvar3);

The Log window displays the hexadecimal string 01234F.

Refer to the following sections for more information related to the material discussed in this section.

Related references
“LIST expression” on page 129

%INSTANCES (C, C++, and PL/I)

Returns the maximum value of %RECURSION (the most recent recursion number) for a given block.

%INSTANCES (reference); ........................................

reference
An automatic variable or a subroutine parameter. If necessary, you can use qualification to specify the variable.

%INSTANCES can be used like a Debug Tool variable.

Usage notes

Debug Tool does not support the %INSTANCES built-in function for Enterprise PL/I programs.

You cannot use the %INSTANCES built-in function while you replay recorded steps.

Examples

C and C++:

• %INSTANCES and %RECURSION can be used together to determine the number of times a function is recursively called. They can also give you access to an automatic variable or parameter in a specific instance of a recursive procedure. Assume, for example, your program contains the following statements.

```c
int RecFn(unsigned int i) {
  if (i == 0) {
    _ctest("*");
  }
}
```

At this point, the _ctest() call gives control to Debug Tool, and you are prompted for commands. Enter the following command.

LIST %INSTANCES(i);

The Log window displays the number of times that RecFn() was interactively called.

To display the value of ‘i’ at the first call of RecFn(), enter the following command.

%RECURSION(i, 1);

• If necessary, you can use qualification to specify the parameter. For example, if the current point of execution is in %block2, and %block3 is a recursive function
containing the variable x, you can write an expression using x by qualifying the variable, as shown in the example below.

%RECURSION(main::%block3::x, %INSTANCES(main::%block3::x, y)

- The following example gets a line of input from stdin using the C library routine gets.
  ```c
  char line[100];
  char *result;
  result = gets(line);
  ```

- The following example removes a file and checks for an error, issuing a message if an error occurs.
  ```c
  int result;
  result = remove("mayfile.dat");
  if (result != 0)
    perror("could not delete file");
  ```

- Debug Tool performs the necessary conversions when a call to a library function is made. The cast operator can be used. In the following example, the integer 2 is converted to a double, which is the required argument type for sqrt.
  ```c
  double sqrtval;
  sqrtval = sqrt(2);
  ```

- Nested function calls can be performed, as shown in the example below.
  ```c
  printf("absolute value is %d\n", abs(-55));
  ```

- C library variables such as errno and stdout can be used, as shown in the example below.
  ```c
  fprintf(stdout, "value of errno is %d\n", errno);
  ```

Refer to the following sections for more information related to the material discussed in this section.

Related references

%RECURSION (C, C++, and PL/I)

%RECURSION (C, C++, and PL/I)

Returns a specific instance of an automatic variable or a parameter in a recursive procedure.

```c
%RECURSION(---reference---,---expression---);
```  

reference

An automatic variable or a subroutine parameter. If necessary, you can use qualification to specify the variable.

expression

The recursion number of the variable or parameter.

To return the oldest recursion of x, specify:

```c
%RECURSION(x,1)
```  

To return the most recent recursion of x, specify:

```c
%RECURSION(x,%INSTANCES(x))
```  

Usage notes

- The higher the value of the expression, the more recent the generation of the variable Debug Tool references.
- %RECURSION can be used like a Debug Tool variable.
• You cannot use the %RECURSION built-in function while you replay recorded steps.
• You cannot use the %RECURSION built-in function while you debug Enterprise PL/I programs.

Refer to the following sections for more information related to the material discussed in this section.

Related references

"%INSTANCES (C, C++, and PL/I)" on page 241

%WHERE (assembler, disassembly, and OS/VS COBOL)

Returns a string that describes the named area (if any) whose address is specified as the operand. %WHERE can be used only as the outermost expression in the LIST command.

```plaintext
%WHERE(expression);
```

expression

An expression that evaluates to a hexadecimal value that is one to four bytes in length.

The following rules are used to evaluate the value of the expression, in the order listed:

1. If the expression value is less than 4096, a decimal number is returned.
2. If the expression value is an address within any known compile unit, the name of the compile unit with a hexadecimal offset is returned.
3. If the expression value is within 4095 bytes of the value in a general register and a USING is in effect for that register, the name of the DSECT that corresponds to the USING instruction with a hexadecimal offset is returned.
4. A hexadecimal number representing the expression value is returned.

Although this function can be used only within an assembler, disassembly, or OS/VS COBOL compile unit, the expression can evaluate to a compile unit in any language.

Usage note

You cannot nest %WHERE into another expression. For example, the following command is not valid:

```plaintext
LIST %WHERE('14B0')||'ABC'
```

Examples

• Assuming that CSECT ROUTINE1 is located at address X'1BC0400', the following command returns "ROUTINE1+X'2A'":

```plaintext
LIST %WHERE(X'1BC042A')
```

• Assume that register R3 contains X'1C4A0' and that the program is stopped at the instruction highlighted in the following lines of code:

```plaintext
USING DATA1,R3
...
SLR R0,R0
...
DATA1 DSECT ,
...
```

Chapter 6. Debug Tool built-in functions 243
The command LIST %WHERE(X'1C4B4') returns DATA1+X'14'.
- The command LIST %WHERE(X'100') returns 256.
- If the expression cannot be resolved to a known location, the command LIST %WHERE(X'1B5C4') returns X'1B5C4'.
Chapter 7. Debug Tool variables

Debug Tool reserves several variables for its own information. These Debug Tool variable names begin with a percent sign (%), to distinguish them from program variables. You can access Debug Tool variables while testing programs in any supported language.

You can use all Debug Tool variables in expressions. Additionally, the variables %EPRn, %FPRn, %GPRn, and %LPRn (representing the types of registers) can be modified, as shown in the COBOL example below.

```cobol
MOVE name_table TO %GPR5;
```

**Note:** Use caution when assigning new values to registers. Important program information can be lost. Do not modify the base register.

To display the value of a Debug Tool variable, use the LIST command, as shown in the example below.

```cobol
LIST %GPR15
```

The table below summarizes the Debug Tool variables.

<table>
<thead>
<tr>
<th>Debug Tool variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%ADDRESS on page 246</td>
<td>Address of the location where your program was interrupted</td>
</tr>
<tr>
<td>%AMODE on page 246</td>
<td>Current AMODE of the suspended program</td>
</tr>
<tr>
<td>%BLOCK on page 247</td>
<td>Name of the current block</td>
</tr>
<tr>
<td>%CAAADDRESS on page 247</td>
<td>Address of the CAA control block associated with the suspended program</td>
</tr>
<tr>
<td>%CONDITION on page 247</td>
<td>Name or number of the condition when Debug Tool is entered because of an AT OCCURRENCE</td>
</tr>
<tr>
<td>%COUNTRY on page 248</td>
<td>Current country code</td>
</tr>
<tr>
<td>%CU or %PROGRAM on page 248</td>
<td>Name of the primary entry point of the current compile unit</td>
</tr>
<tr>
<td>%EPA on page 248</td>
<td>Address of the primary entry point in the current compile unit</td>
</tr>
<tr>
<td>%EPRn on page 248</td>
<td>(C and C++ and PL/I only) Extended-precision floating-point registers</td>
</tr>
<tr>
<td>%FPRn on page 249</td>
<td>Single-precision floating-point registers</td>
</tr>
<tr>
<td>%GPRn on page 249</td>
<td>General-purpose registers at the point of interruption in a program</td>
</tr>
<tr>
<td>%HARDWARE on page 250</td>
<td>Type of hardware where the application is running</td>
</tr>
<tr>
<td>%LINE or %STATEMENT on page 251</td>
<td>Current source line number</td>
</tr>
<tr>
<td>%LOAD on page 251</td>
<td>Name of the load module of the current program, or an asterisk (*)</td>
</tr>
<tr>
<td>%LPRn on page 252</td>
<td>Double-precision floating-point registers</td>
</tr>
<tr>
<td>%UNLANGUAGE on page 252</td>
<td>National language currently in use</td>
</tr>
<tr>
<td>Debug Tool variable</td>
<td>Value</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------</td>
</tr>
<tr>
<td>%PATHCODE&quot; on page 252</td>
<td>Integer identifying the type of change occurring when the program flow reaches a point of discontinuity, and the path condition is raised</td>
</tr>
<tr>
<td>%PLANGUAGE&quot; on page 253</td>
<td>Current programming language</td>
</tr>
<tr>
<td>%PROGRAM</td>
<td>Equivalent to %CU</td>
</tr>
<tr>
<td>%RC&quot; on page 253</td>
<td>Return code from the most recent Debug Tool command</td>
</tr>
<tr>
<td>%RUNMODE&quot; on page 253</td>
<td>String identifying the presentation mode of Debug Tool</td>
</tr>
<tr>
<td>%STATEMENT</td>
<td>Equivalent to %LINE</td>
</tr>
<tr>
<td>%SUBSYSTEM&quot; on page 254</td>
<td>Name of the underlying subsystem, if any, where the program is running</td>
</tr>
<tr>
<td>%SYSTEM&quot; on page 255</td>
<td>Name of the operating system supporting the program</td>
</tr>
</tbody>
</table>

You can access Debug Tool variables even when they have no intrinsic meaning in your operating system or language.

Refer to the following sections for more information related to the material discussed in this section.

**Related references**
- "Attributes of Debug Tool variables in different languages" on page 255

**%ADDRESS**

Contains the address of the location where the program has been interrupted.

**Attributes**

C and C++: void *

COBOL: USAGE POINTER

**For COBOL only:**

- You can use the OFFSET table in the compiler listing to determine statement numbers. To determine the offset of the current statement, subtract %EPA (the address of the primary entry point) from %ADDRESS, as shown in the example below.

  ```
  LIST %ADDRESS - %EPA
  ```

- %ADDRESS might not locate a statement in your COBOL program in all instances. When an error occurs outside of the program, in some instances, %ADDRESS contains the actual interrupt address. This occurs only if Debug Tool is unable to locate the last statement that was executed before control left the program.

**%AMODE**

Contains the current AMODE of the suspended program: 24 or 31. For COBOL programs, the value is always 31.

**Attributes**

C and C++: void *

COBOL: PICTURE S9(4) USAGE COMP
Contains the name of the current block. The block name might not be unique within a compile unit.

To display the name of the current block, use one of the following commands:
- DESCRIBE PROGRAM;
- LIST %BLOCK;

To change the current block, use the SET QUALIFY command.

**Attributes**

C and C++: unsigned char[]

COBOL: PICTURE X(j)

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

- "DESCRIBE command" on page 93
- "LIST expression" on page 129
- "SET QUALIFY" on page 203

Contains the address of the Language Environment CAA control block associated with the suspended program. When you are running without the Language Environment run time, the value NONE is returned.

**Attributes**

C and C++: void *

COBOL: USAGE POINTER

Contains the name or number of the condition when Debug Tool is entered because of an AT OCCURRENCE.

COBOL: PICTURE X(j)

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

- "AT OCCURRENCE" on page 60
%COUNTRY

Contains the current country code.

Attributes

C and C++: unsigned char[]

COBOL: PICTURE X(j)

%CU or %PROGRAM

Contains the name of the primary entry point of the current compile unit.

To change the current compile unit, use the SET QUALIFY command.

%CU is equivalent to %PROGRAM.

Attributes

C and C++: unsigned char[]

COBOL: PICTURE X(j)

Refer to the following sections for more information related to the material discussed in this section.

Related references

SET QUALIFY on page 203

%EPA

Contains the address of the primary entry point of the currently interrupted program. If you are replaying recorded statements, the %EPA variable contains the name of the current location.

Attributes

C and C++: void *

COBOL: USAGE POINTER

Usage note

The value of %EPA is valid only in programs that adhere to standard linkage conventions for R13, R14, and R15.

%EPRn

(%EPR0 and %EPR4; if the application supports IEEE Floating Point Arithmetic, you can also use %EPR1, %EPR5, %EPR8, %EPR9, %EPR12, and %EPR13.)

Represent the extended-precision floating-point registers.

To modify one of these registers, assign a value to the associated %EPRn variable.
%EPRn cannot be used as the target of an assignment while debugging Enterprise PL/I programs.

You cannot use the %EPRn variable while you are replaying recorded statements.

**Attributes**

C and C++: long double

COBOL: these variables are not defined for COBOL programs

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

- "Expression command (C and C++)" on page 104
- "MOVE command (COBOL)" on page 145
- "Assignment command (PL/I)" on page 36

---

**%FPRn**

(%FPR0, %FPR2, %FPR4, and %FPR6; if the application supports IEEE Floating Point Arithmetic, you can also use %FPR1, %FPR3, %FPR5, %FPR7, %FPR9, %FPR11, %FPR13, and %FPR15.)

Represent single-precision floating-point registers.

To modify one of these registers, assign a value to the associated %FPRn variable.

%FPRn cannot be used as the target of an assignment while debugging Enterprise PL/I programs.

**Attributes**

C and C++: float

COBOL: USAGE COMP-1

Refer to the following sections for more information related to the material discussed in this section.

**Related references**

- "Expression command (C and C++)" on page 104
- "MOVE command (COBOL)" on page 145
- "Assignment command (PL/I)" on page 36

---

**%GPRn**

(%GPR0 to %GPR15.)

Represent general-purpose registers at the point of interruption in a program.

To modify one of these registers, assign a value to the associated %GPRn variable.

**Attributes**

C and C++: signed int
COBOL: PICTURE S9(9)
PL/I: FIXED BINARY(31)

Usage notes
- If you modify a %GPRn register, the change is reflected when you resume program execution.
- Do not modify base registers.
- Although assigning new values to variables %GPR12 and %GPR13 does not result in an error, when any subsequent action is taken the newly set values are reset to their previous values.
- %GPRn cannot be used as the target of an assignment while debugging Enterprise PL/I programs.
- In disassembly view, you can replace Rn with %GPRn.
- In assembler, you can replace %GPRn with any symbol defined in the program and whose first use in the program was as a register. You can also use any of the R0, R1, ..., R15 symbols that were not defined in the programs.
- For Enterprise PL/I, if you display the value of %GPRn by using the LIST command, the result is displayed in FIXED BINARY(31) format.
- For PL/I for MVS & VM, if you display the value of %GPRn by using the LIST command, the result is displayed in PX (hex pointer) format.

C and C++ only:
- If you modify the value of %GPR3, then the base register in the program can be lost.

Examples
COBOL:
MOVE name_table TO %GPR15;

C and C++:
MOVE name_table TO %GPR15;

Refer to the following sections for more information related to the material discussed in this section.

Related references
- "Expression command (C and C++)" on page 104
- "MOVE command (COBOL)" on page 145
- "Assignment command (PL/I)" on page 36

%HARDWARE

Identifies the type of hardware where the application program is running. A possible value is: 370/ESA.

Attributes
C and C++: unsigned char[]
COBOL: PICTURE X(j)
%LINE or %STATEMENT

Contains the current line number.

If the current statement is not the first statement on the line, then the line number is followed by a period and the number of the statement with the line. For example, if %LINE = 4.3, then the current statement is the third statement on the fourth source line.

If the program is at the entry or exit of a block, then %LINE contains ENTRY or EXIT, respectively.

If the line number cannot be determined (for example, a run-time line number does not exist or the address where the program is interrupted is not in the program), then %LINE contains an asterisk (*).

For COBOL, %LINE does not return a relative verb (within the line) for labels.

%LINE is equivalent to %STATEMENT.

In the disassembly view, %LINE and %STATEMENT are not supported.

Attributes

C and C++: unsigned char[]

COBOL: PICTURE X(j)

%LOAD

If the current program is part of a fetched or called load module, then %LOAD contains the name of the load module.

If the current program is in the load module that was initially loaded, then %LOAD contains an asterisk (*).

Debug Tool uses the value of %LOAD when you make an unqualified reference to a program or variable.

To change the current load module, use the SET QUALIFY command.

When the Dynamic Debug facility is deactivated (by entering the SET DYNDEBUG OFF command) or SVC screening is disabled, Debug Tool does not recognize load modules that have been loaded by the MVS LOAD service.

Attributes

C and C++: unsigned char[]

COBOL: PICTURE X(j)

Refer to the following sections for more information related to the material discussed in this section.

2. See the Debug Tool Customization Guide for instructions on how to control SVC screening.
Related references
“SET QUALIFY” on page 203

%%LPRn

(%LPR0, %LPR2, %LPR4, and %LPR6; if the application supports IEEE Floating Point Arithmetic, you can also use %LPR1, %LPR3, %LPR5, %LPR7, %LPR9, %LPR11, %LPR13, and %LPR15.)

Represent the double-precision floating-point registers.

To modify one of these registers, assign a value to the associated %LPRn variable.

%LPRn cannot be used as the target of an assignment while debugging Enterprise PL/I programs.

Attributes

C and C++: double

COBOL: USAGE COMP-2

Refer to the following sections for more information related to the material discussed in this section.

Related references
“Expression command (C and C++)” on page 104
“MOVE command (COBOL)” on page 145
“Assignment command (PL/I)” on page 36

%%NLANGUAGE

Indicates the national language currently in use: ENGLISH, UENGLISH, JAPANESE, or KOREAN.

Attributes

C and C++: unsigned char[]

COBOL: PICTURE X(j)

%%PATHCODE

Contains an integer value that identifies the kind of change occurring when the path of program execution has reached a point of discontinuity and the path condition is raised.

The possible values vary according to the language of your program.

If you are replaying recorded statements, you cannot use the %PATHCODE variable.

Attributes

C and C++: signed short int[]
**%PLANGUAGE**

Indicates the programming language currently in use.

%PLANGUAGE returns C for both C and C++.

**Attributes**

C and C++: unsigned char[]

COBOL: PICTURE X(j)

---

**%PROGRAM**

Contains the name of the primary entry point of the current program.

%PROGRAM is equivalent to %CU.

**Attributes**

C and C++: unsigned char[]

COBOL: PICTURE X(j)

---

**%RC**

Contains a return code whenever a Debug Tool command ends.

%RC initially has a value of zero unless the log file cannot be opened, in which case it has a value of -1.

**Note:** The %RC return code is a Debug Tool variable. It is not related to the return code that can be found in Register 15.

**Attributes**

C and C++: signed short int

COBOL: PICTURE S9(4) USAGE COMP

---

**%RUNMODE**

Contains a string identifying the presentation mode of Debug Tool. The possible values are listed below.

- LINE
- SCREEN
- BATCH

**Attributes**

C and C++: unsigned char[]

COBOL: PICTURE X(j)
%Rn

(%R0 to %R15)

Represent the general-purpose registers for the assembler or disassembly CU to which you are currently qualified. These symbols are not valid in a CU in any other language. In addition, these symbols are undefined in assembler and disassembly CUs that are not currently active or for which the applicable general-purpose registers cannot be located. Registers can be located for active assembler and disassembly CUs only if:

- The CU was in control when the user program was suspended, or
- The CU was active in the call chain above the CU that was active when the user program was suspended and all programs in the call chain use standard linkage conventions in relation to R13, R14, and R15 and save all registers in a chained save area.

To modify one of these registers, assign a value to the associated %Rn variable.

Usage notes

- If you modify a %Rn register, the change is reflected when you resume program execution.
- The use of these symbols is equivalent to the use of any register symbols defined in your program.
- The %Rn symbols differ from the %GPRn symbols in that %GPRn represents the value that was actually in the hardware general-purpose register when your user program was suspended, but %Rn represents the value associated in the assembler or disassembly CU to which you are currently qualified. If you are currently qualified to the CU that was active when your user program was suspended, %Rn and %GPRn are identical. However, if you are currently qualified to a CU that was in the call chain but was not the CU that was active when your program was suspended, %Rn and %GPRn will be different. If you are qualified to a CU that was not active when your program was suspended, %Rn is undefined.

Examples
LIST %R1 ;
%R7 = 0 ;

%SUBSYSTEM

Contains the name of the underlying subsystem, if any, where the program is executing. The possible values are listed below:

- CICS
- TSO
- NONE

Attributes

- C and C++: unsigned char[
- COBOL: PICTURE X(j)
%SYSTEM

Contains the name of the operating system supporting the program. The only possible value is MVS.

Attributes

C and C++: unsigned char[]

COBOL: PICTURE X(j)

Attributes of Debug Tool variables in different languages

The table below shows the attributes for Debug Tool variables when used with different programming languages.

<table>
<thead>
<tr>
<th>Debug Tool variable</th>
<th>C and C++ attributes</th>
<th>COBOL attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>%GPRn</td>
<td>signed int</td>
<td>PICTURE S9(9)</td>
</tr>
<tr>
<td>%FPRn</td>
<td>float</td>
<td>USAGE COMP-1</td>
</tr>
<tr>
<td>%LPRn</td>
<td>double</td>
<td>USAGE COMP-2</td>
</tr>
<tr>
<td>%EPRn</td>
<td>long double</td>
<td>n/a</td>
</tr>
<tr>
<td>%ADDRESS</td>
<td>void *</td>
<td>USAGE POINTER</td>
</tr>
<tr>
<td>%AMODE</td>
<td>signed short int</td>
<td>PICTURE S9(4) USAGE COMP</td>
</tr>
<tr>
<td>%BLOCK</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
</tr>
<tr>
<td>%CAAADDRESS</td>
<td>void *</td>
<td>USAGE POINTER</td>
</tr>
<tr>
<td>%CONDITION</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
</tr>
<tr>
<td>%COUNTRY</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
</tr>
<tr>
<td>%CU</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
</tr>
<tr>
<td>%EPA</td>
<td>void *</td>
<td>USAGE POINTER</td>
</tr>
<tr>
<td>%HARDWARE</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
</tr>
<tr>
<td>%LINE</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
</tr>
<tr>
<td>%LOAD</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
</tr>
<tr>
<td>%NLANGUAGE</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
</tr>
<tr>
<td>%PATHCODE</td>
<td>signed short int</td>
<td>PICTURE S9(4) USAGE COMP</td>
</tr>
<tr>
<td>%PLANGUAGE</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
</tr>
<tr>
<td>%PROGRAM</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
</tr>
<tr>
<td>%RC</td>
<td>signed short int</td>
<td>PICTURE S9(4) USAGE COMP</td>
</tr>
<tr>
<td>%RUNMODE</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
</tr>
<tr>
<td>%STATEMENT</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
</tr>
<tr>
<td>%SUBSYSTEM</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
</tr>
<tr>
<td>%SYSTEM</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
</tr>
</tbody>
</table>
Chapter 8. Debug Tool messages

All messages are shown in this section are in mixed case English. The uppercase English message text is the same, but is in uppercase letters.

Each message has a number of the form EQA0nnnx, where EQA indicates that the message is an Debug Tool message, nnn is the number of the message, and x indicates the severity level of each message. The value of x is I, W, E, S, or U, as described below:

I An informational message calls attention to some aspect of a command response that might assist the programmer.

W A warning message calls attention to a situation that might not be what is expected or to a situation that Debug Tool attempted to fix.

E An error message describes an error that Debug Tool detected or cannot fix.

S A severe error message describes an error that indicates a command referring to bad data, control blocks, program structure, or something similar.

U An unrecoverable error message describes an error that prevents Debug Tool from continuing.

Symbols in messages

Many of the Debug Tool messages contain information that is inserted by the system when the message is issued. In this publication, such inserted information is indicated by italicized symbols, as in the following:

EQA1046I The breakpoint-id breakpoint is replaced.

The portion of Debug Tool located on the host notifies you of errors associated with debugging functions carried out by the host.

Refer to the following sections for more information related to the material discussed in this section.

Related tasks

z/OS Language Environment Programming Guide

Related references

“Allowable comparisons for the IF command (COBOL)” on page 116

“Allowable moves for the MOVE command (COBOL)” on page 146
abnormally. Diagnostic information is included in the
EVFERROR.LOG file, in the path specified by the CODETMPDIR
variable in your CONFIG.SYS file.

**EQA1000I**  TEST (cu_name initialization):

**Explanation:** Debug Tool is ready to accept a command from the
terminal. This message is used in line mode when an initial
prompt occurs after Debug Tool initialization and before any
program hooks are reached. Enter a command. If you are not sure
what you can enter, enter HELP or ?. Information is displayed
identifying the commands you are allowed to enter.

**EQA1001I**  The window configuration is configuration; the
sequence of window is sequence

**Explanation:** Used to display SCREEN as part of QUERY
SCREEN.

**EQA1002I**  One window must be open at all times.

**Explanation:** Only one window was open when a CLOSE
command was issued. At least one window must be open at all
times, so the CLOSE command is ignored.

**EQA1003I**  Target window is closed; FIND not performed.

**Explanation:** The window specified in the FIND command is
closed.

**EQA1004I**  Target window is closed; SIZE not performed.

**Explanation:** The window specified in the SIZE command is
closed.

**EQA1005I**  Target window is closed; SCROLL not performed.

**Explanation:** The window specified in the SCROLL command is
closed.

**EQA1006I**  Command

**Explanation:** It is the character string 'Command' in the main
panel command line.

**EQA1007I**  Step

**Explanation:** It is the character string 'Step' in the main panel
command line while stepping.

**EQA1008I**  Scroll

**Explanation:** It is the character string 'Scroll' in the main panel
command line.

**EQA1009I**  DBCS characters are not allowed.

**Explanation:** The user entered DBCS characters in scroll, window
object id, quality, prefix, or panel input areas.

**EQA1010I**  More...

**Explanation:** It is the character string 'More' in the main panel
command line.

**EQA1011I**  Do you really want to terminate this session?

**Explanation:** This is for the END pop-up window.

**EQA1012I**  Enter Y for YES and N for NO

**Explanation:** This is for the END pop-up window. Y, YES, N, and
NO should NOT be translated.

**EQA1013I**  Current command is incomplete, pending more
input

**Explanation:** This informational message is displayed while
entering a block of commands, until the command block is closed
by an END statement.

**EQA1030I**  PENDING:

**Explanation:** Debug Tool needs more input in order to
completely parse a command. This can occur in COBOL, for
example, because PERFORM; was entered on the last line.

**Programmer response:** Complete the command.

**EQA1031I**  list items

**Explanation:** This message is used to list all the items that can
follow a partially parsed command.

**Programmer response:** Reenter the acceptable part of the
command and suffix it with one of the items in this list.

**EQA1032I**  The next word can be one of:

**Explanation:** This title line will be followed by message 1015.

**EQA1033I**  The breakpoint-id breakpoint is replaced.

**Explanation:** This alerts the user to the fact that a previous
breakpoint action existed and was replaced.

**Programmer response:** Verify that this was intended.

**EQA1046I**  The breakpoint-id breakpoint is replaced.

**Explanation:** This alerts the user to the fact that a previous
breakpoint action existed and was replaced.

**Programmer response:** Verify that this was intended.

**EQA1048I**  Another generation of variable name is allocated.

**Explanation:** An ALLOCATE occurred for a variable where an
AT ALLOCATE breakpoint was established.

**EQA1049I**  The breakpoint-id breakpoint action is:

**Explanation:** Used to display a command after LIST AT when
there is no every_clause. Enabled breakpoints only. This message is
followed by a message of one or more lines showing the
commands performed each time the breakpoint is hit.
EQA1050I • EQA1091I

**EQA1050I**
The breakpoint-id breakpoint has an EVERY value of number, a FROM value of number, and a TO value of number. The breakpoint action is:

Explanation: Used to display a command after LIST AT when there is an every_clause. Enabled breakpoints only. This message is followed by a message of one or more lines showing the commands performed each time the breakpoint is hit.

**EQA1051I**
The (deferred) breakpoint-id breakpoint action is:

Explanation: Used to display a command after LIST AT when there is no every_clause. Deferred and enabled breakpoints only. This message is followed by a message of one or more lines showing the commands performed each time the breakpoint is hit.

**EQA1052I**
The (deferred) breakpoint-id breakpoint has an EVERY value of number, a FROM value of number, and a TO value of number. The breakpoint action is:

Explanation: Used to display a command after LIST AT when there is an every_clause. Deferred and enabled breakpoints only. This message is followed by a message of one or more lines showing the commands performed each time the breakpoint is hit.

**EQA1053I**
The (disabled) breakpoint-id breakpoint action is:

Explanation: Used to display a command after LIST AT when there is not an every_clause. For disabled breakpoints only. This message is followed by a message of one or more lines showing the commands performed each time the breakpoint is hit.

**EQA1054I**
The (disabled) breakpoint-id breakpoint has an EVERY value of number, a FROM value of number, and a TO value of number. The breakpoint action is:

Explanation: Used to display a command after LIST AT when there is not an every_clause. For disabled breakpoints only. This message is followed by a message of one or more lines showing the commands performed each time the breakpoint is hit.

**EQA1055I**
The (disabled and deferred) breakpoint-id breakpoint action is:

Explanation: Used to display a command after LIST AT when there is not an every_clause. For disabled and deferred breakpoints only. This message is followed by a message of one or more lines showing the commands performed each time the breakpoint is hit.

**EQA1056I**
The (disabled and deferred) breakpoint-id breakpoint has an EVERY value of number, a FROM value of number, and a TO value of number. The breakpoint action is:

Explanation: Used to display a command after LIST AT when there is an every_clause. For disabled and deferred breakpoints only. This message is followed by a message of one or more lines showing the commands performed each time the breakpoint is hit.

**EQA1057I**
AT stmt-number can be risky because the code for that statement might have been merged with that of another statement.

Explanation: You are trying to issue an AT STATEMENT command against a statement but the code for that statement was either optimized away or combined with other statements.

**EQA1058I**
RUNTO is active at statement_id.

Explanation: Display after LIST AT to reflect RUNTO position.

**EQA1059I**
The Entry breakpoint command for Load_Module_Name => CU Name has been deferred until the CU appears.

Explanation: The compilation unit (program) that you specified could not be located by Debug Tool. The breakpoint is deferred until this CU is entered.

**EQA1060I**
The following suspended breakpoint exists: BP_name.

Explanation: Used to display a command after LIST AT for suspended breakpoints only.

**EQA1061I**
The following suspended and disabled breakpoint exists: BP_name.

Explanation: Used to display a command after LIST AT for suspended and disabled breakpoints only.

**EQA1062I**
Direction an unknown program.

Explanation: The program might be written in an unsupported language or may be a disassembled program. The message is issued as a result of the LIST CALLS command.

**EQA1063I**
Direction Address Address in a PLANG NOTEST block.

Explanation: The compile unit was compiled without the TEST option. The message is issued as a result of the LIST CALLS command.

**EQA1064I**
Direction Place in PLANG CU

Explanation: CU name of the call chain. The message is issued as a result of the LIST CALLS command.

**EQA1065I**
Direction Address Address in PLANG CU

Explanation: The compile unit was compiled without the TEST option and is in the Debug Tool list of CUs.

**EQA1066I**
The previous declaration of variable name will be removed.

Explanation: You declared a variable whose name is the same as a previously declared variable. This declaration overrides the previous one.

**EQA1067I**
The compiler data for program cu_name is

Explanation: This is the title line for the DESCRIBE PROGRAM command.

**EQA1068I**
The program was compiled with the following options:

Explanation: This is the first of a group of DESCRIBE PROGRAM messages.
**EQA1092I**  
**compile option**  
**Explanation:** Used to display a compile option without parameters, for example, NOTEST.

**EQA1093I**  
**compile option (compile suboption)**  
**Explanation:** Used to display a compile option with one parameter, for example, OPT.

**EQA1094I**  
**compile option (compile suboption, compile suboption)**  
**Explanation:** Used to display a compile option with two parameters, for example, TEST.

**EQA1095I**  
**This program has no subblocks.**  
**Explanation:** A DESCRIBE PROGRAM command refers to a program that is totally contained in one block.

**EQA1096I**  
**The subblocks in this program are nested as follows:**  
**Explanation:** The names of the blocks contained by the program are displayed under this title line.

**EQA1097I**  
**space characters block name**  
**Explanation:** The first insert controls the indentation while the second is the block name without qualification.

**EQA1098I**  
**The statement table has the short format.**  
**Explanation:** The statement table is abbreviated such that no relationship between storage locations and statement identifications can be determined.

**Programmer response:** If statement identifications are required, the program must be recompiled with different compiler parameters.

**EQA1099I**  
**The statement table has the NUMBER format.**  
**Explanation:** The program named in the DESCRIBE PROGRAM command was compiled with GONUMBER assumed.

**EQA1100I**  
**The statement table has the STMT format.**  
**Explanation:** The program named in the DESCRIBE PROGRAM command was compiled with GOSTMT assumed.

**EQA1101I**  
**file name**  
**Explanation:** This message is used in listing items returned from the back end in response to the DESCRIBE ENVIRONMENT command.

**EQA1102I**  
**ATTRIBUTES for variable name**  
**Explanation:** Text of a DESCRIBE ATTRIBUTES message.

**EQA1103I**  
**Its address is address**  
**Explanation:** Text of a DESCRIBE ATTRIBUTES message.

**EQA1104I**  
**Compiler: Compiler version**  
**Explanation:** Indicates compiler version for DESCRIBE CU.

**EQA1105I**  
**Its length is length**  
**Explanation:** Text of a DESCRIBE ATTRIBUTES message.

**EQA1106I**  
**Programming language COBOL does not return information for DESCRIBE ENVIRONMENT**  
**Explanation:** COBOL run-time library does not return information to support this command.

**EQA1107I**  
**There are no open files.**  
**Explanation:** This is issued in response to DESCRIBE ENVIRONMENT if no open files are detected.

**EQA1108I**  
**The following conditions are enabled:**  
**Explanation:** This is the header message issued in response to DESCRIBE ENVIRONMENT before issuing the list of enabled conditions.

**EQA1109I**  
**The following conditions are disabled:**  
**Explanation:** This is the header message issued in response to DESCRIBE ENVIRONMENT before issuing the list of disabled conditions.

**EQA1110I**  
**This program has no Statement Table.**  
**Explanation:** This message is used for the DESCRIBE CU command. If a CU was compiled with NOTEST, no statement table was generated.

**EQA1111I**  
**Attributes for names in block block name**  
**Explanation:** This is a title line that is the result of a DESCRIBE ATTRIBUTES *; It precedes the names of all variables contained within a single block.

**EQA1112I**  
**variable name and/or attributes**  
**Explanation:** The first insert controls the indentation while the second is the qualified variable name followed by attribute string. (For C, only the attributes are given.)

**EQA1113I**  
**Currently open files are:**  
**Explanation:** This is the title line for the list of files that are known to be open. This is in response to the DESCRIBE ENVIRONMENT command.

**EQA1114I**  
**The program has insufficient compilation information for the DESCRIBE CU command.**  
**Explanation:** This program has insufficient information. It might be compiled without the TEST option.

**EQA1115I**  
**Common Language Environment math library is being used**  
**Explanation:** This is the response for the DESCRIBE ENVIRONMENT command when the Language Environment math library is being used.
<table>
<thead>
<tr>
<th>EQA1117I</th>
<th>PL/I Math library is being used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>This is the response for the DESCRIBE ENVIRONMENT command when the PL/I math library is being used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1118I</th>
<th>Compile option (compile suboption, compile suboption, compile suboption)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>Used to display a compile option with three parameters, for example TEST(ALL,SYM,SEPARATE).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1119I</th>
<th>Current allocations:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>Heading line for DESCRIBE ALLOCATIONS output.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1120I</th>
<th>VOLUME CAT DISP OPEN DDNAME DNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>Header for DESCRIBE ALLOCATIONS output.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1121I</th>
<th>Allocation description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>Description of the current allocation.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1122I</th>
<th>Insufficient storage is available to process command.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>There was not enough main memory available to process the command.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1123I</th>
<th>Return code return code/reason code reason code from macro name macro invocation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>During the processing of the command, the indicated macro invocation failed with the indicated return and reason codes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1124I</th>
<th>ALLOCATE / FREE failed. Return code return code/reason code reason code from dynamic allocation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>The dynamic allocation failed with the indicated return and reason codes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1125I</th>
<th>ALLOCATE / FREE failed. Dataset dname was not found.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>The indicated dataset was not cataloged or was not found on the volume on which it was cataloged.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1126I</th>
<th>ALLOCATE / FREE failed. The FILE specified was already in use (ALLOCATE) or was not allocated (FREE).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>If the command was ALLOCATE, the specified file was already allocated. FREE the file and retry the ALLOCATE. If the command was FREE, the specified file was not allocated.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1128I</th>
<th>ALLOCATE / FREE failed. Dataset dname is already allocated to another JOB or USER.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>The specified data set is already allocated in such a way that it cannot be allocated with the specified disposition.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1129I</th>
<th>Command not supported on the current platform.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>The command is not supported in the current environment (such as CICS).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1130I</th>
<th>The EQALANGX debug data also contains data for the following CUs:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>This is the header used to display the additional CSECT's included in the EQALANGX data for the current CU.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1131I</th>
<th>CU name CU language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>Used to display CSECT's also included in the EQALANGX data for the current CU.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1132I</th>
<th>EQALANGX version for this CU:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>This message is issued as part of the output of DESCRIBE CUS. It indicates the version of the EQALANGX program used to generate the debug data for the specified CU. If this version is earlier than the current version of the EQALANGX program, unexpected results may occur in some situations.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1133I</th>
<th>Current EQALANGX version:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>This message is issued as part of the output of DESCRIBE CUS. It indicates the version of the EQALANGX program that is current for this version of Debug Tool.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1134I</th>
<th>The INCLUDE files in this program are indexed as follows: INCLUDE_file_names.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>The names of the INCLUDE files contained by the program are displayed under this title line.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1140I</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>This message is used to produce output for LIST (...) ).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1141I</th>
<th>expression name = expression value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>This message is used to produce output for LIST TITLED (...) ) when an expression is a scalar.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1142I</th>
<th>expression element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>This insert is used for naming the expression for expression element.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA1143I</th>
<th>&gt;&gt;&gt; EXPRESSION ANALYSIS &lt;&lt;&lt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explanation:</td>
<td>First line of output from the ANALYZE EXPRESSION command.</td>
</tr>
</tbody>
</table>
EQA1144I • EQA1165I

EQA1144I alignment spaces. It is a bit field with offset bit offset.
Explanation: Text of a DESCRIBE ATTRIBUTES message.

EQA1145I Its Offset is offset.
Explanation: Text of a DESCRIBE ATTRIBUTES message.

EQA1146I column elements
Explanation: This message is used to produce a columned list. For example, it is used to format the response to LIST STATEMENT NUMBERS.

EQA1147I name
Explanation: The name of a variable that satisfies a LIST NAMES request is displayed.

EQA1148I name structure
Explanation: The name of a variable that satisfies a LIST NAMES request is displayed. It is contained within an aggregate but is a parent name and not an elemental data item.

EQA1149I name in parent name
Explanation: The name of a variable that satisfies a LIST NAMES request is displayed. It is contained within an aggregate and is an elemental data item.

EQA1150I name structure in parent name
Explanation: The name of a variable that satisfies a LIST NAMES request is displayed. It is an aggregate contained within another aggregate.

EQA1151I The following names are known in block block name
Explanation: This is a title line that is the result of a LIST NAMES command. It precedes the names of all variables contained within a single block.

EQA1152I The following session names are known
Explanation: This is a title line that is the result of a LIST NAMES command. It precedes the names of all session variables contained within a single block.

EQA1153I The following names with pattern pattern are known in block name
Explanation: This title line precedes the list of variable names that satisfy the pattern in a LIST NAMES command.

EQA1154I The following session names with pattern pattern are known
Explanation: This title line precedes the list of session names that satisfy the pattern in a LIST NAMES command.

EQA1155I The following CUs are known in Load Module name.
Explanation: This title line precedes a list of compile unit names for noninitial load modules in a LIST NAMES CUS command.

EQA1156I The following CUs with pattern pattern are known in Load Module name
Explanation: This title line precedes a list of compile unit names for noninitial load modules that satisfy the pattern in a LIST NAMES CUS command.

EQA1157I There are no CUs with pattern pattern in Load Module name.
Explanation: This line appears when no compile unit satisfied the pattern in a LIST NAMES CUS command for noninitial load modules.

EQA1158I The following CUs have pattern pattern
Explanation: This title line precedes a list of compile unit names for an initial load module in a LIST NAMES CUS command.

EQA1159I There are no CUs with pattern pattern.
Explanation: This line appears when no compile unit satisfied the pattern in a LIST NAMES CUS command for an initial load module.

EQA1160I There are no Procedures with pattern pattern.
Explanation: This line appears when no Procedures satisfied the pattern in a LIST NAMES PROCEDURES command.

EQA1161I The following Procedures have pattern pattern
Explanation: This title line precedes a list of Procedure names for a LIST NAMES PROCEDURES command.

EQA1162I There are no names in block block name
Explanation: The LIST NAMES command found no variables in the specified block.

EQA1163I There are no session names.
Explanation: The LIST NAMES command found no variables that had been declared in the session for the current programming language.

EQA1164I There are no names with pattern pattern in block name.
Explanation: The LIST NAMES command found named variables in the named block but none of the names satisfied the pattern.

EQA1165I There are no session names with pattern pattern.
Explanation: The LIST NAMES command found named variables that had been declared in the session but none of the names satisfied the pattern.
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA1166I</td>
<td>There are no known session procedures.</td>
<td>A LIST NAMES PROCEDURES was issued but no session procedures exist.</td>
</tr>
<tr>
<td>EQA1167I</td>
<td>register name = register value</td>
<td>Used when listing registers.</td>
</tr>
<tr>
<td>EQA1168I</td>
<td>No LIST STORAGE data is available for the requested reference or address.</td>
<td>The given reference or address is invalid.</td>
</tr>
<tr>
<td>EQA1169I</td>
<td>No frequency data is available</td>
<td></td>
</tr>
<tr>
<td>EQA1170I</td>
<td>Frequency of verb executions in cu_name</td>
<td>This is the header produced by the LIST FREQUENCY command.</td>
</tr>
<tr>
<td>EQA1171I</td>
<td>character string = count</td>
<td>This is the frequency count produced by the LIST FREQUENCY command.</td>
</tr>
<tr>
<td>EQA1172I</td>
<td>TOTAL VERBS= total statements, TOTAL VERBS EXECUTED= total statements executed, PERCENT EXECUTED= percent executed</td>
<td>This is the trailer produced by the LIST FREQUENCY command.</td>
</tr>
<tr>
<td>EQA1173I</td>
<td>(history number) ENTRY hook for cu_name</td>
<td>This is a LIST HISTORY message.</td>
</tr>
<tr>
<td>EQA1174I</td>
<td>(history number) ENTRY hook for block block name in cu_name</td>
<td>This is a LIST HISTORY message.</td>
</tr>
<tr>
<td>EQA1175I</td>
<td>(history number) EXIT hook for cu_name</td>
<td>This is a LIST HISTORY message.</td>
</tr>
<tr>
<td>EQA1176I</td>
<td>(history number) EXIT hook for block block name in cu_name</td>
<td>This is a LIST HISTORY message.</td>
</tr>
<tr>
<td>EQA1177I</td>
<td>(history number) STATEMENT hook at statement cu_name =&gt; statement_id</td>
<td>This is a LIST HISTORY message.</td>
</tr>
<tr>
<td>EQA1178I</td>
<td>(history number) PATH hook at statement cu_name =&gt; statement_id</td>
<td>This is a LIST HISTORY message.</td>
</tr>
<tr>
<td>EQA1179I</td>
<td>(history number) Before CALL hook at statement cu_name =&gt; statement_id</td>
<td>This is a LIST HISTORY message.</td>
</tr>
<tr>
<td>EQA1180I</td>
<td>(history number) CALL CEETEST at statement cu_name =&gt; statement_id</td>
<td>This is a LIST HISTORY message.</td>
</tr>
<tr>
<td>EQA1181I</td>
<td>(history number) Waiting for program input from dname</td>
<td>This is a LIST HISTORY message.</td>
</tr>
<tr>
<td>EQA1182I</td>
<td>(history number) LOAD occurred at statement cu_name =&gt; statement_id</td>
<td>This is a LIST HISTORY message.</td>
</tr>
<tr>
<td>EQA1183I</td>
<td>(history number) DELETE occurred at statement cu_name =&gt; statement_id</td>
<td>This is a LIST HISTORY message.</td>
</tr>
<tr>
<td>EQA1184I</td>
<td>(history number) condition name raised at statement cu_name =&gt; statement_id</td>
<td>This is a LIST HISTORY message.</td>
</tr>
<tr>
<td>EQA1185I</td>
<td>(history number) LABEL hook at statement cu_name =&gt; statement_id</td>
<td>This is a LIST HISTORY message.</td>
</tr>
<tr>
<td>EQA1186I</td>
<td>Unable to display value of variable name. Use LIST (variable name) for further details.</td>
<td>This is used to inform the user that for some reason one of the variables cannot be displayed for LIST TITLED.</td>
</tr>
<tr>
<td>EQA1187I</td>
<td>There are no data members in the requested object.</td>
<td>The requested object does not contain any data members. It contains only methods.</td>
</tr>
<tr>
<td>EQA1188I</td>
<td>(history number) DATE hook at statement cu_name =&gt; statement_id</td>
<td>This is a LIST HISTORY message.</td>
</tr>
<tr>
<td>EQA1189I</td>
<td>There are no CUs compiled with debug data. To see the CU names, issue SET DISASSEMBLY ON before LIST command.</td>
<td>This line appears when the setting of Disassembly is OFF and none of the compile units has a debug data. To see all names of these CUS issue SET DISASSEMBLY ON, and then repeat LIST NAMES CUS</td>
</tr>
<tr>
<td>EQA1190I</td>
<td>Unable to update the requested location.</td>
<td>The given reference or address is invalid.</td>
</tr>
</tbody>
</table>
EQA1191I  The length of target and source must be equal.
Explanation: The number of bytes to be altered must be equal the length of source.

EQA1192I  The number of bytes to be altered is too large.
Explanation: A maximum of 8 bytes of storage can be change when source is a hexadecimal number, and 4 bytes when source is integer number.

EQA1193I  There are no variables in section_name.
Explanation: The LIST TITLED command found no variables that had been declared in the section.

EQA1194I  The following variables are known in section section_name
Explanation: This is a title line that is the result of a LIST TITLED command. It precedes the list of all variables contained within a section.

EQA1226I  The EQUATE named EQUATE name is replaced.
Explanation: This alerts the user to the fact that a previous EQUATE existed and was replaced.
Programmer response: Verify that this was intended.

EQA1227I  The following EQUATE definitions are in effect:
Explanation: This is the header for the QUERY EQUATES command.

EQA1228I  EQUATE identifier = "EQUATE string"
Explanation: Used to display EQUATE identifiers and their associated strings. The string is enclosed in quotation marks so that any leading or trailing blanks are noticeable.

EQA1229I  The program is currently exiting block block name.
Explanation: Shows the bearings in an interrupted program.

EQA1230I  The program is currently entering prolog code for block name.
Explanation: Shows the bearings in an interrupted program.

EQA1231I  You are executing commands within a __ctest function.
Explanation: Shows the bearings in an interrupted program.

EQA1232I  You are executing commands within a CEETEST function.
Explanation: Shows the bearings in an interrupted program.

EQA1233I  The established MONITOR commands are:
Explanation: This is the header produced by LIST MONITOR.

EQA1234I  MONITOR monitor number monitor type
Explanation: This is the line produced by LIST MONITOR before each command is displayed.

EQA1235I  The command for MONITOR monitor number monitor type is:
Explanation: This is the header produced by LIST MONITOR monitor number.

EQA1236I  The MONITOR monitor number command is replaced.
Explanation: This is a safety message: the user is reminded that a MONITOR command is replacing an old one.

EQA1237I  The current qualification is block name.
Explanation: Shows the current point of view.

EQA1238I  The current location is cu_name => statement id.
Explanation: Shows the place where the program was interrupted.

EQA1239I  The program is currently entering block block name.
Explanation: Shows the bearings in an interrupted program.

EQA1240I  You are executing commands within a CALL PLITEST statement.
Explanation: Shows the bearings in an interrupted program.

EQA1241I  You are executing commands from the run-time command-list.
Explanation: Shows the bearings in an interrupted program.

EQA1242I  You are executing commands in the breakpoint-id breakpoint.
Explanation: Shows the bearings in an interrupted program.

EQA1243I  The setting of SET-command object is status
Explanation: The status of the object of a SET command is displayed when QUERYed individually.

EQA1244I  SET-command object status
Explanation: The status of the object of a SET command is displayed when issued as part of QUERY SET.

EQA1245I  The current settings are:
Explanation: This is the header for QUERY PKFEYS.

EQA1246I  PFKEY string command
Explanation: Used to display PFKEYS as part of QUERY PKFEYS.
<table>
<thead>
<tr>
<th>EQA1247I</th>
<th>COLOR color highlight intensity colored area</th>
<th><strong>EQA1276I</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong></td>
<td>Used to display SCREEN as part of QUERY SCREEN.</td>
<td><strong>EQA1258E</strong> There is no MONITOR LIST command with the referenced integer.</td>
</tr>
<tr>
<td>EQA1248I</td>
<td>You were prompted because STEP ended.</td>
<td><strong>Explanation:</strong> MONITOR n HEX or MONITOR n DEFAULT are valid only if n represents valid LIST command.</td>
</tr>
<tr>
<td><strong>Explanation:</strong></td>
<td>Shows the bearings in an interrupted program.</td>
<td><strong>EQA1259I</strong> The LOADDEBUGDATA command for CU_name has been deferred until the CU appears.</td>
</tr>
<tr>
<td>EQA1249I</td>
<td>character string - The QUERY source setting file name is not available.</td>
<td><strong>Explanation:</strong> The indicated CU is not currently known to Debug Tool. The LOADDEBUGDATA will be executed when the CU appears in a loaded module.</td>
</tr>
<tr>
<td><strong>Explanation:</strong></td>
<td>The source listing name is not available. The source listing was not required or set prior to this command.</td>
<td><strong>EQA1260I</strong> The CU specified for the LOADDEBUGDATA command is already an assembler or OS/VS COBOL CU.</td>
</tr>
<tr>
<td>EQA1250I</td>
<td>SET INTERCEPT is already set on or off for FILE filename.</td>
<td><strong>Explanation:</strong> An LDD has already been done for the CU specified in the LDD command. This LDD may have been done previously by the user or an implicit LDD may have been done for the CU. This happens when a user-entered LDD is successful and, subsequently, the CU goes away and later reappears.</td>
</tr>
<tr>
<td><strong>Explanation:</strong></td>
<td>You tried to issue the SET INTERCEPT ON/OFF for a file that is already set to ON/OFF. This is just an informational message to notify you that you are trying to duplicate the current setting. The command is ignored.</td>
<td><strong>EQA1261I</strong> The requested view is already active.</td>
</tr>
<tr>
<td>EQA1251I</td>
<td>You were prompted because RUNTO ended.</td>
<td><strong>Explanation:</strong> A CHANGEVIEW command requested a view that is already the active view for the currently qualified CU.</td>
</tr>
<tr>
<td><strong>Explanation:</strong></td>
<td>The program has stopped because RUNTO cursor/statement command reached the cursor location or pointed statement number.</td>
<td><strong>EQA1262I</strong> The requested view is not supported by the currently qualified CU.</td>
</tr>
<tr>
<td>EQA1252I</td>
<td>******* AUTOMONITOR *******</td>
<td><strong>Explanation:</strong> A CHANGEVIEW command requested a view that is not supported by the programming language of the currently qualified CU.</td>
</tr>
<tr>
<td><strong>Explanation:</strong></td>
<td>Header for the automonitor area in the monitor window.</td>
<td><strong>EQA1263W</strong> Breakpoints have been set on statements that would be suppressed in the requested view. The view is not changed.</td>
</tr>
<tr>
<td>EQA1253I</td>
<td>You were prompted because Playback replay mode ended.</td>
<td><strong>Explanation:</strong> Debug Tool does not currently support changing to a view that would suppress statements that currently contain breakpoints.</td>
</tr>
<tr>
<td><strong>Explanation:</strong></td>
<td>A PLAYBACK STOP command was processed, which terminated Playback replay mode.</td>
<td><strong>EQA1264I</strong> The current view is Current_View.</td>
</tr>
<tr>
<td>EQA1254I</td>
<td>The LOADDEBUGDATA command was not processed.</td>
<td><strong>Explanation:</strong> This message is displayed in response to the QUERY CURRENT VIEW command.</td>
</tr>
<tr>
<td><strong>Explanation:</strong></td>
<td>An error occurred so the LLD command was not processed.</td>
<td><strong>EQA1265E</strong> Command failed due to an internal communications error.</td>
</tr>
<tr>
<td>EQA1255E</td>
<td>The CU specified for the LOADDEBUGDATA command is not a disassembly CU.</td>
<td><strong>Explanation:</strong> The previous command could not be completed because of an internal communications error.</td>
</tr>
<tr>
<td><strong>Explanation:</strong></td>
<td>Only a disassembly CU can be identified as assembler CU.</td>
<td><strong>EQA1266I</strong> You were prompted because JUMPTO ended.</td>
</tr>
<tr>
<td>EQA1256E</td>
<td>An error occurred while attempting to load the debug (EQALANGX) file for a specified CU.</td>
<td><strong>Explanation:</strong> The program has stopped because JUMPTO command reached the cursor location or pointed statement number.</td>
</tr>
<tr>
<td><strong>Explanation:</strong></td>
<td>Either the file containing the EQALANGX debug data could not be found or there was an undetermined error loading the EQALANGX file for a specified CU.</td>
<td><strong>EQA1267I</strong> TEST:</td>
</tr>
<tr>
<td>EQA1257E</td>
<td>The MONITOR parameter HEX or DEFAULT is valid only for LIST reference command.</td>
<td><strong>Explanation:</strong> Debug Tool is ready to accept a command from the terminal.</td>
</tr>
<tr>
<td><strong>Explanation:</strong></td>
<td>MONITOR n HEX or MONITOR n DEFAULT are valid only if n represents LIST reference command.</td>
<td><strong>Programmer response:</strong> Enter a command. If you are not sure what you can enter, enter HELP or ?. Information is displayed identifying the commands you are allowed to enter.</td>
</tr>
</tbody>
</table>
The USE file is empty.

Programmer response: Correct the file specification and retry.

Explanation: This is part of a command that is being displayed in the log or in response to a LIST AT. Since a group of commands can be involved, their appearance is improved by indenting the subgroups. Therefore, the first insert is used for indentation, and the second to contain the command. This is the command as it is understood by Debug Tool.
• Truncated keywords are no longer truncated.
• Lowercase to uppercase conversion was done where appropriate.
• Only a single command is contained in a record. If multiple commands are involved, additional records are prepared using this format.

TEST (cu_name> statement_id):

Explanation: Debug Tool is ready to accept a command from the terminal. This message is used in line mode when an initial prompt occurs at a statement and a statement table is available.

Programmer response: Enter a command. If you are not sure what you can enter, enter HELP or ?. Information is displayed identifying the commands you are allowed to enter.

TEST (cu_name ENTRY):

Explanation: Debug Tool is ready to accept a command from the terminal. This message is used in line mode when an initial prompt occurs at a compile unit entry.

Programmer response: Enter a command. If you are not sure what you can enter, enter HELP or ?. Information will be displayed identifying the commands you are allowed to enter.

TEST (cu_name> block name ENTRY):

Explanation: Debug Tool is ready to accept a command from the terminal. This message is used in line mode when an initial prompt occurs at a block entry.

Programmer response: Enter a command. If you are not sure what you can enter, enter HELP or ?. Information will be displayed identifying the commands you are allowed to enter.

TEST (cu_name EXIT):

Explanation: Debug Tool is ready to accept a command from the terminal. This message is used in line mode when an initial prompt occurs at a compile unit exit.

Programmer response: Enter a command. If you are not sure what you can enter, enter HELP or ?. Information will be displayed identifying the commands you are allowed to enter.

TEST (cu_name block name EXIT):

Explanation: Debug Tool is ready to accept a command from the terminal. This message is used in line mode when an initial prompt occurs at a block exit.

Programmer response: Enter a command. If you are not sure what you can enter, enter HELP or ?. Information will be displayed identifying the commands you are allowed to enter.

TEST (label-name LABEL);

Explanation: Debug Tool is ready to accept a command from the terminal. This message is used in line mode when an initial prompt occurs at a label.

Programmer response: Enter a command. If you are not sure what you can enter, enter HELP or ?. Information is displayed identifying the commands you are allowed to enter.

(Application program has terminated)

Explanation: Debug Tool is ready to accept a command from the terminal. This message is used in full-screen mode when an initial prompt occurs at the termination of the application program.

Unknown

Explanation: Debug Tool is ready to accept a command from the terminal. This message is used in full-screen mode when an initial prompt occurs and the location is unknown.

initialization

Explanation: Debug Tool is ready to accept a command from the terminal. This message is used in full-screen mode when an initial prompt occurs after Debug Tool initialization and before any program hooks are reached.

ddname program output

Explanation: Displays program output with the ddname preceding the output.

The program is waiting for input from ddname

Explanation: Debug Tool has gained control because the program is waiting for input.

Use the INPUT command to enter recsize characters for the intercepted fixed-format file.

Explanation: Prompts you for intercepted input of fixed-format file.

Use the INPUT command to enter up to a maximum of recsize characters for the intercepted variable-format file.

Explanation: Prompt user for intercepted input of variable-formatted file.
EQA1293I  TEST (cu-name);
Explanation: Debug Tool is ready to accept a command from the terminal. This message is used in linemode when an initial prompt occurs at a statement and a statement table is not available.
Programmer response: Enter a command. If you are not sure of what you can enter, enter HELP or ?. Information is displayed identifying the available commands you are allowed to enter.

EQA1301I  A ABEND code ABEND occurred.
Explanation: A system or user ABEND has been detected by Debug Tool.

EQA1306I  You were prompted because the CONDITION name condition was raised in your program.
Explanation: The program has stopped running due to the occurrence of the named condition.
Programmer response: Verify the request. Reenter the command and name a specific procedure if necessary.

EQA1307I  You were prompted because an ATTENTION interrupt occurred.
Explanation: The attention request from the terminal was recognized and the Debug Tool was given control.

EQA1308I  You were prompted because a condition was raised in your program.
Explanation: The program stopped running due to the occurrence of a condition whose name is unknown.

EQA1309I  CONDITION name is a severity or class SEVERITY code condition.
Explanation: The condition named is described by its severity level or class code. See the z/OS Language Environment Programming Guide.

EQA1316I  Block block name contains the following statements:
Explanation: This message precedes the message that identifies all statement numbers in the block.

EQA1317I  block level space characters block name
Explanation: This message is used instead of EQA1097I when the number of block levels is greater than the indentation allowed.

EQA1318I  space_characters INCLUDE_file_name index_number
Explanation: The first insert controls the indentation, the second is the INCLUDE file name, and the third insert is the source index block level.

EQA1324I  Bottom of data reached.
Explanation: FIND has reached the bottom of the data without finding the string being searched for.

EQA1325I  Top of data reached.
Explanation: FIND has reached the top of the data without finding the string being searched for.

EQA1326I  character string
Explanation: This message is used to display the results of a command that the user has entered, such as LIST STORAGE.

EQA1327I  character string character string character string
Explanation: This message is used to display the results of a command that the user has entered.

EQA1329I  The procedure named procedure name has the form:
Explanation: This is the information that is produced when a LIST PROCEDURE command is processed. This message is followed by a message of one or more lines showing the commands that form the procedure.

EQA1330I  You are not currently within a procedure.
Explanation: The LIST PROCEDURE command was issued without naming a session procedure and the current command context is outside of a session procedure.
Programmer response: Verify the request. Reenter the command and name a specific procedure if necessary.

EQA1331I  The RETRIEVE queue is empty.
Explanation: There are no entries in the retrieve queue.

EQA1332I  FIND has continued from top of area.
Explanation: FIND searched the file to the end of the string without finding it and continues the search from the top, back to the starting point of the search.

EQA1333I  The string was found.
Explanation: FIND was successful in locating the target string. The line on which the string was found is displayed just above this message when operating in line mode.

EQA1334I  The operating system has generated the following message:
Explanation: The Operating System can issue its own messages. These are relayed to the user.

EQA1335I  OS message
Explanation: The operating system can issue its own messages. These are relayed to the user.

EQA1336I  IBM Debug Tool Version 7 Release 1 Mod 0 time stamp 5655-R44 and 5655-R45: (C) Copyright IBM Corp. 1992, 2006
Explanation: This message is used to place the Debug Tool logo, a timestamp, and copyright at the beginning of the log.
This message is also issued in response to the CALL %VER
| EQA1337I | Its address is address and its length is length |
| Explanation: | Text of a DESCRIBE ATTRIBUTES message for PL/I. |
| EQA1338I | Its offset is offset and its length is length |
| Explanation: | Text of a DESCRIBE ATTRIBUTES message for PL/I. |
| EQA1339I | Its length is length |
| Explanation: | Text of a DESCRIBE ATTRIBUTES message for PL/I. |
| EQA1340I | Its address is address |
| Explanation: | Text of a DESCRIBE ATTRIBUTES message for PL/I. |
| EQA1341I | Its Offset is offset |
| Explanation: | Text of a DESCRIBE ATTRIBUTES message for PL/I. |
| EQA1342I | ATTRIBUTES for variable name variable type |
| Explanation: | Text of a DESCRIBE ATTRIBUTES message for PL/I. |
| EQA1343I | Presently not in accessible storage |
| Explanation: | The requested variable cannot be accessed. |
| EQA1344I | The OTHERWISE statement would have been executed but was not present |
| Explanation: | The was no OTHERWISE clause present in the SELECT statement and none of the WHEN clauses were selected. This message is simply indicating that the OTHERWISE clause would have been executed if it had been present. |
| EQA1351I | Not enough memory available for PLAYBACK data collection. Increase memory specification on the PLAYBACK ENABLE command. |
| Explanation: | Not enough memory was available for Playback data collection. Increase the amount of memory available to Playback on the PLAYBACK ENABLE command. |
| EQA1354I | Return code RC from from PLAYBACK run-time API API name. PLAYBACK disabled. |
| Explanation: | The specified return code indicated an error condition detected by the Playback run-time API. Playback is disabled. |
| Programmer response: | Return code 63 indicates not enough memory was available for Playback. Restart your Debug Tool session and enter the PLAYBACK ENABLE command using the integer option. For example: PLAYBACK ENABLE = 10000 |
| EQA1355I | The current status of PLAYBACK is: PB insert1, PB insert2, PB insert3, PB insert4. |
| Explanation: | This message is issued in response to the QUERY PLAYBACK command. |
| EQA1356I | PLAYBACK START is not active. You are not in PLAYBACK replay mode. |
| Explanation: | This message is issued in response to the QUERY PLAYBACK LOCATION command when PLAYBACK replay is not active. |
| EQA1357I | PLAYBACK replay mode is active with PB Data. The direction is PB direction. |
| Explanation: | This message is issued in response to the QUERY PLAYBACK LOCATION command when PLAYBACK replay is active. |
| EQA1358I | The current location in the PLAYBACK buffer is PB location. |
| Explanation: | This message is issued in response to the QUERY PLAYBACK LOCATION command when PLAYBACK replay is active. |
| EQA1359I | Command is not allowed in PLAYBACK replay mode. |
| Explanation: | The previous command is not supported after PLAYBACK START. |
| EQA1360I | Command is not allowed in PLAYBACK NODATA replay mode. |
| Explanation: | The previous command is not supported after PLAYBACK START. |
| EQA1361I | PLAYBACK command not processed. |
| Explanation: | The previous command was ignored because of errors that were indicated by previous messages. |
EQA1362I  There is no data in the PLAYBACK buffer. 
Explanation:  A PLAYBACK START command was entered but 
the PLAYBACK buffer is empty. The PLAYBACK START command is not processed.

EQA1363W  PLAYBACK replay has reached the first record 
in the PLAYBACK buffer. You must enter 
PLAYBACK FORWARD or PLAYBACK STOP. 
Explanation:  Playback replay has reached the first record in the buffer. You cannot move farther backward.

EQA1364W  PLAYBACK replay has reached the last record in 
the PLAYBACK buffer. You must enter 
PLAYBACK BACKWARD or PLAYBACK STOP. 
Explanation:  Playback replay has reached the last record in the buffer. You cannot move farther forward.

EQA1365I  PLAYBACK replay has stepped over one or 
more deleted compile units or compile units for 
which PLAYBACK has been disabled. 
Explanation:  Playback replay has encountered data for a compile unit that is no longer active because the load module containing it was canceled or deleted, because the enclave containing it terminated, etc. Playback cannot replay this data because the necessary control blocks are no longer present.

EQA1370I  The following name_type names are currently 
exclude_or_included by user_or_debugtool. 
Explanation:  This message precedes the output of the NAMES QUERY command and indicates the type of names that follow this message.

EQA1371I  name 
Explanation:  This message lists the names output by the NAMES QUERY command.

EQA1372I  There are no name_type names currently 
exclude_or_included by user_or_debugtool. 
Explanation:  This message is issued by the NAMES QUERY command when there is no data to be displayed.

EQA1373E  A pattern of "*" is invalid. 
Explanation:  The NAMES EXCLUDE command does not allow a pattern of "*".

EQA1402E  Each window must have unique letters of L, M, and S. 
Explanation:  The user entered either duplicated letters or just one letter.

EQA1403E  Invalid prefix command was entered. 
Explanation:  The user entered an invalid prefix command.

EQA1404E  Search target not found. 
Explanation:  The target for the search command was not found.

EQA1405E  No previous search arguments exist; find not performed. 
Explanation:  A FIND command was issued without an argument. Since the FIND command had not been issued previously, Debug Tool had nothing to search for.

EQA1406E  Invalid window ID. 
Explanation:  The window header field contains an invalid window ID. Valid window IDs are SOURCE, MONITOR, and LOG.

EQA1407E  Invalid scroll amount entered. 
Explanation:  Scroll field contains an invalid scroll amount.

EQA1408E  Duplicate window ID 
Explanation:  More than one window header field contains the same window id.

EQA1409E  Debug Tool Utilities and Advanced Functions is required to use function. 
Explanation:  This function is available only with Debug Tool Utilities and Advanced Functions.

EQA1410E  Variable_name is a LABEL on a modifiable instruction. No AT commands can be issued against it. 
Explanation:  The specified label is on an instruction that is modified at some point in the program. Breakpoints cannot be set on such an instruction.

EQA1430W  The EQUATE named EQUATE name was has not been established. 
Explanation:  CLEAR EQUATE <name> was attempted for an EQUATE name that has not been established.

Programmer response:  For a list of the current EQUATES definitions, issue QUERY EQUATES.

EQA1431W  There are no EQUATE definitions in effect. 
Explanation:  CLEAR EQUATE or QUERY EQUATES was issued but there are no EQUATE definitions.
### EQA1432E • EQA1453E

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
<th>Programmer response</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA1432E</td>
<td><em>function is not supported.</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation: Language/Country is not supported.</td>
<td></td>
</tr>
<tr>
<td>EQA1433E</td>
<td>Switching to the programming language <em>language-name</em> is invalid because there are no <em>language-name</em> compilation units in the initial load module.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation: A SET PROGRAMMING LANGUAGE command was issued, but the initial load module contains no compilation units compiled in the language specified (or implied).</td>
<td></td>
</tr>
<tr>
<td>EQA1434E</td>
<td>Error in setting debug <em>name</em> to ??????????.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation: Refer to the maximum number of CUs allowed for debugging.</td>
<td></td>
</tr>
<tr>
<td>EQA1435E</td>
<td>Error in setting <em>name</em>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation: This is a generic message for SET command errors.</td>
<td></td>
</tr>
<tr>
<td>EQA1436W</td>
<td>SET EXECUTE is OFF -- command will not be executed.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation: The command was parsed but not executed.</td>
<td></td>
</tr>
<tr>
<td>EQA1437W</td>
<td>SET DYNDEBUG can not be executed at this time. SET DYNDEBUG can only be executed at the beginning of a debugging session, before any STEP or GO commands. The DYNDEBUG status has not been changed.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation: The Dynamic Debug facility setting can not be changed to ON in the middle of a debugging session.</td>
<td></td>
</tr>
<tr>
<td>EQA1438W</td>
<td>SET DYNDEBUG can not be executed at this time. SET DYNDEBUG can only be executed at the beginning of a debugging session, before any STEP or GO commands. The DYNDEBUG status has not been changed.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation: The Dynamic Debug facility setting can not be changed to OFF in the middle of a debugging session.</td>
<td></td>
</tr>
<tr>
<td>EQA1439E</td>
<td>This CU is not AUTOMONITOR capable for expressions.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation: The CU is not AUTOMONITOR capable.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Programmer response: Refer to the description of the SET AUTOMONITOR command in the <em><a href="#">Debug Tool Reference and Messages</a></em> document to determine the requirements this CU must fulfill in order to use the SET AUTOMONITOR command.</td>
<td></td>
</tr>
<tr>
<td>EQA1440E</td>
<td>SET AUTOMONITOR ON is not valid for this CU. Use the SET AUTOMONITOR ON LOG command to activate the statement trace function.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation: The current CU is not AUTOMONITOR capable. SET AUTOMONITOR ON LOG will activate the statement trace.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Programmer response: Refer to the description of the SET AUTOMONITOR command in the <em><a href="#">Debug Tool Reference and Messages</a></em> document to determine the requirements this CU must fulfill in order to use the SET AUTOMONITOR command.</td>
<td></td>
</tr>
<tr>
<td>EQA1441I</td>
<td>The statement trace is now active. Use the SET AUTOMONITOR OFF command to deactivate the statement trace.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation: The statement trace is active for a CU that is not AUTOMONITOR capable.</td>
<td></td>
</tr>
<tr>
<td>EQA1442E</td>
<td>DYNDEBUG cannot be SET OFF when running without the Language Environment run-time.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation: The Dynamic Debug facility cannot be deactivated while running without the Language Environment run-time.</td>
<td></td>
</tr>
<tr>
<td>EQA1443I</td>
<td>There are no INTERCEPT specifications in effect.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation: QUERY INTERCEPT was issued but there are no INTERCEPT specifications active.</td>
<td></td>
</tr>
<tr>
<td>EQA1444I</td>
<td>For this command to be effective, SETTINGS must be saved with SET RESTORE SETTINGS AUTO in effect.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation: In order for SET RESTORE BPS AUTO or SET RESTORE MONITORS AUTO to be effective, the settings must be automatically restored when Debug Tool is started. You entered one of these commands but the current setting is SET RESTORE SETTINGS NOAUTO.</td>
<td></td>
</tr>
<tr>
<td>EQA1449E</td>
<td>The command is not supported with PL/1. Use PL/1 built-in function HEX to obtain hexadecimal values.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation: %HEX and MONITOR LIST HEX are not valid for PL/1. It is recommended that the PL/1 built-in function HEX be used instead For example: LIST HEX(expr) or MONITOR LIST HEX(expr).</td>
<td></td>
</tr>
<tr>
<td>EQA1450E</td>
<td>Unable to display the result from expression evaluation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation: The entire result from the expression evaluation cannot be displayed; for example, the array is too large.</td>
<td></td>
</tr>
<tr>
<td>EQA1451E</td>
<td><em>operand</em> contains incompatible data type.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation: Comparison or assignment involves incompatible data types, or incompatible or unsupported date fields. If you are using COBOL, see <em><a href="https://example.com">Allowable comparisons for the IF command (COBOL)</a></em> for allowable comparisons for the Debug Tool IF command, and <em><a href="https://example.com">Allowable moves for the MOVE command (COBOL)</a></em> for allowable moves for the Debug Tool MOVE command.</td>
<td></td>
</tr>
<tr>
<td>EQA1452E</td>
<td><em>argument name</em> is not a valid argument.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation: The specified argument is not valid.</td>
<td></td>
</tr>
<tr>
<td>EQA1453E</td>
<td>The number of arguments is not correct.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Explanation: There are either too many or too few arguments specified.</td>
<td></td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td>Explanation</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EQA1454E</td>
<td>operand name is not a valid operand.</td>
<td>The specified operand is undefined or is an invalid literal.</td>
</tr>
</tbody>
</table>
| EQA1455E   | An unsupported operator/operand is specified.                                | An operator or an operand was not understood, and therefore was not processed. Examples of when this message is issued when using COBOL include:  
  • An attempt to perform arithmetic with a nonnumeric data item  
  • An attempt to perform arithmetic with a windowed date field or a year-last date field |
<p>| EQA1456E   | The variable variable name is undefined or is incorrectly qualified.        | The named variable could not be located or undefined.                                           |
| EQA1457E   | The exponent exponent contains a decimal point.                             | This feature is not supported. No decimal point is allowed in exponent specification.           |
| EQA1458E   | The address of data item has been determined to be invalid.                 | This can happen for items within a data record where the file is not active or the record area is not available; for items in a structure following Occurs, depending on the item where the ODO variable was not initialized; or for items in the LINKAGE SECTION that are not based on a valid address. |
| EQA1459E   | literal string is not a valid literal.                                      | The combination of characters specified for the literal is not a valid literal.                  |
| EQA1460E   | Operand operand name should be numeric.                                     | A nonnumeric operand was found where a numeric operand was expected.                            |
| EQA1461E   | Invalid data for data item is found.                                        | The memory location for a data item contains data that is inconsistent with the data type of the item. The item might not have been initialized. |
| EQA1462E   | Invalid sign for data item is found.                                        | The sign position of a signed data item contains an invalid sign. The item might not have been initialized. |
| EQA1463E   | A divisor of 0 is detected in a divide operation.                           | The expression contains a divide operation where the divisor was determined to be zero.         |
| EQA1464E   | data item is used as a receiver but it is not a data name.                  | The target of an assignment is not valid.                                                      |
| EQA1465E   | The TGT for a program is not available.                                     | The program might have been deleted or canceled.                                               |
| EQA1466E   | data item is not a valid subscript or index.                                | The subscript or index might be out of range or an ODO variable might not be initialized.      |
| EQA1467E   | No subscript or index is allowed for data item                             | One or more subscripts or indexes were specified for a data item that was not defined as a table. The reference to the data item is not allowed. |
| EQA1468E   | Missing subscripts or indexes for data item                                | A data item defined as a table was referenced without specifying any subscripts or indexes. The reference is not allowed. |
| EQA1469E   | Incorrect number of subscripts or indexes for data item                    | A data item defined as a table was referenced with incorrect number of subscripts or indexes. The reference is not allowed. |
| EQA1470E   | Incorrect length specification for data item                               | The length of a data item is incorrect for the definition, usually due to a faulty ODO object. |
| EQA1471E   | Incorrect value for ODO variable data item                                 | The ODO variable might not have been initialized, or the current value is out of range.        |
| EQA1472E   | Invalid specification of reference modification.                          | The specification of the reference modification is not consonant with the length field.         |
| EQA1473E   | Invalid zero value for data item                                           | The value of a data item is zero. A zero is invalid in the current context.                    |
| EQA1474E   | procedure name was found where a data name was expected.                   | Invalid name is specified for a data item.                                                     |
| EQA1475E   | data item is an invalid qualifier in a qualified reference.               | A qualified reference is invalid. One or more qualifiers might be undefined or not in the same structure as the desired data item. |</p>
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
<th>Explanation</th>
<th>Programmer Response</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA1476E</td>
<td>Too many qualifiers in a qualified reference.</td>
<td>The qualified reference contains more than the legal number of qualifiers.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1477E</td>
<td>DATA DIVISION does not contain any entries.</td>
<td>There is no data to display for a LIST * request because the DATA DIVISION does not contain any entries.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1478E</td>
<td>No status available for sort file sort file</td>
<td>Status was requested for a sort file. There is never a status available for a sort file.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1479E</td>
<td>Unable to locate any TGT. An attempt to locate any TGT failed.</td>
<td>No COBOL program exists in TEST mode.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1480E</td>
<td>operand name is an invalid operand for SET command.</td>
<td>The operands for a SET command are incorrect. At least one of the operands must be index name.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1481E</td>
<td>Too many digits for the exponent of floating point literal data item</td>
<td>The exponent specified for a floating-point literal contains too many digits.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1482E</td>
<td>command name command is terminated due to an error in processing.</td>
<td>The command is terminated unsuccessfully because an error occurred during processing.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1483E</td>
<td>reference could not be formatted for display.</td>
<td>The requested data item could not be displayed due to an error in locating or formatting the data item.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1484E</td>
<td>Resources (for example, heap storage) are not available for processing and the command is terminated unsuccessfully.</td>
<td>The command could not be completed due to inadequate resources.</td>
<td>Increase the region size and restart Debug Tool.</td>
<td></td>
</tr>
<tr>
<td>EQA1485E</td>
<td>The command is not supported because the CU is compiled with incorrect compiler options.</td>
<td>For COBOL, the CUs must be compiled with VS COBOL II Version 1 Release 3 and the TEST compiler or FDUMP option, or AD/Cycle® COBOL and the compile-time TEST option.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1486E</td>
<td>variable name is presently not in accessible storage.</td>
<td>The variable might be CONTROLLED or AUTOMATIC and does not yet exist.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1487E</td>
<td>The number of dimensions for variable name is number -- but number have been specified.</td>
<td>The wrong number of subscripts were specified with the variable reference.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1488E</td>
<td>The indices in variable name are invalid. Use the DESCRIBE ATTRIBUTES command (without any indices specified) to see the valid indices.</td>
<td>The subscripts with the variable reference do not properly relate to the variable’s characteristics.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1489S</td>
<td>variable name is not a based variable but a locator has been supplied for it.</td>
<td>A pointer cannot be used unless the variable is BASED.</td>
<td>Use additional qualification to get to the desired variable.</td>
<td></td>
</tr>
<tr>
<td>EQA1490S</td>
<td>variable name cannot be used as a locator variable.</td>
<td>Only variables whose data type is POINTER or OFFSET can be used to locator with other variables.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1491S</td>
<td>There is no variable named character string, and if it is meant to be a built-in function, the maximum number of arguments to the character string built-in function is number, but number were specified. If it is meant to be a STORAGE alteration command, the syntax is not valid.</td>
<td>A subscripted variable could not be found. Its name, however, is also that of a PL/I built-in function. If the built-in function was intended, the wrong number of arguments were present. It can be also STORAGE alteration function. If that command was intended, then invalid syntax was present.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1492S</td>
<td>There is no variable named character string, and if it is meant to be a built-in function, the minimum number of arguments to the character string built-in function is number, but number were specified.</td>
<td>A subscripted variable could not be found. Its name, however, is also that of a PL/I built-in function. If the built-in function was intended, more arguments must be present.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1493E</td>
<td>There is no variable named character string, and if it is meant to be a built-in function, remember built-in functions are allowed only in expressions.</td>
<td>A variable could not be found. Its name, however, is also that of a PL/I built-in function. If the built-in function was intended, it is not in the correct context. Note that in Debug Tool, pseudo-variables cannot be the target of assignments.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1494S</td>
<td>variable name is an aggregate. It cannot be used as a locator reference.</td>
<td>The variable that is being as a locator is not the correct data type.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error Code</td>
<td>Error Description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1495S</td>
<td>The name <strong>variable name</strong> is ambiguous and cannot be resolved.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explanation: Names of structure elements can be ambiguous if not fully qualified. For example, in DCL 1 A, 2 B, 3 Z POINTER, 2 C, 3 Z POINTER, the names Z and A.Z are ambiguous.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programmer response: Retry the command with enough qualification so that the name is unambiguous.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1496S</td>
<td>The name <strong>variable name</strong> refers to a structure, but structures are not supported within this context.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explanation: Given DCL 1 A, 2 B FIXED, 2 C FLOAT, the name A refers to a structure.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programmer response: Break the command into commands for each of the basic elements of the structure, or use the DECLARE command with a BASED variable to define a variable overlaying the structure.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1497S</td>
<td>An aggregate cannot be used as an index into an array.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explanation: Given DCL A(2) FIXED BIN(15) and DCL B(2) FIXED BIN(15), references to A(B), A(B+2), and so on are invalid.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programmer response: Use a scalar as the index.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1498S</td>
<td>Generation and recursion numbers must be positive.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explanation: In %GENERATION(x,y) and %RECURSION(x,y), y must be positive.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1499S</td>
<td>Generation and recursion expressions cannot be aggregate expressions.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explanation: In %GENERATION(x,y) and %RECURSION(x,y), y must be a scalar.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1500S</td>
<td>%RECURSION can be applied only to parameters and automatic variables.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explanation: In %RECURSION(x,y), x must be a parameter or an automatic variable.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1501S</td>
<td>%RECURSION number of procedure name does not exist. The present number of recursions of the block block name is number.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explanation: In %RECURSION(x,y), y must be no greater than the number of recursions of the block where x is declared.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1502S</td>
<td>%Generation can be applied only to controlled variables.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explanation: In %GENERATION(x,y), x must be controlled.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1503S</td>
<td>%Generation number of variable name does not exist. The present number of allocations of variable name is number.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explanation: In %GENERATION(x,y), y must be no greater than the number of allocations of the variable x.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA1504S</td>
<td>%Generation number of %RECURSION (procedure name, number) does not exist. The present number of allocations of %RECURSION (procedure name, number) is number.</td>
</tr>
<tr>
<td>Explanation: In %GENERATION(x,y), y must be no greater than the number of allocations of the variable x.</td>
<td></td>
</tr>
<tr>
<td>EQA1505S</td>
<td>The variable <strong>variable name</strong> belongs to a FETCHed procedure and is a CONTROLLED variable that is not a parameter. This violates the rules of PL/I.</td>
</tr>
<tr>
<td>Explanation: PL/I does not allow FETCHed procedures to contain CONTROLLED variable types.</td>
<td></td>
</tr>
<tr>
<td>Programmer response: Correct the program.</td>
<td></td>
</tr>
<tr>
<td>EQA1506S</td>
<td>The variable <strong>character string</strong> cannot be used.</td>
</tr>
<tr>
<td>Explanation: The variable belongs to the class of variables, such as members of structures with REFER statements, which Debug Tool does not support.</td>
<td></td>
</tr>
<tr>
<td>EQA1507E</td>
<td>The expression in the QUIT command must be a scalar that can be converted to an integer value.</td>
</tr>
<tr>
<td>Explanation: The expression in the QUIT command cannot be an array, a structure or other data aggregate, and if it is a scalar, it must have a type that can be converted to integer.</td>
<td></td>
</tr>
<tr>
<td>EQA1508E</td>
<td>An internal error occurred in C run time during expression processing.</td>
</tr>
<tr>
<td>Explanation: This message applies to C. An internal error occurred in the C run time and the command is terminated.</td>
<td></td>
</tr>
<tr>
<td>EQA1509E</td>
<td>The unary operator <strong>operator name</strong> requires a scalar operand.</td>
</tr>
<tr>
<td>Explanation: This message applies to the C unary operator ! (logical negation).</td>
<td></td>
</tr>
<tr>
<td>EQA1510E</td>
<td>The unary operator <strong>operator name</strong> requires a modifiable lvalue for its operand.</td>
</tr>
<tr>
<td>Explanation: This message applies to the C unary operators ++ and --.</td>
<td></td>
</tr>
<tr>
<td>EQA1511E</td>
<td>The unary operator <strong>operator name</strong> requires an integer operand.</td>
</tr>
<tr>
<td>Explanation: This message applies to the C unary operator ~ (bitwise complement).</td>
<td></td>
</tr>
<tr>
<td>EQA1512E</td>
<td>The unary operator <strong>operator name</strong> requires an operand that is either arithmetic or a pointer to a type with defined size.</td>
</tr>
<tr>
<td>Explanation: This message applies to the C unary operators + and -. These operators cannot be applied to pointers to void-function designators, or pointers to functions.</td>
<td></td>
</tr>
</tbody>
</table>
EQA1513E  The unary operator operator requires an arithmetic operand.
Explanation: This message applies to the C unary operator + and -.

EQA1514E  Too many arguments specified in function call.
Explanation: This message applies to C function calls.

EQA1515E  Too few arguments specified in function call.
Explanation: This message applies to C function calls.

EQA1516E  The logical operator operator requires a scalar operand.
Explanation: This message applies to the C binary operators && (logical and) and || (logical or).

EQA1517E  The operand of the type cast operator must be scalar.
Explanation: This message applies to the C type casts.

EQA1518E  The named type of the type cast operator must not be an expression.
Explanation: This message applies to the C type casts.

EQA1519E  A real type cannot be cast to a pointer type.
Explanation: This message applies to C type casts. In the example 'float f;', the type cast '(float *) f' is invalid.

EQA1520E  A pointer type cannot be cast to a real type.
Explanation: Invalid operand for the type cast operator.

EQA1521E  The operand in a typecast must be scalar.
Explanation: This message applies to C type casts.

EQA1522E  Argument argument in function call function has an invalid type.
Explanation: This message applies to C function calls.

EQA1523E  Invalid type for function call.
Explanation: This message applies to C function calls.

EQA1524E  The first operand of the subscript operator must be a pointer to a type with defined size.
Explanation: This message applies to the C subscript operator. The subscript operator cannot be applied to pointers to void, function designators or pointers to functions.

EQA1525E  Subscripts must have integer type.
Explanation: This message applies to the C subscript operator.

EQA1526E  The first operand of the sizeof operator must not be a function designator, a typedef, a bitfield or a void type.
Explanation: This message applies to the C unary operator sizeof.

EQA1527E  The second operand of the operator operator must be a member of the structure or union specified by the first operand.
Explanation: This message applies to the C operators (select member) and -> (point at member).

EQA1528E  The first operand of the operator operator must have type pointer to struct or pointer to union.
Explanation: This message applies to the C operator -> (point at member).

EQA1529E  The operand of the operator operator must be an array, a function designator, or a pointer to a type other than void.
Explanation: This message applies to the C indirection operator.

EQA1530E  The first operand of the operator operator must have type struct or union.
Explanation: This message applies to the C subscript operator (select member).

EQA1531E  The relational operator operator requires comparable data types.
Explanation: This message applies to the C relational operators. For example, <, >, <=, >=, and ==.

EQA1532E  The subtraction operator requires that both operands have arithmetic type or that the left operand is a pointer to a type with defined size and the right operand has the same pointer type or an integral type.
Explanation: This message applies to the C binary operator -. The difference between two pointers to void or two pointers to functions is undefined because sizeof is not defined for void types and function designators.

EQA1533E  Assignment contains incompatible types.
Explanation: This message applies to C assignments, for example, +=, -=, and *=.

EQA1534E  The TEST expression in the switch operator must have integer type.
Explanation: This applies to the test expression in a C switch command.

EQA1535E  The addition operator requires that both operands have arithmetic or that one operand has integer type and the other operand is a pointer to a type with defined size.
Explanation: This message applies to the C binary operator +.
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA1536E</td>
<td>The operand of the address operator must be a function designator or an Ivalue that is not a bitfield.</td>
<td>This message applies to the C unary operator &amp; (address).</td>
</tr>
<tr>
<td>EQA1537E</td>
<td>Invalid constant for the C language.</td>
<td>This message applies to C constants.</td>
</tr>
<tr>
<td>EQA1538E</td>
<td>Argument argument in function call function is incompatible with the function definition. Since Warning is on, the function call is not made.</td>
<td>This message applies to C function calls. The argument must have a type that would be valid in an assignment to the parameter.</td>
</tr>
<tr>
<td>EQA1539E</td>
<td>The binary operator operator requires integer operands.</td>
<td>This message applies to the C binary operator requires integer operands.</td>
</tr>
<tr>
<td>EQA1540E</td>
<td>The binary operator operator requires a modifiable Ivalue for its first operand.</td>
<td>This message applies to the C binary assignment operators.</td>
</tr>
<tr>
<td>EQA1541E</td>
<td>The binary operator operator requires arithmetic operands.</td>
<td>This message applies to the C binary operators * and / .</td>
</tr>
<tr>
<td>EQA1542E</td>
<td>Source in assignment to an enum is not a member of the enum. Since Warning is on, the operation is not performed.</td>
<td>This message applies to C. You attempted to assign a value to enum, but the value is not legitimate for that enum.</td>
</tr>
<tr>
<td>EQA1543E</td>
<td>Invalid value for the shift operator operator.</td>
<td>This message applies to the C binary operators &lt;&lt; (bitwise left shift) and &gt;&gt; (bitwise right shift). Shift values must be nonnegative and less than 33. These tests are made only when WARNING is on.</td>
</tr>
<tr>
<td>EQA1544E</td>
<td>Array subscript is negative. Since Warning is on, the operation is not performed.</td>
<td>This message applies to the C subscripts.</td>
</tr>
<tr>
<td>EQA1545E</td>
<td>Array subscript exceeds maximum declared value. Since Warning is on, the operation is not performed.</td>
<td>This message applies to the C subscripts.</td>
</tr>
<tr>
<td>EQA1546E</td>
<td>ZeroDivide would have occurred in performing a division operator. Since Warning is on, the operation is not performed.</td>
<td>Divide by zero is detected by C run time.</td>
</tr>
<tr>
<td>EQA1547E</td>
<td>variable is presently not in accessible storage.</td>
<td>This message applies to C. Use the LIST NAMES command to list all known variables.</td>
</tr>
<tr>
<td>EQA1548E</td>
<td>There is no variable named variable</td>
<td>This message applies to C. Use the LIST NAMES command to list all known variables.</td>
</tr>
<tr>
<td>EQA1549E</td>
<td>The function call function is not performed because the function linkages do not match.</td>
<td>This message applies to C function calls and can occur, for example, when a function's linkage is specified as CEE, but the function was compiled with linkage OS.</td>
</tr>
<tr>
<td>EQA1550E</td>
<td>There is no typedef identifier named name</td>
<td>This message applies to C. The message is issued, for example, in response to the command DESCRIBE ATTRIBUTE typedef x, if x is not a typedef identifier.</td>
</tr>
<tr>
<td>EQA1551E</td>
<td>name is the name of a member of an enum type.</td>
<td>This message applies to C.</td>
</tr>
<tr>
<td>EQA1552E</td>
<td>The name name is invalid.</td>
<td>This message applies to C declarations.</td>
</tr>
<tr>
<td>EQA1553E</td>
<td>Linkage type for function call function is unknown.</td>
<td>This message applies to C function calls.</td>
</tr>
<tr>
<td>EQA1554E</td>
<td>Function call function has linkage type PL/I, which is not supported.</td>
<td>This message applies to C function calls.</td>
</tr>
<tr>
<td>EQA1555E</td>
<td>Function call function has linkage type FORTRAN which is not supported.</td>
<td>This message applies to C function calls.</td>
</tr>
<tr>
<td>EQA1556E</td>
<td>name is a tag name. This cannot be listed since it has no storage associated with it.</td>
<td>This message applies to C tag names.</td>
</tr>
</tbody>
</table>
### EQA1557E  •  EQA1581E

**Explanation:**

This message applies to C names.

**EQA1558E**  

*name* has storage class void, not permitted on the LIST command.

**Explanation:**

This message applies to C. In the example 'void' funcname (...), the command LIST TITLED (funcname()) is invalid.

**EQA1559E**  

The second operand of the %RECURSION operator must be arithmetic.

**Explanation:**

This message applies to C. In %RECURSION(x,y), the second expression, y, must have arithmetic type.

**EQA1560E**  

The second operand of the %RECURSION operator must be positive.

**Explanation:**

This message applies to C. In %RECURSION(x,y), the second expression, y, must be positive.

**EQA1561E**  

The first operand of the %RECURSION operator must be a parameter or an automatic variable.

**Explanation:**

This message applies to C. In %RECURSION(x,y), the first expression, x, must be a parameter or an automatic variable.

**EQA1562E**  

The first operand of the %INSTANCE operator must be a parameter or an automatic variable.

**Explanation:**

This message applies to C. In %INSTANCE(x,y), the first expression, x, must be a parameter or an automatic variable.

**EQA1563E**  

Generation specified for %RECURSION is too large.

**Explanation:**

This message applies to C. In %RECURSION(x,y), the recursion number, y, exceeds the number of generations of x that are currently active.

**EQA1564E**  

The identifier *identifier* has been replaced.

**Explanation:**

This message applies to C declarations.

**EQA1565E**  

The declaration is too large

**Explanation:**

This message applies to C declarations.

**EQA1566E**  

An attempt to modify a constant was made. Since Warning is on, the operation is not performed.

**Explanation:**

This message applies to C.

**EQA1567E**  

An attempt to take the address of a variable with register storage was made. Since Warning is on, the operation is not performed.

**Explanation:**

This message applies to C.

**EQA1568E**  

Type of expression to %DUMP must be a literal string.

**Explanation:**

This message applies to C.

**EQA1569E**  

Octal constant is too long.

**Explanation:**

This message applies to C constants.

**EQA1570E**  

Octal constant is too big.

**Explanation:**

This message applies to C constants.

**EQA1571E**  

Hex constant is too long.

**Explanation:**

This message applies to C constants.

**EQA1572E**  

Decimal constant is too long.

**Explanation:**

This message applies to C constants.

**EQA1573E**  

Decimal constant is too big.

**Explanation:**

This message applies to C constants.

**EQA1574E**  

Float constant is too long.

**Explanation:**

This message applies to C constants.

**EQA1575E**  

Float constant is too big.

**Explanation:**

This message applies to C constants.

**EQA1576E**  

The environment is not yet fully initialized.

**Explanation:**

You can STEP and try the command again.

**EQA1577E**  

Size of the aggregate is too large.

**Explanation:**

This message applies to PL/I constants.

**EQA1578E**  

Only "=" and "~=" are allowed as operators in comparisons involving program control data.

**Explanation:**

Other relationships between program control data are not defined.

**Programmer response:**

Check to see if a variable was misspelled.

**EQA1579E**  

Program control data may be compared only with program control data of the same type.

**Explanation:**

ENTRY vs ENTRY, LABEL vs LABEL, etc. are okay. LABEL vs ENTRY is not.

**EQA1580E**  

Area variables cannot be compared.

**Explanation:**

Equivalency between AREA variables is not defined.

**EQA1581E**  

Aggregates are not allowed in conditional expressions such as the expressions in IF ... THEN, WHILE (...), UNTIL (...), and WHEN (... clauses.

**Explanation:**

This is not supported.
Programmer response: Check to see if the variable name was misspelled. If this was not the problem, you must find other logic to perform the task.

EQA1582E  Only "=a" and "=~a" are allowed as operators in comparisons involving complex numbers.
Explanation: Equal and not equal are defined for complex variables, but you have attempted to relate them in some other way.

EQA1583E  Strings with the GRAPHIC attribute may be concatenated only with other strings with the GRAPHIC attribute.
Explanation: You are not allowed to concatenate GRAPHIC (DBCS) strings to anything other than other GRAPHIC (DBCS) strings.

EQA1584E  Strings with the GRAPHIC attribute may be compared only with other strings with the GRAPHIC attribute.
Explanation: Equivalency between the GRAPHIC data type and other data types has not been defined.

EQA1585E  Only numeric data, character strings, and bit strings may be the source for conversion to character data.
Explanation: You are trying to convert something to a character format when such a relationship has not been defined.

EQA1586E  Only numeric data, character strings, and bit strings may be the source for conversion to bit data.
Explanation: You are trying to convert something to a bit format when such a relationship has not been defined.

EQA1587E  Only numeric data, character strings, bit strings, and pointers may be the source for conversion to numeric data.
Explanation: You are trying to convert something to a numeric format when such a relationship has not been defined.

EQA1588E  Aggregates are not allowed in control expressions.
Explanation: This message applies to PL/1 constants.

EQA1589W  CONVERSION would have occurred in performing a CHARACTER to BIT conversion, but since WARNING is on, the conversion will not be performed.
Explanation: The specified conversion probably contained characters that were something other than ‘0’ or ‘1’. Since the conversion to BIT could therefore not be done, this message is displayed rather than raising the CONVERSION condition.

EQA1590W  Varying string variable name has a length that is greater than its declared maximum. It will not be used in expressions until it is fixed.
Explanation: The variable named has been declared as VARYING with length n, but its current length is greater than n. The variable might be uninitialized or might have been written over.

EQA1591W  Varying string variable name has a negative string length. It will not be used in expressions until it is fixed.
Explanation: The variable named has been declared as VARYING with length n, but its current length is less than 0. The variable might be uninitialized or it might have been written over.

EQA1592W  Fixed decimal variable variable name contains bad data. Since WARNING is on, the operation will not be performed.
Explanation: A variable contains bad decimal data if its usage would cause a data exception to occur (that is, its numeric digits are not 0–9 or its sign indicator is invalid), or it has even precision but its leftmost digit is nonzero. LIST STORAGE can be used to find the contents of the variable, and an assignment statement can be used to correct them.

EQA1593W  The size of AREA variable variable name is less than zero. Since WARNING is on, the operation will not be performed.
Explanation: Negative sizes are not understood and, therefore, are not processed.

EQA1594W  The size of AREA variable variable name exceeds its declared maximum size. Since WARNING is on, the operation will not be performed.
Explanation: Performing the operation would alter storage that is outside of the AREA. Such an operation is not within PL/1, so will be avoided.

EQA1595W  Fixed binary variable variable name contains more significant digits than its precision allows. Since WARNING is on, the operation will not be performed.
Explanation: For example, a FIXED BIN(5,0) variable can have only 5 significant digits thus limiting its valid range of values to –32 through 31 inclusive.

EQA1596E  The subscripted variable variable name was not found. The name matches a built-in function, but the parameters are wrong.
Explanation: This message applies to PL/1 constants.

EQA1597E  AREA condition would have been raised
Explanation: This message applies to PL/1 constants.

EQA1598E  The bounds and dimensions of all arrays in an expression must be identical.
Explanation: Array elements of an expression (such as A + B or A = B) must all have the same number of dimensions and the same lower and upper bounds for each dimension.
EQA1599E • EQA1617W

EQA1599E  You cannot assign an array to a scalar.
Explanation:  The PL/I language does not define how a scalar would represent an array; the assignment is rejected as an error.

EQA1600E  Aggregate used in wrong context.
Explanation:  This message applies to PL/I constants.

EQA1601E  The second expression in the built-in function
name built-in function must be greater than or
equal to 1 and less than or equal to the number
of dimensions of the first expression.
Explanation:  The second expression of the named built-in
function is dependent upon the dimensions of the array (the first
built-in function argument).
Programmer response:  Correct the relationship between the first
and second arguments.

EQA1602E  The second expression in the built-in function
name built-in function must not be an aggregate.
Explanation:  Debug Tool does not support aggregates in this
case.

EQA1603E  The first argument in the built-in function
name built-in function must be an array expression.
Explanation:  The named built-in function expects an array to be the
first argument.

EQA1604E  Argument number number in the built-in function
name built-in function must be a variable.
Explanation:  You used something other than a variable name (for
example, a constant) in your invocation of the named built-in
function.

EQA1605E  STRING( variable name ) is invalid because the
STRING built-in function can be used only with
bit, character and picture variables.
Explanation:  You must use a variable of the correct data type
with the STRING built-in function.

EQA1606E  POINTER( variable name,...) is invalid because
the first argument to the POINTER built-in
function must be an offset variable.
Explanation:  The first argument to POINTER was determined to
be something other than an OFFSET data type.

EQA1607E  POINTER( variable name,...) is invalid because
the second argument to the POINTER built-in
function must be an area variable.
Explanation:  The second argument to POINTER was determined
to be something other than an AREA data type.

EQA1608E  OFFSET( variable name,...) is invalid because the
first argument to the OFFSET built-in function
must be a pointer variable.
Explanation:  The first argument to OFFSET was determined to
be something other than a POINTER data type.

EQA1609E  OFFSET( variable name,...) is invalid because the
second argument to the OFFSET built-in
function must be an area variable.
Explanation:  The second argument to OFFSET was determined
to be something other than an AREA data type.

EQA1610E  built-in function name( variable name ) is invalid
because the argument to the built-in function name
built-in function must be a file reference.
Explanation:  The name built-in function requires the name of a
FILE to operate. Some other data type was used as the argument.

EQA1611E  COUNT( variable name ) must refer to an open
STREAM file.
Explanation:  You must name an open STREAM file in the
COUNT built-in function.

EQA1612E  LINENO( variable name ) must refer to an open
PRINT file.
Explanation:  You must name an open PRINT file in the LINENO
built-in function.

EQA1613E  SAMEKEY( variable name ) must refer to a
RECORD file.
Explanation:  You must name a RECORD file in the SAMEKEY
built-in function. This requirement is tested for all file constants,
but is tested for file variables only if the file variable is associated
with an open file.

EQA1614E  The argument in the built-in function name
built-in function must be a variable.
Explanation:  The built-in function is expecting a variable but a
constant or some other invalid item appeared as one of the
arguments.

EQA1615E  Argument to POINTER is an aggregate when
pointer is being used as a locator.
Explanation:  This message applies to PL/I constants.

EQA1616E  The result of invoking the GRAPHIC built-in
function must not require more than 16383
DBCS characters.
Explanation:  GRAPHIC(x,y) is illegal if y > 16383, and
GRAPHIC(x) is illegal if length(CHAR(x)) > 16383.

EQA1617E  The first argument to the built-in function name
built-in function is negative, but since
WARNING is on, the evaluation will not be
performed.
Explanation:  The specified built-in function would fail if a
negative argument was passed. Use of the built-in function will be
avoided.
### EQA1628W
The second argument to the built-in function name `built-in function` is negative, but since `WARNING` is on, the evaluation will not be performed.

**Explanation:** The specified built-in function would fail if a negative argument was passed. Use of the built-in function will be avoided.

### EQA1629E
Built-in function `built-in function` is undefined if X=0 and Y=0

**Explanation:** This applies to PL/I constants.

### EQA1630E
The argument in `built-in function` is too large.

**Explanation:** This applies to the PL/I trigonometric built-in functions. For short floating-point arguments, the limits are:

**COS and SIN**
- `ABS(X) <= (2**18)*pi`
- `ABS(X) <= (2**17)*pi` if X is complex
- `ABS(IMAG(X)) <= (2**17)*pi` if X is complex
- `ABS(IMAG(X)) <= (2**18)*pi` if X is complex
- `ABS(IMAG(X)) <= (2**19)*pi` if X is complex

**COSD, SIND and TAND**
- `ABS(X) <= (2**18)*180`

For long floating-point arguments, the limits are:

**COS and SIN**
- `ABS(X) <= (2**50)*pi`
- `ABS(X) <= (2**49)*pi` if X is complex
- `ABS(IMAG(X)) <= (2**49)*pi` if X is complex
- `ABS(IMAG(X)) <= (2**50)*pi` if X is complex
- `ABS(IMAG(X)) <= (2**51)*pi` if X is complex

**COSD, SIND and TAND**
- `ABS(X) <= (2**50)*180`

For extended floating-point arguments, the limits are:

**COS and SIN**
- `ABS(X) <= (2**106)*pi`
- `ABS(X) <= (2**105)*pi` if X is complex
- `ABS(IMAG(X)) <= (2**105)*pi` if X is complex
- `ABS(IMAG(X)) <= (2**106)*pi` if X is complex
- `ABS(IMAG(X)) <= (2**107)*pi` if X is complex

**COSD, SIND and TAND**
- `ABS(X) <= (2**106)*180`

### EQA1631E
The subject of the SUBSTR pseudovariable (character string) is not a string.

**Explanation:** You are trying to get a substring from something other than a string.
EQA1632E  Argument to pseudovariable must be complex numeric.
Explanation: This message applies to PL/I constants.

EQA1633E  The first argument to a pseudovariable must refer to a variable, not an expression or a pseudovariable.
Explanation: The arguments that accompany a pseudovariable are incorrect.

EQA1634E  The length of the bit string that would be returned by UNSPEC is greater than the maximum for a bit variable. Processing of the expression will stop.
Explanation: This will occur in UNSPEC(A) where A is CHARACTER(n) and n > 4095, where A is CHARACTER(n) VARYING and n > 4093, where A is AREA(n) and n > 4080, etc.

EQA1635E  Maximum number of arguments to PLIDUMP subroutine is two
Explanation: This message applies to PL/I constants.

EQA1636E  Invalid argument in CALL %DUMP
Explanation: This message applies to PL/I constants.

EQA1637E  PL/I cannot process the expression expression name.
Explanation: This applies to PL/I constants.

EQA1638E  Argument argument number to the MPSTR built-in function must not have the GRAPHIC attribute.
Explanation: GRAPHIC (DBCS) strings are prohibited as arguments to the MPSTR built-in function.

EQA1639E  ALLOCATION( variable name ) is invalid because the ALLOCATION built-in function can be used only with controlled variables.
Explanation: You must name a variable that is ALLOCATEable.
Programmer response: The variable by that name cannot be a controlled variable within the current context. If the variable exists somewhere else (and is a controlled variable), you should use qualification with the variable name.

EQA1640E  variable name is an aggregate and hence is invalid as an argument to the POINTER built-in function when that built-in function is used as a locator.
Explanation: The argument to the POINTER built-in function is invalid. The argument to the POINTER built-in function should be an OFFSET data type for the first argument, or an AREA data type for the second argument.

EQA1641E  Structures are not supported within this context.
Explanation: Given DDCL 1 A, 2 B FIXED, 2 C FLOAT, the name A refers to a structure.

Programmer response: Break the command into commands for each of the basic elements of the structure, or use the DECLARE command with a BASED variable to define a variable overlaying the structure.

EQA1642E  The first argument to the built-in function name built-in function must have POINTER type.
Explanation: This applies to the POINTERADD built-in function. The first argument must have pointer type, and it must be possible to convert the other to FIXED BIN(31,0).

EQA1643E  The argument in the built-in function name built-in function must have data type: data type.
Explanation: This message applies to various built-in functions. By built-in function, the datatypes required are:
ENTRYADDR
ENTRY
BINARYVALUE
POINTER
BINVALUE
POINTER

EQA1644W  STRINGRANGE is disabled and the SUBSTR arguments are such that STRINGRANGE ought to be raised. Debug Tool will revise the SUBSTR reference as if STRINGRANGE were enabled.
Explanation: See the Language Reference built-in function chapter for the description of when STRINGRANGE is raised. See the Language Reference condition chapter for the values of the revised SUBSTR reference.

EQA1645E  The subject of the pseudovariable name pseudovariable must have data type: data type.
Explanation: This message applies to various pseudovariables. By pseudovariable, the datatypes required are:
ENTRYADDR
ENTRY VARIABLE

EQA1646E  built-in function (z) is undefined if z is COMPLEX.
Explanation: This applies to the PL/I ACOS, ASIN, ATAND, COSD, ERE, ERFC, LOG2, LOG10, SIND and TAND built-in functions. This applies to PL/I constants.

EQA1647I  Value is unprintable. Use LIST %HEX ( variable name to display the value.

EQA1648S  Only session variables may be modified in PLAYBACK replay mode.
Explanation: An attempt was made to modify storage during PLAYBACK replay mode when DATA was in effect. Only session variables can be modified in this situation.
EQA1649E  Error: see Command Log.
Explanation: An error has occurred during expression evaluation. See the Debug Tool Command Log for more detailed information.

EQA1650E  The range of statements statement_id - statement_id is invalid because the two statements belong to different blocks.
Explanation: AT stmt1-stmt2 is valid only if stmt1 and stmt2 are in the same block.

EQA1651W  The breakpoint-id breakpoint has not been established.
Explanation: You just issued a CLEAR/LIST command against a breakpoint that does not exist.
Programmer response: Verify that you referred to the breakpoint using the same syntax that was used to establish it. Perhaps a CLEAR command occurred since the command that established the breakpoint.

EQA1652E  Since the program for the statement statement-number does not have hooks at statements, AT commands are rejected for all statements in the program.
Explanation: The program has not been prepared properly so AT commands are rejected for all statements in the program.
Programmer response: Make sure the program has been prepared properly by checking Part 2. Preparing your program for debugging and Appendix E Syntax of the TEST Compiler option in the Debug Tool User's Guide. Also, the LIST LINE NUMBERS command can be used to list all statement or line numbers that are valid locations for an AT, LINE or AT STATEMENT breakpoint.

EQA1653E  A file name is invalid in this context.
Explanation: A command (for example, AT ENTRY) specified a C file name where a function or compound statement was expected.

EQA1654E  Since the cu cu_name does not have hooks at block entries and exits, all AT ENTRY and AT EXIT commands will be rejected for the cu.
Explanation: A compile unit must have been compiled with TEST(BLOCK), TEST(PATH) or TEST(ALL) for hooks to be present at block exits and block entries.

EQA1655E  Since the program for the label label-name does not have hooks at labels, AT commands are rejected for all labels in the program.
Explanation: A compilation unit must have been compiled with TEST(PATH) or TEST(ALL) for hooks to be present at labels.

EQA1656E  statement_id contains a value that is invalid in this context.
Explanation: %STATEMENT and %LINE are invalid in AT commands at block entry and block exit, and in AT and LIST STATEMENT commands at locations that are outside of the program.

EQA1657W  There are no breakpoint-class breakpoints set.
Explanation: The command CLEAR/LIST AT was entered but there are no AT breakpoints presently set, or the command CLEAR/LIST AT class was entered but there are no AT breakpoints presently set in that class.

EQA1658W  There are no enabled breakpoint-class breakpoints set.
Explanation: The command CLEAR/LIST AT was entered but there are no enabled AT breakpoints presently set in the requested class of breakpoints.

EQA1659W  There are no disabled breakpoint-class breakpoints set.
Explanation: The command CLEAR/LIST AT was entered but there are no disabled AT breakpoints presently set in the requested class of breakpoints.

EQA1660W  The breakpoint-id breakpoint is not enabled.
Explanation: You issued a specific LIST AT ENABLED command against a breakpoint that is not enabled.

EQA1661W  The breakpoint-id breakpoint is not disabled.
Explanation: You issued a specific LIST AT DISABLED command against a breakpoint that is not disabled.

EQA1662W  The breakpoint-id breakpoint cannot be triggered because it is disabled.
Explanation: You cannot TRIGGER a disabled breakpoint.

EQA1663W  There are no breakpoints set. No breakpoints are currently set.

EQA1664W  There are no disabled breakpoints set.
Explanation: No disabled breakpoints are currently set.

EQA1665W  There are no enabled breakpoints set.
Explanation: No enabled breakpoints are currently set.

EQA1666W  The breakpoint-id breakpoint is already enabled.
Explanation: You cannot ENABLE an enabled breakpoint.

EQA1667W  The breakpoint-id breakpoint is already disabled.
Explanation: You cannot DISABLE a disabled breakpoint.

EQA1668W  The attempt to set this breakpoint has failed.
Explanation: For some reason, when Debug Tool tried to set this breakpoint, an error occurred. This breakpoint cannot be set.
EQA1669W • EQA1683E

EQA1669W The FROM or EVERY value in a breakpoint command must not be greater than the specified TO value.

Explanation: In an every clause specified with a breakpoint command, if the TO value was specified, the FROM or EVERY value must be less than or equal to the TO value.

EQA1670W GO/RUN BYPASS is ignored. It is valid only when entered for an AT CALL, AT GLOBAL CALL, or AT OCCURRENCE.

Explanation: GO/RUN BYPASS is valid only when Debug Tool is entered for an AT CALL, AT GLOBAL CALL, or AT OCCURRENCE breakpoint.

EQA1671W AT OCCURRENCE breakpoint or TRIGGER of condition condition-name cannot have a reference specified. This command not processed.

Explanation: The following AT OCCURRENCE conditions must have a qualifying reference: CONDITION, ENDFILE, KEY, NAME, PENDING, RECORD, TRANSMIT and UNDEFINEDFILE. This would also apply to the corresponding TRIGGER commands.

EQA1672W AT OCCURRENCE breakpoint or TRIGGER of condition condition-name must have a valid reference specified. This command not processed.

Explanation: The following AT OCCURRENCE conditions must have a valid qualifying reference: CONDITION, ENDFILE, KEY, NAME, PENDING, RECORD, TRANSMIT and UNDEFINEDFILE. This would also apply to the corresponding TRIGGER commands.

EQA1673W An attempt to automatically restore an AT breakpoint type breakpoint failed.

Explanation: Debug Tool was attempting to restore a breakpoint that had been set in the previous process and has failed in that attempt. There are two reasons this could have happened. If the Compile Unit (CU) has been changed (that is, modified and recompiled/linked) between one process and the next or a breakpoint had been established for a statement or variable that no longer exists due to the change, when Debug Tool attempts to reestablish that breakpoint, it will fail with this message.

EQA1674W An attempt to automatically disable an AT breakpoint type breakpoint failed.

Explanation: Debug Tool was attempting to disable a breakpoint for a CU that has been deleted from storage (or deactivated), and failed in that attempt.

EQA1675E variable name is not a LABEL variable or constant. No GOTO or JUMPTO commands can be issued against it.

Explanation: You are trying to use a GOTO or JUMPTO command with a variable name that cannot be associated with a label in the program.

EQA1676S label name is a label variable that is uninitialized or that has been zeroed out. It cannot be displayed and should not be used except as the target of an assignment.

Explanation: You are trying to make use of a LABEL variable, but the control block representing that variable contains improper information (for example, an address that is zero).

EQA1677S file name is a file variable that is uninitialized or that has been zeroed out. It cannot be displayed and should not be used except as the target of an assignment.

Explanation: You are trying to make use of a FILE variable, but the control block representing that variable contains improper information (for example, an address that is zero).

EQA1678E The program CU-name has a short statement number table, and therefore no statement numbers in the program can be located.

Explanation: A command requires determining which statement was associated with a particular storage address. A statement table could not be located to relate storage to statement identifications.

Programmer response: Check to see if the program had been compiled using release name. If so, was the statement table suppressed?

EQA1679E variable name is not a controlled variable. An ALLOCATE breakpoint cannot be established for it.

Explanation: You cannot establish an AT ALLOCATE breakpoint for a variable that cannot be allocated.

EQA1680E variable name is a controlled parameter. An ALLOCATE breakpoint can be established for it only when the block in which it is declared is active.

Explanation: Debug Tool cannot, at this time, correlate a block to the named variable. As a result, a breakpoint cannot be established.

Programmer response: Establish the breakpoint via an AT ENTRY ... AT ALLOCATE ...

EQA1681E variable name is not a FILE variable or constant.

Explanation: ON/SIGNAL file-condition (variable) is invalid because the variable is not a PL/I FILE variable.

EQA1682E variable name is not a CONDITION variable.

Explanation: ON/SIGNAL CONDITION (variable) is invalid because the variable is not a PL/I CONDITION variable.

EQA1683E Since the cu cu_name does not have hooks at statements with modified behavior due to the Millennium Language Extensions, all AT DATE commands will be rejected for the cu.

Explanation: A compile unit must have been compiled with the DATEPROC option and either TEST(STMT) or TEST(ALL) for hooks to be present at statements affected by the Millennium Language Extensions.
Restoring of assembler breakpoints is not currently supported.
Explanation: Breakpoints in assembler compile units are not restored.

The command AT *Keyword* is not supported in the Compile Unit Cu_name.
Explanation: The command is not supported for a DISASSEMBLY compile unit. Only the AT OFFSET form of the AT command is supported for a DISASSEMBLY compile unit.

The command/list *Keyword* is not supported for Disassembly View.
Explanation: The command or option is not supported for a DISASSEMBLY compile unit. See the Debug Tool Reference and Messages document for information about the restrictions on the use of this command.

Variable *Variable* is not available during Playback replay.
Explanation: The expression cannot be evaluated during Playback replay, because the indicated variable is not available during replay.

A breakpoint cannot be set on this statement when the STORAGE runtime option is in effect. Remove STORAGE or set the breakpoint after the next LR R13,Rx instruction.
Explanation: When the STORAGE runtime option is in effect, breakpoints are not allowed on the prologue instructions between the first BALR R14,R15 and the next LR R13,Rx. You may set a breakpoint on an instruction following the next LR R13,Rx or you may rerun your program without the STORAGE runtime option and set a breakpoint on the specified statement.

The current programming language does not return information for DESCRIBE ENVIRONMENT.
Explanation: The current programming language does not support the DESCRIBE ENVIRONMENT command.

AT OCCURRENCE breakpoint or trigger of condition *string_ptr* is not supported with the current language. This command is not processed.
Explanation: The command is not supported for a ENTERPRISE PLI.

Variable *Variable* has a hex value that is too long to display.
Explanation: The expression has a hex value that exceeds the maximum length limit required to be displayable.

Conditional Expression *Conditional.Logic_EXPR* in WHEN clause can not be evaluated.
Explanation: The conditional expression is not valid. Make sure the variable is known in current compile unit or that the attributes are compatible.

Conditional Expression *Conditional.Logic_EXPR* in WHEN clause can not be evaluated at current location.
Explanation: The conditional expression is not valid. Make sure the variable is known in current compile unit or that the attributes are compatible.

The CU containing a referenced variable has not yet been entered. Storage does not exist for the referenced variable.
Explanation: You have attempted to evaluate a variable in an implicitly created CU. Storage has not yet been allocated for this variable.

Address Length Info Flags Name
Explanation: This message contains the output from the DESCRIBE LO80000 command.

The session procedure, *procedure name*, is either undefined or is hidden within a larger, containing procedure.
Explanation: This is issued in response to a CALL, CLEAR or QUERY command when the target session procedure cannot be located. It cannot be located for one of two reasons: it was not defined or it was imbedded with another session procedure.
EQA1701E  The maximum number of arguments to the %DUMP built-in subroutine is 2, but number were specified.

Explanation: %DUMP does not accept more than two parameters.

EQA1702E  Invalid argument in CALL %DUMP.

Explanation: In PL/I, the %DUMP arguments must be scalar data that can be converted to character. In C, the %DUMP arguments must be pointers to character or arrays of character.

EQA1703E  No arguments can be passed to a session procedure.

Explanation: Parameters are not supported with the CALL procedure command.

EQA1704E  Invalid or incompatible dump options or suboptions

Explanation: This message is from the feedback code of Language Environment CEE3DMP call.

EQA1705E  Dump argument exceeds the maximum length allowed.

Explanation: The dump option allows a maximum of 255 characters. The dump title allows a maximum of 80 characters.

EQA1706E  pgmname must be loaded before calling the program.

Explanation: The CALL command was terminated unsuccessfully.

EQA1707E  The following data was produced by Fault Analyzer.

Explanation: This message is used as a header for the call %FA.

EQA1708E  The HOGAN environment is not available.

Explanation: The Computer Sciences Corporation's KORE-HOGAN product is not installed.

EQA1709E  Command CALL %HOGAN is only available in a CICS environment.

Explanation: The CALL %HOGAN command is only valid in a CICS environment with Computer Sciences Corporation's KORE-HOGAN installed.

EQA1710E  You are not authorized to execute that function.

Explanation: The function that you requested has been rejected by a security manager.

EQA1711E  Program can not be found.

Explanation: An error occurred in locating the program needed to perform the function you requested.

EQA1712E  Function not available in Dual Terminal Mode.

Explanation: The function that you requested is not supported when Debug Tool is running in Dual Terminal mode.

EQA1713E  Load module load_module could not be found.

Explanation: The indicated load module was specified as an operand of the DESCRIBE LOADMODS command but is not an active load module.

EQA1714I  BP Operation successful for suspended breakpoint.

Explanation: The requested breakpoint was successfully performed on a suspended breakpoint.

EQA1720E  There is no declaration for variable name.

Explanation: A command (for example, CLEAR VARIABLES) requires the use of a variable, but the specified variable was not declared (or was previously cleared).

Programmer response: For a list of session variables that can be referenced in the current programming language, use the LIST NAMES TEST command.

EQA1721E  The size of the variable is too large.

Explanation: A variable can require no more than 2**24 - 1 bytes in a non-XA machine and no more than 2**31 - 1 bytes in an XA machine.

EQA1722E  Error in declaration; invalid attribute variable name.

Explanation: A session variable is declared with invalid or unsupported attribute.

EQA1723E  There is no session variables defined.

Explanation: The CLEAR VARIABLES command is entered but there is no declaration for session variables.

EQA1724E  There is no tag type tag named tag name.

Explanation: This message applies to C. It is issued, for example, after DESCRIBE ATTRIBUTES enum x if x is not an enum tag.

EQA1725E  tag type tag name is already defined.

Explanation: This message applies to C. A tagged enum, struct, or union type cannot be redefined, unless all variables and type definitions referring to that type and then the type itself are first cleared. For example, given

```c
enum colors {red,yellow,blue} primary, * ptrPrimary;
enum colors cannot be redefined unless primary, ptrPrimary, and then enum colors are first cleared.
```

EQA1726E  tag type tag name cannot be cleared while one or more declarations refer to that type.

Explanation: This message applies to C. A CLEAR DECLARE of a tagged enum, struct, or union type is invalid while one or more declarations refer to that type. For example, given
enum colors {red,yellow,blue} primary, * ptrPrimary;

CLEAR DECLARE enum colors is invalid until CLEAR
DECLARE (primary, ptrPrimary) is issued.

EQA1727E  enum member name is the name of a declared
variable. It cannot be used as the name of a
member of an enum type.
Explanation:  This message applies to C. For example, given
int blue;
The use of the name blue in the following declaration is invalid:
enum teamColors {blue,gold};

EQA1728E  The tag type tag name is recursive: it contains
itself as a member.
Explanation:  This message applies to C. A struct or union type
must not contain itself as a member. For example, the following
declaration is invalid:
struct record {
    int member;
    struct record next;
}

EQA1729E  An error occurred during declaration processing.
Explanation:  Unable to process the declaration. The command is
terminated unsuccessfully.

EQA1740E  EQALANGX debug file cannot be found for
Compile_Unit_name. Use the SET SOURCE
command to indicate the new location of the
EQALANGX file.
Explanation:  The EQALANGX file containing the listing and the
debugging tables cannot be found. Some of the possible conditions
that could cause this are: The debug file does not exist under the
default DSName, or the user does not have authorization to access
the debug file.

EQA1741E  Error in setting DBCS ON when the terminal is
not DBCS capable.
Explanation:  Error in setting DBCS ON when the debug session
terminal is not DBCS capable.

EQA1742I  Debug Trace: Trace Data
Explanation:  This is output generated by internal Debug Tool
trace for problem determination purposes only.

EQA1743I  save_restore_cmd not restored from dsname
Explanation:  An attempt was made to read the specified data set
in order to determine if settings should be automatically restored
or to restore the breakpoints and/or monitor settings. However,
the member did not exist, the data set could not be read, or the
data set contained invalid data. This might result from data having
never been saved in this data set.

EQA1744I  save_restore_cmd is in effect for dsname
Explanation:  An attempt was made to automatically restore
settings from the specified data set. However, the SETTINGS
NOAUTO option was in effect when the set data was saved and,
therefore, the set data will not be restored.

EQA1745I  save_restore_cmd restored from dsname
Explanation:  The specified data was successfully restored from
the specified data set.

EQA1746E  save_restore_cmd were not saved. Data set does
not exist: dsname
Explanation:  An attempt was made to save the indicated data in
the specified data set. However, the data set does not exist.
Allocate and catalog the data set and retry the operation.

EQA1747I  save_restore_cmd saved to dsname with restore_cmd
Explanation:  The indicated data was successfully saved to the
specified data set with the indicated restore options.

EQA1748E  save_restore_cmd unable to open dsname. Possible
RACF® error, invalid member name, etc.
Explanation:  An attempt was made to open the specified data set
in order to determine if settings should be automatically restored
or to save the current settings. However, the data set could be
allocated but could not be opened. This may be the result of not
having RACF access to the data set, of having a member name
specified that did not exist, or any other problem that could cause
a System 013 Abend.

EQA1749E  save_restore_cmd data set dsname is allocated to
another user or job.
Explanation:  An attempt was made to allocate the specified data
set in order to determine if settings should be automatically
restored or to save the current settings. However, the data set
could not be allocated because it was already allocated to another
user.

EQA1750E  An error occurred during expression evaluation.
Explanation:  Unable to evaluate the expression. The command is
terminated unsuccessfully.

EQA1751E  Program pgmname not found.
Explanation:  A bad program name is specified in a CALL
command and processing is terminated unsuccessfully.

EQA1752S  Comparison in command-name command was
invalid. The command was ignored.
Explanation:  This message applies to COBOL. The operands to
be compared are of incompatible types.

EQA1753S  The nesting of "switch" command exceeded the
maximum.
Explanation:  This message applies to C. There are too many
nested levels of switch commands.
### EQA1754S • EQA1779S

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA1754S</td>
<td>An error occurred in &quot;switch&quot; command processing. The command is terminated.</td>
<td>This message applies to C. The switch command is terminated because an error occurred during processing.</td>
</tr>
<tr>
<td>EQA1755S</td>
<td>Comparison with the keyword-name keyword in command-name command was invalid. The command was ignored.</td>
<td>This message applies to COBOL. The operands to be compared are incompatible. For example, the following comparison is invalid: EVALUATE TRUE when 6 = List ('invalid'); when other List ('other'); END-EVALUATE.</td>
</tr>
<tr>
<td>EQA1756W</td>
<td>EQALANGX file not available for CSECT CMD_NAME.</td>
<td>A debug data file is not found for the supplied CSECT name.</td>
</tr>
<tr>
<td>EQA1763E</td>
<td>save_restore_cmd failed because dsname is not a partitioned data set.</td>
<td>The specified data set must be a partitioned data set.</td>
</tr>
<tr>
<td>EQA1764E</td>
<td>save_restore_cmd could not locate data set dsname.</td>
<td>The specified data set could not be located.</td>
</tr>
<tr>
<td>EQA1765E</td>
<td>save_restore_cmd error rc-reason allocating dsname.</td>
<td>The specified data set could not be allocated. The return code and reason code are shown as &quot;ldd-xcccxyyqy&quot;. In most cases, xxxz is the 599Error code from dynamic allocation. This code may be used to determine more information about the source of the error. You should also inspect the MVS console log for other messages associated with this error.</td>
</tr>
<tr>
<td>EQA1766E</td>
<td>The target of the GOTO command is in an inactive block.</td>
<td>You are trying to GOTO a block that is not active. If it is inactive it doesn’t have a register save area, base registers, and so on -- all of the mechanics established that would permit the procedure to execute.</td>
</tr>
<tr>
<td>EQA1767S</td>
<td>No offset was found for label &quot;label&quot;.</td>
<td>No offset associated with the label was found; the code associated with the label might have been removed by optimization.</td>
</tr>
<tr>
<td>EQA1768S</td>
<td>The label &quot;label&quot; is not known.</td>
<td>The label is not known.</td>
</tr>
<tr>
<td>EQA1769S</td>
<td>The label &quot;label&quot; is ambiguous - multiple labels of this name exist.</td>
<td>The label is ambiguous; multiple labels of this name exist.</td>
</tr>
<tr>
<td>EQA1770S</td>
<td>The GOTO is not permitted, perhaps because of optimization.</td>
<td>The GOTO command is not recommended. For COBOL, this might be due to optimization, or because register contents other than the code base cannot be guaranteed for the target.</td>
</tr>
<tr>
<td>EQA1771S</td>
<td>The GOTO is not permitted due to language rules.</td>
<td>The GOTO command is not recommended. For COBOL, this might be due to optimization, or because register contents other than the code base cannot be guaranteed for the target.</td>
</tr>
<tr>
<td>EQA1772S</td>
<td>The GOTO was not successful.</td>
<td>There are various reasons why a GOTO command can be unsuccessful; this message covers all the other situations not covered by the other message in the GOTO LABEL messages group.</td>
</tr>
<tr>
<td>EQA1773E</td>
<td>GOTO is invalid when the target statement number is in a C function.</td>
<td>The target statement number in a GOTO command must belong to an active procedure.</td>
</tr>
<tr>
<td>EQA1776E</td>
<td>The target of the JUMPTO command is in an inactive block.</td>
<td>You are trying to JUMPTO a block that is not active. If it is inactive it doesn’t have a register save area, base registers, and so on -- all of the mechanics established that would permit the procedure to execute.</td>
</tr>
<tr>
<td>EQA1777E</td>
<td>variable_name is not a LABEL variable or constant. No JUMPTO commands can be issued against it.</td>
<td>You are trying to JUMPTO a variable name that cannot be associated with a label in the program.</td>
</tr>
<tr>
<td>EQA1778S</td>
<td>The JUMPTO is not allowed, perhaps because of optimization.</td>
<td>The JUMPTO command is not recommended. For COBOL, this may be due to optimization, or because register contents other than the code base cannot be guaranteed for the target.</td>
</tr>
<tr>
<td>EQA1779S</td>
<td>The JUMPTO is not permitted due to language rules.</td>
<td>The JUMPTO command is not recommended. For COBOL, this may be due to optimization, or because register contents other than the code base cannot be guaranteed for the target.</td>
</tr>
</tbody>
</table>
The JUMPTO was not successful.
Explanation: There are various reasons why a JUMPTO command may not be successful; this message covers all the other situations not covered by the other message in the JUMPTO LABEL messages group.

JUMPTO is invalid when the target statement number is in a C function.
Explanation: The target statement number in a JUMPTO command must belong to an active procedure.

There are no entries in the HISTORY table.
Explanation: Debug Tool has not yet encountered any of the situations that cause entries to be put into the HISTORY table; so it is empty.

There are no STATEMENT entries in the HISTORY table.
Explanation: LIST STATEMENTS or LIST LAST n STATEMENTS was entered, but there are no STATEMENT entries in the HISTORY table. Debug Tool was not invoked for any STATEMENT hooks.

There are no PATH entries in the HISTORY table.
Explanation: LIST PATH or LIST LAST n PATH was entered, but there are no PATH entries in the HISTORY table. Debug Tool was not invoked for any PATH hooks.

Requested register(s) not available.
Explanation: You are trying to work with a register but none exist in this context (for example, during environment initialization).

There are no active blocks.
Explanation: The LIST CALLS command was issued prior to any STEP or GO.

The pattern pattern is invalid.
Explanation: A pattern is invalid if it is longer than 128 bytes or has more than 16 parts. (Each asterisk and each name fragment forms a part.)

Only the ADDR and POINTER built-in functions may be used to specify an address in the LIST STORAGE command.
Explanation: LIST STORAGE(built-in function(...) is invalid if the built-in function is not the ADDR or POINTER built-in function.

ENTRY, FILE, LABEL, AREA, EVENT or TASK variables are not valid in a LIST command.
Explanation: The contents of these program control variables can be displayed by using the HEX or UNSPEC built-in functions or by using the LIST STORAGE command.

Block Block_name is not currently active.
Explanation: The block is not currently active for LIST TITLED Programmer response: Issue LIST TITLED or LIST TITLED * from within the block.

Symbol information at current location is not accessible.
Explanation: The symbols including variables or other data may not have been allocated at this location, you may STEP and issue the command again.

The command element character is invalid.
Explanation: The command entered could not be parsed because the specified element is invalid.

The command element character is ambiguous.
Explanation: The command entered could not be parsed because the specified element is ambiguous.

The hyphen cannot appear as the last character in an identifier.
Explanation: COBOL identifiers cannot end in an hyphen.

Incomplete command specified.
Explanation: The command, as it was entered, requires additional command elements (for example, keywords, variable names). Refer to the definition of the command and verify that all required elements of the command are present.

End-of-source has been encountered after an unmatched comment marker.
Explanation: A /* ... was entered but an */ was not present to close the comment. The command is discarded.
Programmer response: You must either add an */ to the end of the comment or explicitly indicate continuation with an SBCS hyphen.

End-of-source has been encountered after an unmatched quotation mark.
Explanation: The start of a constant was entered (a quotation mark started the constant) but another quotation mark was not found to terminate the constant before the end of the command was reached.
Programmer response: There could be several solutions for this, among them:
1. You must either add a quotation mark to the end of the constant or explicitly indicate continuation (with an SBCS hyphen).
2. If DBCS is ON you should also verify that you didn’t try to start a constant with an SBCS quotation mark and terminate it with a DBCS quotation mark (or vice versa).
3. You might have entered a character constant that contained a quotation mark — and you didn’t double it.
### EQA1812E • EQA1832E

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA1812E</td>
<td>A decimal exponent is required.</td>
<td>In COBOL, an E in a float constant must be followed by at least one decimal digit (optionally preceded by a sign). In <strong>C</strong>, if a + or – sign is specified after an E in a float constant, it must be followed by at least one decimal digit.</td>
</tr>
<tr>
<td>EQA1813E</td>
<td>Error reading DBCS character codes.</td>
<td>An unmatched or nested shift code was found.</td>
</tr>
<tr>
<td>EQA1814E</td>
<td>Identifier is too long.</td>
<td>All identifiers must be contained in 255 bytes or less. COBOL identifiers must be contained in 30 bytes or less and C identifiers in 255 bytes or less.</td>
</tr>
<tr>
<td>EQA1815E</td>
<td>Invalid character code within DBCS name, literal or DBCS portion of mixed literal.</td>
<td>A character code point was encountered that was not within the defined code values for the first or second byte of a DBCS character.</td>
</tr>
<tr>
<td>EQA1816E</td>
<td>An error was found at line line-number in the current input file.</td>
<td>An error was detected while parsing a command within a USE file, or within a file specified on the run-time TEST option. It occurred at the record number that was displayed.</td>
</tr>
<tr>
<td>EQA1817E</td>
<td>Invalid hexadecimal integer constant specified.</td>
<td>A hexadecimal digit must follow 0x.</td>
</tr>
<tr>
<td>EQA1818E</td>
<td>Invalid octal integer constant specified.</td>
<td>Only an octal digit can follow a digit-0.</td>
</tr>
<tr>
<td>EQA1819E</td>
<td>A COBOL DBCS name must contain at least one nonalphanumeric double byte character.</td>
<td>All COBOL DBCS names must have at least one double byte character not defined as double byte alphanumeric. For EBCDIC, these are characters with X'42' in the leading byte, with the trailing byte in the range X'41' to X'FE'. For ASCII, the leading byte is X'82' and the trailing byte is in the range X'40' to X'7E'.</td>
</tr>
<tr>
<td>EQA1820E</td>
<td>Invalid double byte alphanumeric character found in a COBOL DBCS name. Valid COBOL double byte alphanumeric characters are: A-Z, a-z, 0-9.</td>
<td>Alphanumeric double-byte characters have a leading byte of X'42' in EBCDIC and X'82' in ASCII. The trailing byte is an alphanumeric character. The valid COBOL subset of these is A-Z, a-z, 0-9.</td>
</tr>
<tr>
<td>EQA1821E</td>
<td>The DBCS representation of the hyphen was the first or last character in a DBCS name.</td>
<td>COBOL DBCS names cannot have a leading or trailing DBCS hyphen.</td>
</tr>
<tr>
<td>EQA1822E</td>
<td>A DBCS Name, DBCS literal or mixed SBCS/DBCS literal may not be continued.</td>
<td>A DBCS Name, DBCS literal or mixed SBCS/DBCS literal may not be continued.</td>
</tr>
<tr>
<td>EQA1823E</td>
<td>An end of line was encountered before the end of a DBCS name or DBCS literal.</td>
<td>An end of line was encountered before finding a closing shift-in control code. This message is for the System/370 environment.</td>
</tr>
<tr>
<td>EQA1824E</td>
<td>A DBCS literal or DBCS name contains no DBCS characters.</td>
<td>A shift-out shift-in pair of control characters were found with no intervening DBCS characters. This message is for the System/370 environment.</td>
</tr>
<tr>
<td>EQA1825E</td>
<td>End-of-source was encountered while processing a DBCS name or DBCS literal.</td>
<td>No closing Shift-In control code was found before end of file. This message is for the System/370 environment.</td>
</tr>
<tr>
<td>EQA1826E</td>
<td>A DBCS literal was not delimited by a trailing quote or apostrophe.</td>
<td>No closing quotation mark</td>
</tr>
<tr>
<td>EQA1827E</td>
<td>Invalid separator character found following a DBCS name.</td>
<td>A comma or semicolon must be specified following a DBCS name.</td>
</tr>
<tr>
<td>EQA1828E</td>
<td>Fixed binary constants are limited to 31 digits.</td>
<td>A fixed binary constant must be between −2<strong>31 and 2</strong>31 exclusive.</td>
</tr>
<tr>
<td>EQA1829E</td>
<td>Fixed decimal constants are limited to 15 digits.</td>
<td>A fixed decimal constant must be between −10<strong>15 and 10</strong>15 exclusive.</td>
</tr>
<tr>
<td>EQA1830E</td>
<td>Float binary constants are limited to 109 digits.</td>
<td>This limit applies to all PL/I FLOAT BINARY constants.</td>
</tr>
<tr>
<td>EQA1831E</td>
<td>Float decimal constants are limited to 33 digits.</td>
<td>This limit applies to all PL/I FLOAT DECIMAL constants.</td>
</tr>
<tr>
<td>EQA1832E</td>
<td>Floating-point exponents are limited to 3 digits.</td>
<td>This limit applies to all C float constants and to all PL/I FLOAT BINARY constants.</td>
</tr>
</tbody>
</table>
EQA1833E  Float decimal exponents are limited to 2 digits.
Explanation:  This limit applies to all PL/I FLOAT DECIMAL constants.

EQA1834E  Float binary constants must be less than 1E+252B.
Explanation:  This limit applies to all PL/I FLOAT BINARY constants.

EQA1835E  Float decimal constants must be less than 7.23700557733226221397318656304298E+75.
Explanation:  This limit applies to all PL/I FLOAT DECIMAL constants.

EQA1836E  Float constants are limited to 35 digits.
Explanation:  This limit applies to all C float constants.

EQA1837E  Float constants must be bigger than 5.39760534693402798664699142502496E-79 and less than 7.237005577332262213973186563042929E+75.
Explanation:  This is the range of values allowed by C.

EQA1838E  The module "module" cannot be loaded - it was NOT found.
Explanation:  The LOAD MODULE request failed, the module was not found.

EQA1839E  The module "module" cannot be deleted.
Explanation:  The DELETE MODULE request was unsuccessful.

EQA1840E  The module "module" was NOT loaded by the Debug Tool and therefore CANNOT be deleted.
Explanation:  Only modules loaded by the Debug Tool may be deleted.

EQA1841E  The module "module" cannot be loaded because it was already loaded by Debug Tool.
Explanation:  The LOAD MODULE request was unsuccessful.

EQA1842E  IDISNAP could not be loaded. Verify Fault Analyzer is available or loaded.
Explanation:  IDISNAP is a part of the product Fault Analyzer. Verify Fault Analyzer is installed properly.

Programmer response: Verify IDISNAP routine is available in the environment.

EQA1843E  %FA complete. See your Fault Analyzer history file.
Explanation:  %FA was complete.

Programmer response:  %FA was complete. See your Fault Analyzer history file.

EQA1844E  LOAD is not supported during initialization. STEP or GO, then retry the LOAD command.
Explanation:  You cannot issue a LOAD request until Debug Tool initialization is complete.

EQA1845I  An implicit LOAD was issued for module "loadmod".
Explanation:  An implicit LOAD command was issued for the specified load module as the result of a QUALIFY LOAD or QUALIFY CU command for a load module that was not currently loaded. A corresponding CLEAR LOAD will be issued when execution is resumed.

EQA1846I  An implicit CLEAR LOAD was issued for module "loadmod".
Explanation:  An implicit CLEAR LOAD command was issued for the specified load module. This module was previously loaded as the result of a QUALIFY LOAD or QUALIFY CU command for a load module that was not currently loaded.

EQA1847W  A load module that was loaded as the result of a LOAD command contains LE CUs but LE is not active. The LE CUs will not be created.
Explanation:  A LOAD command was used in a non-LE environment to load a load module that contained one or more LE CUs. These CUs cannot be created until LE is active. CUs will be created only for non-LE CUs in this load module.

EQA1848I  An implicit CU was created for "CU_name" in "loadmod".
Explanation:  An implicit CU was created for the specified CU in the specified load module as the result of a QUALIFY CU command for a load module that was already loaded and a COBOL CU that had not yet been executed. The CU will be destroyed when execution is resumed.

EQA1849I  Implicit CU "CU_name" in "loadmod" is being destroyed.
Explanation:  The specified implicitly created CU is being destroyed. This module was previously created as the result of a QUALIFY CU command for a CU that had not already been created.

EQA1850I  The CU CU_name cannot be destroyed.
Explanation:  The attempt to destroy the specified CU was unsuccessful.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA1872E</td>
<td>An error occurred while opening file; file name. The file may not exist, or is not accessible.</td>
</tr>
<tr>
<td>EQA1873E</td>
<td>An error occurred during an input or output operation.</td>
</tr>
<tr>
<td>EQA1874I</td>
<td>The command command name can be used only in full screen mode.</td>
</tr>
<tr>
<td>EQA1875I</td>
<td>Insufficient storage available.</td>
</tr>
<tr>
<td>EQA1876E</td>
<td>Not enough storage to display results.</td>
</tr>
<tr>
<td>EQA1877E</td>
<td>An error occurred in writing messages to the dump file.</td>
</tr>
<tr>
<td>EQA1878E</td>
<td>The cursor is not positioned at a variable name.</td>
</tr>
<tr>
<td>EQA1879E</td>
<td>The listing file name given is too long.</td>
</tr>
<tr>
<td>EQA1880E</td>
<td>You may not resume execution when the program is waiting for input.</td>
</tr>
<tr>
<td>EQA1881E</td>
<td>The INPUT command is only valid when the program is waiting for input.</td>
</tr>
<tr>
<td>EQA1882E</td>
<td>The logical record length for file name is out of bounds. It will be set to the default.</td>
</tr>
<tr>
<td>EQA1883E</td>
<td>Error closing previous log file; Return code = rc</td>
</tr>
<tr>
<td>EQA1884E</td>
<td>An error occurred during processing the source listing. Check return code return code in the Using the Debug Tool manual for more detail.</td>
</tr>
<tr>
<td>EQA1885I</td>
<td>Attempt to open INSPPREF failed. User did not specify the Preferences File TEST option and/or did not allocate INSPPREF.</td>
</tr>
<tr>
<td>EQA1886I</td>
<td>*** Global preferences file commands follow ***</td>
</tr>
<tr>
<td>EQA1887I</td>
<td>*** User preferences file commands follow ***</td>
</tr>
<tr>
<td>EQA1888I</td>
<td>*** Commands file commands follow ***</td>
</tr>
<tr>
<td>EQA1889I</td>
<td>*** Global preferences file commands end ***</td>
</tr>
<tr>
<td>EQA1890I</td>
<td>*** User preferences file commands end ***</td>
</tr>
<tr>
<td>EQA1891I</td>
<td>*** Commands file commands end ***</td>
</tr>
<tr>
<td>EQA1892I</td>
<td>Global Preferences file exists: file name</td>
</tr>
<tr>
<td>EQA1902W</td>
<td>The command has been terminated because of the attention request.</td>
</tr>
</tbody>
</table>

Explanation:
- EQA1872: An error during the initial processing (OPEN) of the file occurred.
- EQA1873: An error occurred performing an input or output operation.
- EQA1874: This command is one of a collection that is allowed only when your terminal is operating in full-screen mode. The function is not supported in line mode or in a batch mode.
- EQA1875: This message is issued when not enough storage is available to process the last command issued or to handle the last invocation.
- EQA1876: Increase size of virtual storage.
- EQA1877: This could be caused by a bad file name specified with the call dump FNAME option.
- EQA1878: Reposition the cursor and reenter.
- EQA1879: Under MVS, data definition names are limited to 8 characters and data set names are limited to 44 characters. If a partitioned data set is named, the member name must be specified (with up to 8 characters, enclosed in parentheses).
- EQA1880: The user attempted to issue a GOS/RUN or STEP request when the program was waiting for input. The input must be entered to resume execution.
- EQA1881: The user attempted to enter the INPUT command when the program was not waiting for any input.
- EQA1882: The logical record length is less than 32 bytes or greater than 256 bytes.
- EQA1883: The user attempted to open a new log file and the old one could not be closed; the new log file is used, however.
- EQA1884: An error occurred during processing of the list lines command. Possible return codes:
  1 - Debug Tool failed to open the log file.
  2 - The listing file could not be found or allocated.
  3 - The CU was not compiled with the correct compile option.
  4 - The CU was not compiled with the correct compile option.
  5 - The CU was not compiled with the correct compile option.
  6 - The CU was not compiled with the correct compile option.
  7 - Failed due to inadequate resources.
- EQA1885: If the Preferences File TEST option is not specified, the default is INSPPREF. The user did not specify the Preferences File TEST option, so Debug Tool assumes INSPPREF and tries to open it. If INSPPREF is not allocated, this open fails.
- EQA1886: Start of commands in the global preferences file.
- EQA1887: Start of commands in the user preferences file.
- EQA1888: Start of commands in the commands file.
- EQA1889: End of commands in the global preferences file.
- EQA1890: End of commands in the user preferences file.
- EQA1891: End of commands in the commands file.
- EQA1892: The global preferences file is opened successfully.
- EQA1902: The previously-executing command was terminated because of an attention request. Normal debugging can continue.
<table>
<thead>
<tr>
<th>Message Code</th>
<th>Message Text</th>
<th>Explanation</th>
<th>Programmer Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA1903E</td>
<td>An attention request has been issued. Enter QUIT to terminate Debug Tool or GO or RUN to resume execution.</td>
<td>The attention key was pressed three times because the application was looping either in system code or application code without debugging hooks. Only the GO/RUN and QUIT commands are valid at this point.</td>
<td></td>
</tr>
<tr>
<td>EQA1904E</td>
<td>The STEP and GO/RUN commands are not allowed at termination.</td>
<td>The STEP and GO/RUN commands are not allowed after the application program ends.</td>
<td></td>
</tr>
<tr>
<td>EQA1905W</td>
<td>You cannot trigger a condition in your program at this time.</td>
<td>The environment is in a position that it would not be meaningful to trigger a condition. For example, you have control during environment initialization.</td>
<td></td>
</tr>
<tr>
<td>EQA1906S</td>
<td>The condition named CONDITION name is unknown.</td>
<td>A condition name was expected, but the name entered is not the name of a known condition.</td>
<td></td>
</tr>
<tr>
<td>EQA1907W</td>
<td>The attempt to trigger this condition has failed.</td>
<td>For some reason, when Debug Tool tried to trigger the specified condition, it failed and the condition was not signaled.</td>
<td></td>
</tr>
<tr>
<td>EQA1918S</td>
<td>The block name block qualification block name is ambiguous.</td>
<td>There is another block that has the same name as this block.</td>
<td>Provide further block name qualification: by load module name, by compile unit name, or by additional block names if a nested block.</td>
</tr>
<tr>
<td>EQA1919E</td>
<td>The present block is not nested. You cannot QUALIFY UP.</td>
<td>While you can QUALIFY to any block, you cannot QUALIFY UP (for example, change the qualification to the block's parent) unless there really is a parent of that block. In this case, there is no parent of the currently-qualified block.</td>
<td>You have either misinterpreted your current execution environment or you have to qualify to some block explicitly.</td>
</tr>
<tr>
<td>EQA1920E</td>
<td>The present block has no dynamic parent. You cannot QUALIFY RETURN.</td>
<td>While you can QUALIFY to any block you cannot QUALIFY RETURN (for example, change the qualification to the block's invoker) unless there really is an invoker of that block. In this case, there is no invoker of the currently-qualified block.</td>
<td>You have either misinterpreted your current run-time environment or you have to qualify to some block explicitly.</td>
</tr>
<tr>
<td>EQA1921S</td>
<td>There is no block named block name.</td>
<td>The block that you named could not be located by Debug Tool.</td>
<td>Provide further block name qualification: by load module name, by compile unit name, or by additional block name(s) if a nested block.</td>
</tr>
<tr>
<td>EQA1922S</td>
<td>There is no block named block name within block block qualification.</td>
<td>The qualification you are using (or the spelling of the block names) prevented Debug Tool from locating the target block.</td>
<td>Verify that the named block should be within the current qualification.</td>
</tr>
<tr>
<td>EQA1923S</td>
<td>There is no compilation unit named cu name.</td>
<td>The compilation unit (program) that you named could not be located by Debug Tool.</td>
<td></td>
</tr>
<tr>
<td>EQA1924S</td>
<td>Statement statement id is not valid.</td>
<td>The statement number does not exist or cannot be used. Note that the statement number could exist but is not used.</td>
<td></td>
</tr>
<tr>
<td>EQA1925S</td>
<td>There is no load module named load module name.</td>
<td>Load module qualification is referring to a load module that cannot be located.</td>
<td>The load module might be missing or it might have been loaded before Debug Tool was first used. On the System/370, Debug Tool is aware of additional load modules only if they were FETCHed after Debug Tool got control for the first time.</td>
</tr>
<tr>
<td>EQA1926S</td>
<td>There is no cu named cu name within load module load module name.</td>
<td>The compilation unit might be misspelled or missing.</td>
<td></td>
</tr>
<tr>
<td>EQA1927S</td>
<td>There are number CUs named cu name, but neither belongs to the current load module.</td>
<td>The compilation unit you named is not unique.</td>
<td>Add further qualification so that the correct load module will be known.</td>
</tr>
<tr>
<td>EQA1928S</td>
<td>The block name block name is ambiguous.</td>
<td>There is another block that has the same name as this block.</td>
<td>Provide further block name qualification: by load module name, by compile unit name, or by additional block names if a nested block.</td>
</tr>
</tbody>
</table>
EQA1929S  Explicit qualification is required because the location is unknown.

Explanation:  The current location is unknown; as such, the reference or statement must be explicitly qualified.

Programmer response:  Either explicitly set the qualification using the SET QUALIFY command or supply the desired qualification to the command in question.

EQA1930S  There is no compilation unit named CU-name in the current enclave.

Explanation:  The compilation unit (program) that you named could not be located in the current enclave by Debug Tool.

EQA1931S  There is no cu named CU-name within load module load module name in the current enclave.

Explanation:  The compilation unit might be misspelled or missing, or it might be outside of the current enclave.

EQA1932S  Block or CU block_name is not currently available

Explanation:  The block or CU that you named could not be located by Debug Tool.

Programmer response:  Provide further block name qualification—by load module name, by compile unit name, or by additional block names(s) if a nested block.

EQA1933W  The program was compiled with the OPTIMIZE compiler option either by a release of the COBOL compiler that does not support debugging of optimized code, or by a release of the compiler that is missing the required service for debugging of optimized code. Until it is recompiled with the proper release and service level of the COBOL compiler, results of Debug Tool commands are unpredictable.

Explanation:  Debug Tool does not have accurate information about the program, and thus cannot provide reliable results.

Programmer response:  Recompile the program with one of the following or later versions of the COBOL compiler:

- COBOL FOR OS/390 & VM Version 2 with APAR PQ63234 installed
- Enterprise COBOL for z/OS and OS/390 Version 3 Release 1 with APAR PQ63235 installed
- Enterprise COBOL for z/OS and OS/390 Version 3 Release 2

EQA1934E  The assignment was not performed because the assigned value might not be used by the program, due to optimization.

Explanation:  Results are unreliable, because the program might use the previous value that was saved in temporary storage or a register.

Programmer response:  You can SET WARNING OFF to allow the update to take place or recompile the program without optimization.

EQA1935E  Data Item variable name was discarded due to optimization.

Explanation:  The program was compiled with the OPTIMIZE(FULL) option, and the compiler discarded the data item because it was not referenced in the program.

EQA1936W  The assignment was performed but the assigned value might not be used by the program, due to optimization.

Explanation:  Results might be unreliable because the program might use the previous value that was saved in temporary storage or a register.

Programmer response:  Recompile the program without the Optimize option.

EQA1937W  This breakpoint is deferred.

Explanation:  The compilation unit (program) that you specified could not be located by the Debug Tool. The breakpoint is deferred until this CU is entered.

EQA1938W  Provide a CU (Program) Name to qualify the block name.

Explanation:  The CU name (Program) must be added to the block name to allow Debug Tool to locate the block named.

EQA1940E  variable name is a not a level-one identifier.

Explanation:  You are trying to clear an element of a structure. You must clear the entire structure by naming its level-one identifier.

EQA1941E  ATANH(x) is undefined if x is REAL and ABS(x) >= 1.

Explanation:  This applies to the PL/I ATANH built-in function.

EQA1942E  LOG(z) is undefined if z is COMPLEX and z = 0.

Explanation:  This applies to the PL/I LOG built-in function.

EQA1943E  built-in function (x) is undefined if x is REAL and x <= 0.

Explanation:  This applies to the PL/I LOG, LOG2 and LOG10 built-in functions.

EQA1944E  built-in function (x,y) is undefined if x=0 and y=0.

Explanation:  This applies to the PL/I ATAN and ATAND built-in functions.

EQA1945I  There are no variables in the statement to display.

Explanation:  The current statement has no variables.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA1946I</td>
<td>The variable is too big to be displayed. Resources (for example, heap storage) are not available.</td>
<td>The command could not be completed due to inadequate resources. The variable is too big.</td>
</tr>
<tr>
<td>EQA1949E</td>
<td>The EQALANGX file does not match the object for Compile Unit name. The EQALANGX file cannot be used.</td>
<td>An EQALANGX file containing the assembler debugging information does not match the object. Either the CSECT length is different, selected object code is different, or the EQALANGX file is not for the correct language.</td>
</tr>
<tr>
<td>EQA1950E</td>
<td>The MONITOR table is empty. If the first MONITOR command entered is numbered, it must have number 1.</td>
<td>A MONITOR n command was issued when the MONITOR table is empty, but n is greater than 1.</td>
</tr>
<tr>
<td>EQA1951E</td>
<td>The number of entries in the MONITOR table is monitor-number. New MONITOR commands must be unnumbered or have a number less than or equal to monitor-number.</td>
<td>A MONITOR n command was issued but n is greater than 1 plus the highest numbered MONITOR command.</td>
</tr>
<tr>
<td>EQA1952E</td>
<td>The MONITOR command table is full. No unnumbered MONITOR commands will be accepted.</td>
<td>A MONITOR command was issued but the MONITOR table is full.</td>
</tr>
<tr>
<td>EQA1953E</td>
<td>No command has been set for MONITOR monitor-number.</td>
<td>A LIST MONITOR n or CLEAR MONITOR n command was issued, but n is greater than the highest numbered MONITOR command.</td>
</tr>
<tr>
<td>EQA1954E</td>
<td>The command for MONITOR monitor-number has already been cleared.</td>
<td>A CLEAR MONITOR n command was issued, but MONITOR has already been cleared.</td>
</tr>
<tr>
<td>EQA1955E</td>
<td>There are no MONITOR commands established.</td>
<td>A LIST MONITOR or CLEAR MONITOR command was issued, but there are no MONITOR commands established.</td>
</tr>
<tr>
<td>EQA1956E</td>
<td>No previous FIND argument exists. FIND operation not performed.</td>
<td>A FIND command must include a string to find when no previous FIND command has been issued.</td>
</tr>
<tr>
<td>EQA1957E</td>
<td>String could not be found.</td>
<td>A FIND attempt failed to find the requested string.</td>
</tr>
<tr>
<td>EQA1958E</td>
<td>The requested SYSTEM command could not be run.</td>
<td>A SYSTEM command was issued. The underlying operating system received it but did not process it successfully.</td>
</tr>
<tr>
<td>EQA1959E</td>
<td>The requested SYSTEM command was not recognized.</td>
<td>The underlying operating system was passed a command that was not recognized. The system could not process the command.</td>
</tr>
<tr>
<td>EQA1960S</td>
<td>There is an error in the definition of variable variable name. Attribute information cannot be displayed.</td>
<td>The specified variable has an error in its definition or length and address information is not currently available in the execution of the program.</td>
</tr>
<tr>
<td>EQA1961E</td>
<td>Automonitor cannot be removed or replaced. Use SET AUTOMONITOR OFF command.</td>
<td>The Automonitor can only be set off with the SET AUTOMONITOR OFF command.</td>
</tr>
<tr>
<td>EQA1962E</td>
<td>Automonitor is already set off.</td>
<td>The Automonitor function is already off.</td>
</tr>
<tr>
<td>EQA1963S</td>
<td>The command command is not supported on this platform.</td>
<td>The given command is not supported on the current platform.</td>
</tr>
<tr>
<td>EQA1964E</td>
<td>Source or Listing data is not available, or the CU was not compiled with the correct compile options.</td>
<td>The source or listing information is not available. Some of the possible conditions that could cause this are: The listing file could not be found, the CU was not compiled with the correct compile options, inadequate resources were available. When using CICS this condition could occur because one of more of the following TDQueues are not defined: • CINL (for source and listing support) • CIGZ (for COBOL side file support) • CIBM (for Enterprise PL/I side file support)</td>
</tr>
<tr>
<td>EQA1965E</td>
<td>Attributes of source of assignment statement conflict with target variable name. The assignment cannot be performed.</td>
<td>The assignment contains incompatible data types; the assignment cannot be made.</td>
</tr>
</tbody>
</table>
EQA1966E  The AREA condition would have been raised during an AREA assignment, but since WARNING is on, the assignment will not be performed.
Explanation: The operation, if performed, would result in the AREA condition. The condition is being avoided by rejecting the operation.

EQA1967E  The subject of the built-in function name pseudovariable (character string) must be complex numeric.
Explanation: You are trying to get apply the PL/I IMAG or REAL pseudovariable to a variable that is not complex numeric.

EQA1968W  You cannot use the GOTO command at this time.
Explanation: The program environment is such that a GOTO cannot be performed correctly. For example, you could be in control during environment initialization and base registers (supporting the GOTO logic) have not been established yet.

EQA1969E  GOTO label-constant or JUMPTO label-constant will not be permitted because that constant is the label for a FORMAT statement.
Explanation: There are several statement types that are not allowable as the target of a GOTO or JUMPTO command. FORMAT statements are one of them.

EQA1970E  The 3-letter national language code national language is not supported for this installation of Debug Tool. Uppercase United States English (UEN) will be used instead.
Explanation: The national-language-specified conflicts with the supported national languages for this installation of Debug Tool. Verify that the Language Environment run-time NATLANG option is correct.

EQA1971E  The return code in the QUIT command must be nonnegative and less than 1000.
Explanation: For PL/I, the value of the return code must be nonnegative and less than 1000.

EQA1972E  variable name is not a LABEL constant No AT commands can be issued against it
Explanation: LABEL variables cannot be the object of the AT command.

EQA1973E  The FIND argument cannot exceed a string length of 64
Explanation: Shorten the search argument to a string length 64 or less.

EQA1974E  The FIND argument is invalid, the string length is zero
Explanation: Supply a search argument inside the quotes.

EQA1975E  SYSDEBUG/SEPARATE file can not be found for Compile_Unit_name which was compiled with SEPARATE compile option but the debug file containing the debugging tables and the listing created by the compiler can not be found. Use the Set Source command to indicate the new location of the SYSDEBUG/SEPARATE file.
Explanation: The Debug File containing the listing and the debugging tables can not be found. Some of the possible conditions that could cause this are: The Debug File was deleted from the system, or the user does not have authorization to access the debug file.

EQA1976E  The debug information for Compile_Unit_name has already been validated, changing the debug file is not allowed. The command will not be performed.
Explanation: A Debug File containing the listing and the debugging tables has already been validated.

EQA1977E  The Debug File creation date does not match the object for Compile_Unit_name, but further validation showed that debug data in the file can still be used.
Explanation: A Debug File containing the listing and the debugging tables does not match the creation date of the object.

EQA1978E  The Debug File creation date does not match the object for Compile_Unit_name. The Debug file can not be used.
Explanation: A Debug File containing the listing and the debugging tables does not match the creation date of the object, and the data it contains is not valid.

EQA1979E  The Debug File for Compile_Unit_name is not available or was not found.
Explanation: The Debug File was nowhere to be found.

EQA1980E  The Debug File for Compile_Unit_name could not be opened or read.
Explanation: I/O errors when trying to open/read Debug File.

EQA1981E  Invalid mode name, transaction program name, or partner LU name associated with symbolic_destination_name. Mode_name= mode_name and partner_LU_name= partner_LU_name
Explanation: A conversation allocation request failed due to invalid conversation characteristics obtained from the APPC/MVS side information file. There could be several reasons for this:
1. The mode_name characteristic specifies a mode name that is either not recognized by the LU as valid or is reserved for SNA service transaction programs.
2. The TP_name characteristic specifies an SNA service transaction program name.
3. The partner_LU_name characteristic specifies a partner LU name that is not recognized by the LU as being valid.
Programmer response: Contact your APPC/MVS system administrator to modify the characteristics associated with the given symbolic_destination_name in the side information file. For
information about the recommended values for mode_name and 
TP_name, see the CODE/370 Installation manual. The OS/2 system error log can contain valuable diagnostic information. To 
access the system error log, select System Error Log from the 
FFST/2 folder or type SYSLOG at the OS/2 command line.

**EQA1982E**  
Permanent conversation allocation failure for 
symbolic_destination_name, Partner LU_name= 
partner LU_name and mode_name= mode_name.

Explanation: The conversation cannot be allocated because of a 
condition that is not temporary. There could be several reasons for 
this:
1. The workstation where the partner LU_name is defined is 
turned off or Communications Manager/2 is not started.
2. The partner LU_name has not been defined.
3. The current session limit for the specified partner LU_name and 
mode_name pair is zero.
4. A system definition error or a session-activation protocol error 
has occurred.

Programmer response: Ensure that you specified the correct 
symbolic_destination_name or contact your APPC/MVS system 
administrator to correct the condition. The OS/2 system error log 
can contain valuable diagnostic information. To access the system 
error log, select System Error Log from the FFST/2 folder or type SYSLOG at the OS/2 command line.

**EQA1983E**  
Temporary conversation allocation failure for 
symbolic_destination_name, Partner LU_name= 
partner LU_name and mode_name= mode_name.

Explanation: The conversation cannot be allocated because of a 
condition that might be temporary. There could be several reasons for 
this:
1. Undefined mode_name (not temporary)
2. Temporary lack of resources at the host LU or workstation LU.

Verify that mode_name is defined on the target workstation using 
the CM/2 Communication Manager Setup panels. If mode_name is 
defined on the workstation, contact your MVS/ESA system 
programmer to ensure that mode_name is also defined on the MVS system. The OS/2 system error log can contain valuable diagnostic 
information. To access the system error log, select System Error Log from the FFST/2 folder or type SYSLOG at the OS/2 command line.

**EQA1984E**  
The workstation transaction program is 
permanently unavailable at 
symbolic_destination_name, Partner LU_name= 
partner LU_name.

Explanation: Partner LU_name rejected the allocation request 
because the host program specified a workstation program that 
partner LU_name recognizes but it cannot start. There could be 
several reasons for this:
1. Missing transaction program definition on the workstation.
2. Invalid OS/2 program path and file name specified in the 
transaction program definition.

Programmer response: Define the transaction program on the 
workstation or ensure that the transaction program definition is correct. The symbolic_destination_name can be used to obtain the 
workstation transaction program name from the APPC/MVS side 
information table. For information about the recommended values 
for TP_name, see the CODE/370 Installation manual. The OS/2 system error log can contain valuable diagnostic information. To 
access the system error log, select System Error Log from the 
FFST/2 folder or type SYSLOG at the OS/2 command line.

**EQA1985E**  
Unrecognized transaction program name at 
symbolic_destination_name, Partner LU_name= 
partner LU_name.

Explanation: Partner LU_name rejected the allocation request 
because the host program specified a workstation TP_name that 
partner LU_name does not recognize. The transaction program 
definition is missing on the workstation.

Programmer response: Define the transaction program on the 
workstation. The symbolic_destination_name can be used to obtain the 
workstation transaction program name from the APPC/MVS side 
information table. For information about the recommended values 
for TP_name, see the CODE/370 Installation manual. The OS/2 system error log can contain valuable diagnostic information. To access the system error log, select System Error Log from the FFST/2 folder or type SYSLOG at the OS/2 command line.

**EQA1986E**  
Unexpected TCP/IP error: Modules= module_name, 
Locations= location_id, TCP/IP call= call_type, 
return_code= rc.

Explanation: The host communications code received an 
unexpected return code from a TCP/IP call. The information 
displayed is for diagnostic purposes.
• module_name is the name of the communications module issuing 
the TCP/IP call
• location_id is an internal three-digit identifier for the TCP/IP call 
within the module
• call_type is the TCP/IP call type (for example, CONNECT or 
SHUTDOWN)
• rc is the unexpected return code that is displayed in decimal 
format

Programmer response: For remote debug mode, you need to 
provide the correct TCP/IP address and/or port number of the 
workstation.

**EQA1987E**  
Debugger terminated, execution continues.

Explanation: The initialization of the LU 6.2 conversation 
between the host and the workstation (in a batch process) has 
failed. The debugger is terminated and the execution of the batch 
application continues. Note the accompanying messages as to 
possible causes.

**EQA1988E**  
Severe internal error. PWS Debug Tool 
terminated.

Explanation: PWS Debug Tool detected a severe internal error. It 
has been shutdown.

Programmer response: Diagnostic information was recorded in 
either the EVFERROR.LOG or the EQALu62.LOG. The path where 
these logs are stored is in the CODETMPDIR environment variable 
in CONFIG.SYS.

**EQA1989E**  
Invalid session ID - session_ID.

Explanation: Conversation initialization failed due to an invalid 
session ID in the Session Parameter. There could be several reasons 
for this,
1. The session ID is longer than 8 characters or contains invalid 
characters. Valid session IDs consist of 1-8 alphanumeric 
characters.
2. There is already another PWS Debug Tool session with the
given session ID.

Programmer response: Diagnostic information is recorded in
either the EVFERROR.LOG or the EQALU62.LOG. The path where
these logs are stored is in the CODETMPDIR environment variable
in CONFIG.SYS. If there is already an existing PWS Debug Tool
session with the given session ID then a different session ID must
be provided for concurrent debug sessions on the same
workstation. If a session ID is not specified, it defaults to
CODEDT. For a description of the Session Parameter and its
contents, see the Debug Tool manual.

---

Table 7. Definitions for error_type, insert1, insert2, and
insert3

<table>
<thead>
<tr>
<th>error_type</th>
<th>insert1</th>
<th>insert2</th>
<th>insert3</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPL or INQUIRE RPL</td>
<td>gnnencomm where: gg - CaFlag, nn - CsFunc, oo - CSFlagO0, and mm - CsRegMod</td>
<td>xtrnxxx where: n-R15,r-R0, or n-Rn-RPLSense if RtnCodeFdBk2</td>
<td>pppaXXX where: pp-RPLCode, dd-RmCode, and hkkk-FdBk2</td>
</tr>
<tr>
<td>LOGMODE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

ACB | gnnencomm where: gg - CaFlag, nn - CsFunc, oo - CSFlagO0, and mm - CsRegMod | Error Offset in EQAYVTAM | ACBERR |

---

MODCB | gnnencomm where: gg - CaFlag, nn - CsFunc, oo - CSFlagO0, and mm - CsRegMod | Error Offset in EQAYVTAM | xxxxyyz where: xxx-0000, yy-MODCB R0, and zz-MODCB R15 |

---

This message is issued whenever a permanent error is detected
communicating with the VTAM terminal. A terminal condition is
then signaled to LE causing program termination.
### Table 7. Definitions for error_type, insert1, insert2, and insert3 (continued)

<table>
<thead>
<tr>
<th>error_type</th>
<th>insert1</th>
<th>insert2</th>
<th>insert3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic</td>
<td>ggmcomm where: gg - CsFlag, nn - CsFunc, oo - CSFlagOr, and mm - CSReqMod</td>
<td>Error Offset in EQAYVTAM</td>
<td>0</td>
</tr>
<tr>
<td>Function</td>
<td>ggmcomm where: gg - CsFlag, nn - CsFunc, oo - CSFlagOr, and mm - CSReqMod</td>
<td>Error Offset in EQAYVTAM</td>
<td>EQAYVTAM function code</td>
</tr>
<tr>
<td>Storage</td>
<td>ggmcomm where: gg - CsFlag, nn - CsFunc, oo - CSFlagOr, and mm - CSReqMod</td>
<td>Error Offset in EQAYVTAM</td>
<td>0</td>
</tr>
<tr>
<td>Undefined LU</td>
<td>ggmcomm where: gg - CsFlag, nn - CsFunc, oo - CSFlagOr, and mm - CSReqMod</td>
<td>Error Offset in EQAYVTAM</td>
<td>0</td>
</tr>
<tr>
<td>Unknown</td>
<td>ggmcomm where: gg - CsFlag, nn - CsFunc, oo - CSFlagOr, and mm - CSReqMod</td>
<td>Error Offset in EQAYVTAM</td>
<td>0</td>
</tr>
</tbody>
</table>

**Programmer response:** If an ACB error is reported, check with your installer to ensure that the VTAM modifications required by Debug Tool have been made. If Undefined LU error is reported, check the MFI operand of the TEST parameter to ensure that the correct VTAM terminal Logical Unit identifier was specified and that the terminal in known to VTAM. Otherwise, contact IBM support.

**EQA1999E** Load module loadmod_name could not be found.

**Explanation:** The indicated load module was specified as an operand of the DESCRIBE LOADMODS command but is not a active load module.

**EQA2001E** Ambiguous conversion between "&1" and "&2".

**Problem determination:** (*where* &1 is a C and C++ type &2 is a C and C++ type)

**Explanation:** The debugger was not able to find a single type common to the two specified types and was therefore unable to convert from one to the other.

**Programmer response:** Explicitly cast the type to an intermediate type and then convert to requested type.

**EQA2002E** A return value is not allowed for this function.

**Explanation:** A function with a return type of "void" cannot return a value.

**Programmer response:** Remove the value or expression from the return statement, remove the return statement, or change the return type of the function.

**EQA2003E** Identifier "&1" is undefined.

**Problem determination:** (*where* &1 is a C and C++ name)

**Explanation:** The specified identifier is used but has not been defined.

**Programmer response:** Define the identifier before using it. Check its spelling. If the identifier has been defined in a header file, check that any required macros have been defined.

**EQA2004E** &1 member "&2" cannot be accessed.

**Problem determination:** (*where* &1 is the keyword "private" or "protected" &2 is a class member name)

**Explanation:** The specified member is private, protected, or is a member of a private base class and cannot be accessed from the current scope.

**Programmer response:** Check the access specification rules for the member function and change the access specifier if necessary. If the member function belongs to a base class, check the access specifier of the base class where the current class is defined.

**EQA2005E** Return value of type "&1" is expected.

**Problem determination:** (*where* &1 is a C and C++ type)

**Explanation:** No value is returned from the current function, but the function is expecting a nonvoid return value. The function was declared with a return type but the debugger did not detect a return statement. Only functions with a void return type can have no return statement or have a return statement with no return value.

**Programmer response:** Return a value from the function or change the function’s return type to void.

**EQA2006E** "&1" cannot be made a &2 member.

**Problem determination:** (*where* &1 is a class member name &2 is the keyword "public", "protected" or "private")

**Explanation:** An attempt is made to give private access to a base class member or to give an access that is different from the access the member was declared with. A derived class can only change the access of a base class member to public or protected if the access of that member was not private in the base class.

**Programmer response:** Remove the invalid access statement or change the access specifier in the base class.

**EQA2007E** The array boundary in "&1" is missing.

**Problem determination:** (*where* &1 is a C and C++ type)

**Explanation:** An array must be defined with at least one element. Use a pointer if you want to dynamically allocate memory for the array.

**Programmer response:** Add an array bound.
EQA2008E • EQA201E

EQA2008E  The bit-field length must be an integral constant expression.
Explanation: The bit-field length, which is the value to the right of the colon, must be an integer. A constant expression has a value that can be determined during compilation and does not change during execution.
Programmer response: Change the bit-field length to an integral constant expression.

EQA2009E  "&1" is not a base class of "&2".
Problem determination: (where &1 is a class name &2 is a class name)
Explanation: A derived class attempted to access elements of a class it did not inherit from. A derived class can only access elements of its base class or base classes.
Programmer response: Ensure the class names are correct and the classes are derived properly.

EQA2010E  The array bound must be a positive integral constant expression.
Explanation: The debugger detected an array declaration that did not have a constant that is greater than 0 for the array bounds. Use pointers if you want to dynamically allocate storage for arrays.
Programmer response: Change the array bound to an integral constant expression or change it to a pointer. A constant expression has a value that can be determined during compilation and does not change during execution.

EQA2011E  "&1" has the same name as its containing class.
Problem determination: (where &1 is a C++ name)
Explanation: The debugger has detected conflicting names for objects within a class declaration. Nested class declarations must have different names.
Programmer response: Change the name of the conflicting class.

EQA2012E  A destructor can only be used in a function declaration or in a function call.
Explanation: The debugger detected an incorrect destructor call.
Programmer response: Check the call to the destructor to ensure no braces are missing. If the braces are correct, remove the destructor call.

EQA2013E  An initializer is not allowed for "&1".
Problem determination: (where &1 is a C and C++ name or keyword)
Explanation: The debugger detected an initializer where one is not allowed. For example, a class member declarator cannot contain an initializer.
Programmer response: Remove the initializer.

EQA2014E  The string must be terminated before the end of the line.
Explanation: The debugger detected a string that was not terminated before an end-of-line character was found.
Programmer response: End the string before the end of the line, or use "\" to continue the string on the next line. The "\" must be the last character on the line.

EQA2015E  An expression of type "&1" cannot be followed by the function call operator ()
Explanation: The debugger detected an expression followed by the function call operator. The expression must be of type function, pointer to function, or reference to function.
Programmer response: Change the type of expression or remove the function call operator.

EQA2016E  The "this" keyword is only valid in class scope.
Explanation: An attempt to use the C++ keyword this was detected outside class scope. The keyword this cannot be used outside a class member function body.
Programmer response: Remove or move the this keyword.

EQA2017E  A destructor cannot have arguments.
Programmer response: Remove the arguments from the destructor.

EQA2018E  A declaration has been made without a type specification.
Explanation: The debugger detected a typedef specification that did not have a type associated with it.
Programmer response: Add a type specification to the declaration.

EQA2019E  Class qualification for "&1" is not allowed.
Problem determination: (where &1 is a C++ name)
Explanation: Explicit class qualification is not allowed in this context.
Programmer response: Remove the class qualification.

EQA2020E  The "&1" operator is not allowed between "&2" and "&3".
Problem determination: (where &1 is a C and C++ type &2 is a C and C++ type &3 is a C and C++ type)
Explanation: The debugger detected an illegal operator between two operands. For user-defined types, you must overload the operator to accept the user-defined types.
Programmer response: Change the operator or change the operands.

EQA2021E  "&1" cannot be converted to "&2".
Problem determination: (where &1 is a C and C++ type &2 is a C and C++ type)
Explanation: The type conversion cannot be performed because there is no conversion between the types. This can occur in an
initialization, assignment, or expression statement.

**Programmer response:** Change one of the types or overload the operator.

---

**EQA2022E**  
Operand for "&1" must be a pointer or an array.

**Problem determination:** (where &1 is a C and C++ operator)

**Explanation:** The specified operator must have an operand that is a pointer or an array.

**Programmer response:** Change the operand to either a pointer or an array.

---

**EQA2023E**  
Syntax error - "&1" is not a class name.

**Problem determination:** (where &1 is a C++ name)

**Explanation:** A class name must be specified in this context.

**Programmer response:** Specify a class name. Check the spelling.

---

**EQA2024E**  
Operand of "&1" operator must be an lvalue.

**Problem determination:** (where &1 is a C and C++ operator)

**Explanation:** The debugger detected an operand that is not an lvalue. An lvalue is an expression that represents an object. For example, the left hand side of an assignment statement must be an lvalue.

**Programmer response:** Change the operand to an lvalue.

---

**EQA2025E**  
const expression cannot be modified.

**Explanation:** You can initialize a const object, but its value cannot change afterwards.

**Programmer response:** Eliminate the const type qualifier from the expression or do not use it with the increment/decrement operators.

---

**EQA2026E**  
An expression of type "&1" is not allowed on the left side of "&2&3".

**Problem determination:** (where &1 is a C and C++ type &2 is a C and C++ operator &3 is a C and C++ name)

**Explanation:** The debugger detected a mismatch between the operands of an operator.

**Programmer response:** Change the operand type or use a different operator.

---

**EQA2027E**  
"&1" is neither an immediate base class nor a nonstatic data member of class "&2".

**Problem determination:** (where &1 is a C++ name)

**Explanation:** The debugger has detected an element of the initializer list that is not an element of the member list. In the constructor initializer list, you can only initialize immediate base classes and data members not inherited from a base class.

**Programmer response:** Change the constructor initializer list.

---

**EQA2028E**  
Constructor initializer list is not allowed for nonconstructor function.

**Explanation:** An attempt is being made to give a constructor initializer list to a nonconstructor function. A constructor initializer list is only allowed for a constructor function.

**Programmer response:** Remove the constructor initializer list.

---

**EQA2029E**  
Variable "&1" is not allowed in an argument initializer.

**Problem determination:** (where &1 is a C++ name)

**Explanation:** The debugger has detected a default argument initialized by a parameter.

**Programmer response:** Remove the parameter from the default argument initialization.

---

**EQA2030E**  
There are too many initializers in the initializer list.

**Explanation:** The debugger detected more initializers than were present in the function declaration.

**Programmer response:** Remove one or more initializers from the initializer list. Make sure the number of initializers in the initializer list corresponds to the number of arguments in the function declaration.

---

**EQA2031E**  
An initializer is not allowed for an array allocated by "new".

**Programmer response:** Remove the initializer or remove the "new" allocation.

---

**EQA2032E**  
The bit-field length must not be more than &1.

**Problem determination:** (where &1 is a number)

**Explanation:** The bit-field length must not exceed the maximum bit size of the bit-field type.

**Programmer response:** Reduce the bit-field length.

---

**EQA2033E**  
The type of "&1" cannot be "&2".

**Problem determination:** (where &1 is a C++ construct &2 is a C++ type)

**Explanation:** The debugger detected a conflict in a type declaration.

**Programmer response:** Change the type.

---

**EQA2034E**  
Function overloading conflict between "&1" and "&2".

**Problem determination:** (where &1 is a function type &2 is a function type)

**Explanation:** The debugger detected function argument types that did not match.

**Programmer response:** Change the argument declarations of the functions.
EQA2035E  •  EQA2046E

EQA2035E  •  Declarations of the same &1 must not specify default initializers for the same argument.

Problem determination:  (where &1 is the word "function" or the keyword "template")

Explanation:  The debugger has detected a duplicate default initializer value for the same argument in both overloaded functions or in both templates.

Programmer response:  Ensure that you wanted to declare the same function or template, if that is the case, remove one of the default initializers. Otherwise, remove one of the declarations or overload the function.

EQA2036E  •  Call does not match any argument list for "&1".

Problem determination:  (where &1 is a function name)

Explanation:  No variant of the overloaded function matches the argument list. The argument mismatch could be by type or number of arguments.

Programmer response:  Change the argument list on the call to the overloaded function or change the argument list on one of the overloaded function variants so that a match is found.

EQA2037E  •  Call to "&1" matches more than one function.

Problem determination:  (where &1 is a function name)

Explanation:  More than one variant of the overloaded function matches equally well with the argument list specified on the call.

Programmer response:  Change the argument list on the call to the overloaded function or change the argument list on one of the overloaded function variants so that only one match is found.

EQA2038E  •  The "operator" declaration must declare a function.

Explanation:  The keyword "operator" can only be used to declare an operator function.

Programmer response:  Check the declaration of the operator and make sure the function declarator () appears after it. Use the "operator" keyword to declare an operator function or remove it.

EQA2039E  •  Operand for "&1" is of type "&2" that is not of type pointer to member.

Problem determination:  (where &2 is a C++ type)

Explanation:  The specified operator must have an operand that is of type pointer to member.

Programmer response:  Change the operand to type pointer to member.

EQA2040E  •  "&1" is not allowed as a function return type.

Problem determination:  (where &1 is a C and C++ type)

Explanation:  You cannot declare a function with a function or an array as its return type.

Programmer response:  Declare the function to return a pointer to the function or the array element type.

EQA2041E  •  "&1" is not allowed as an array element type.

Problem determination:  (where &1 is a C and C++ type)

Explanation:  The declaration of an array of functions or references, or an array of type void is not valid.

Programmer response:  Remove the declaration or change the declaration so that it is an array of pointer to functions, pointers to references, or pointers to void.

EQA2042E  •  const variable "&1" does not have an initializer.

Problem determination:  (where &1 is a variable name)

Explanation:  You can only assign a value to a const variable using an initializer. This variable has no initializer, so it can never be given a value.

Programmer response:  Initialize the variable or remove the const keyword.

EQA2043E  •  Nonstatic member "&1" must be associated with an object or a pointer to an object.

Problem determination:  (where &1 is a class member name)

Explanation:  The debugger detected a nonstatic member making a reference to an object that has not been instantiated. You can reference only static members without associating them with an instance of the containing class.

Programmer response:  Check the spelling and the class definition. Change the name of the class or function, or define the function as static in that class.

EQA2044E  •  "&1" is not a member of "&2".

Problem determination:  (where &1 is a C++ name &2 is a class name)

Explanation:  The class is used explicitly as the scope qualifier of the member name, but the class does not contain a member of that name.

Programmer response:  Check the spelling of the scope qualifier. Change the scope qualifier to the class containing that member, or remove it.

EQA2045E  •  Wrong number of arguments for "&1".

Problem determination:  (where &1 is a function or type name)

Explanation:  A function or an explicit cast has been specified with the wrong number of arguments.

Programmer response:  Use the correct number of arguments. Ensure that overloaded functions have the correct number and type of arguments.

EQA2046E  •  "&1" must be a class member.

Problem determination:  (where &1 is a C++ name)

Explanation:  Conversion functions and certain operator functions must be class members. They cannot be defined globally.

Programmer response:  Remove the global definition or make the function a class member.
EQA2047E  An argument type of "&1" is not allowed for "&2".

Problem determination: (where &1 is a C and C++ type &2 is a function name)

Explanation: The function being declared has restrictions on what types its arguments can have. The specified type is not allowed for this argument.

Programmer response: Change the argument type.

EQA2048E  "&2" cannot have a return type of "&1".

Problem determination: (where &1 is a C++ type &2 is an operator function)

Explanation: The specified operator function has the wrong return type.

Programmer response: Change the return type.

EQA2049E  The array operator must have one operand of pointer type and one of integral type.

Explanation: This error can result from the incorrect use of the array operator.

Programmer response: Change the operands of the array operator.

EQA2050E  Wrong number of arguments specified in the function call.

Explanation: The number of arguments in the function call does not match the number of arguments in the function declaration.

Programmer response: Ensure the function declaration and function call specify the same number of arguments.

EQA2051E  "&1" operator is not allowed for type "&2".

Problem determination: (where &1 is a C and C++ operator &2 is a C and C++ type)

Explanation: The specified operator cannot be used with operands of this type.

Programmer response: Change either the operator or the operands.

EQA2052E  Syntax error - expected "&1" and found "&2".

Problem determination: (where &1 is a C++ token &2 is a C++ token)

Explanation: A syntax error was found while parsing the expression. The message identifies what the debugger expected and what it actually found. Often the source of the error is an unmatched parenthesis or a missing semicolon.

Programmer response: Correct the syntax.

EQA2053E  "&1" is not allowed for &2.

Problem determination: (where &1 is a keyword &2 is a C++ construct)

Explanation: The attribute or name cannot be specified in the given context. The debugger detected incompatible names that conflict with the language definition.

Programmer response: Change the ambiguous reference in the conversion.

EQA2054E  "&1" conflicts with previous "&2" declaration.

Problem determination: (where &1 is a keyword &2 is a keyword)

Explanation: The declaration conflicts with a previous declaration of the same symbol.

Programmer response: Remove one of the declarations or make them identical.

EQA2055E  The "operator->" function must return a class type that contains an "operator->" function.

Explanation: The "operator->" function must return either a class type, a reference to a class type, or a pointer to class type, and the class type must itself have an "operator->" function.

Programmer response: Change the return value of the "operator->" function.

EQA2056E  Unused "&1" definition.

Problem determination: (where &1 is the keyword struct or class)

Explanation: An unnamed class or struct definition was found that has no object associated with it. The definition can never be referenced. A class can be unnamed, but it cannot be passed as an argument or returned as a value. An unnamed class cannot have any constructors or destructors.

Programmer response: Create an object for the class or struct, or remove the definition.

EQA2057E  Internal debugger error at line &1 in module "&2"; &3.

Explanation: The debugger detected an error within itself from which it cannot recover. The error was found within the debugger itself.

Programmer response: Note the line and module references in this message. Contact your IBM Representative Debug Tool support.

EQA2058E  Reference to member "&1" of undefined class "&2".

Problem determination: (where &1 is a member name &2 is a class name)

Explanation: The member has been explicitly given the specified class as a scope qualifier but the class (and hence the member) has not been defined.

Programmer response: Check the spelling of the scope qualifier. Change the scope qualifier to the class containing that member, or remove it.

EQA2059E  Pointer conversion may be wrong if the classes are related in a multiple inheritance hierarchy.

Explanation: The relationship between the classes in a pointer conversion is not known. If the target class is later defined as a base class of the source class in a multiple inheritance, this conversion will be wrong if the value of the pointer should have been modified by the conversion.

Programmer response: Change the ambiguous reference in the conversion.
EQA2060E The reference variable "&1" is uninitialized.

Problem determination: (&1 is a variable name)

Explanation: Reference variables must be initialized.

Programmer response: Initialize the reference variable or remove it.

EQA2061E "&1" must already be declared.

Problem determination: (&1 is a class or enum name)

Explanation: The specified class or enum name must have been declared before this use of the name.

Programmer response: Declare the class or enum name before you use it. Check the correct spelling of the name.

EQA2062E Unrecognized source character "&1", code point &2.

Problem determination: (&1 is a character &2 is an integer)

Explanation: The specified character is not a valid character in a C and C++ expression. The code point displayed represents its hexadecimal value.

Programmer response: Remove the character.

EQA2063E A local class cannot have a non-inline member function "&1".

Problem determination: (&1 is a function name)

Explanation: A class declared within a function must have all of its member functions defined inline, because the class will be out of scope before non-inline functions can be defined.

Programmer response: Define the functions inline, or move the class definition out of the scope of the function.

EQA2064E The size of "&1" is unknown in "&2" expression.

Problem determination: (&1 is a C and C++ type)

Explanation: The operation cannot be performed because the size of the specified type is not known.

Programmer response: Ensure the size of the type is known before this expression.

EQA2065E Assignment in logical expression.

Explanation: The logical expression contains an assignment (=). An equality comparison (==) might have been intended.

Programmer response: Change the operator or the expression.

EQA2066E Conversion from "&1" to "&2" may cause truncation.

Problem determination: (&1 is a C and C++ type &2 is a C and C++ type)

Explanation: The specified conversion from a wider to a narrower type might cause the loss of significant data.

Programmer response: Remove the conversion from a wider to a narrower type.

EQA2067E "goto &1" bypasses initialization of "&2".

Problem determination: (&1 is the C and C++ label used with the goto keyword &2 is the variable being initialized)

Explanation: Jumping past a declaration with an explicit or implicit initializer is not valid unless the declaration is in an inner block or unless the jump is from a point where the variable has already been initialized.

Programmer response: Enclose the initialization in a block statement.

EQA2068E References to "&1" may be ambiguous. The name is declared in base classes "&2" and "&3".

Problem determination: (&3 is a C++ class name)

Explanation: The debugger detected the base classes of a derived class have members with the same names. This will cause ambiguity when the member name is used. This is only an informational message because the declaration of a member with an ambiguous name in a derived class is not an error. The ambiguity is only flagged as an error if you use the ambiguous member name.

Programmer response: Change one of the names, or always fully qualify the name.

EQA2069E Ambiguous reference to "&1", declared in base classes "&2" and "&3".

Problem determination: (&3 is a C++ class name)

Explanation: The derived class made a reference to a member that is declared in more than one of its base classes and the debugger cannot determine which base class member to choose.

Programmer response: Change one of the names, or always fully qualify the name.

EQA2070E Conversion from "&1" to "&2" is ambiguous.

Problem determination: (&1 is a C++ type &2 is a C++ type)

Explanation: There is more than one way to perform the specified conversion. This ambiguity can be caused by an overloaded function.

Programmer response: Change or remove the conversion.

EQA2071E "&1" is only valid for non-static member functions.

Problem determination: (&1 is the keyword const or volatile)

Explanation: const and volatile are only significant for nonstatic member functions, since they are applied to the "this" pointer.

Programmer response: Remove const and volatile from all static members.

EQA2072E Character literal is null.

Explanation: An empty character literal has been specified. A string literal might have been intended.

Programmer response: Remove the character literal, change it to a string literal, or give it a value.
EQA2073E  "&1" has more than one base class "&2".
Problem determination: (where &1 is a class name &2 is a class name)
Explanation: A derived class has inherited the same base class in more than one path and the debugger cannot determine which one to choose.
Programmer response: Remove one of the inheritances.

EQA2074E  "&1" is a &2 base class of "&3".
Problem determination: (where &1 is a class name &2 is the keyword private or protected &3 is a class name)
Explanation: An attempt is being made to convert a pointer to a derived class into a pointer to a private or protected base class.
Programmer response: Remove the pointer conversion.

EQA2075E  &1 "&2" is not allowed in a union.
Problem determination: (where &1 is a C++ construct &2 is a C++ name)
Explanation: Unions must not be declared with base classes, virtual functions, static data members, members with constructors, members with destructors, or members with class copying assignment operators.
Programmer response: Remove any such members from the union declaration.

EQA2076E  union "&1" cannot be used as a base class.
Problem determination: (where &1 is a union name)
Explanation: Unions cannot be used as base classes for other class declarations.
Programmer response: Remove the union as a base class for other class declarations.

EQA2077E  Local variable "&1" is inaccessible from "&2".
Problem determination: (where &1 is a variable name &2 is a class name)
Explanation: An automatic variable within a function is not accessible from local classes declared within the function.
Programmer response: Remove the reference to the local variable, or move the variable to a different scope.

EQA2078E  Value of enumerator "&1" is too large.
Problem determination: (where &1 is an enumerator name)
Explanation: The value of an enumerator must be a constant expression that is promotable to a signed integer value.
Programmer response: Reduce the value of the enumerator.

EQA2079E  A constant is being used as a conditional expression.
Explanation: The condition to an if, for, or switch is constant and therefore, that condition will always hold.
Programmer response: Remove the constant or ignore this message.

EQA2080E  The argument to a not (!) operator is constant.
Explanation: The debugger has detected a constant after the ! operator that might be a coding error.
Programmer response: Remove the constant or ignore this message.

EQA2081E  There is more than one character in a character constant.
Explanation: Using more than one character in a character constant (for example, 'ab') might not be portable across machines.
Programmer response: Remove the extra character(s) or change the character constant to a string constant.

EQA2082E  Possible pointer alignment problem with the "&1" operator.
Problem determination: (where &1 is a C and C++ operator)
Explanation: A pointer that points to a type with less strict alignment requirements is being assigned, cast, returned or passed as a parameter to a pointer that is a more strictly aligned type. This is a potential portability problem.
Programmer response: Remove the pointer reference or change the alignment.

EQA2083E  A constant expression is being cast to a pointer.
Explanation: Casting a constant value to a pointer is not portable to other platforms.
Programmer response: Remove the constant expression from the cast expression.

EQA2084E  Precision will be lost in assignment to (possibly sign-extended) bit-field "&1".
Explanation: A constant is being assigned to a signed bit field that cannot represent the constant. Precision might be lost and the stored value will be incorrect.
Programmer response: Increase the size of the bit field.

EQA2085E  Precision will be lost in assignment to bit-field "&1".
Explanation: A constant is being assigned to a bit field, and because the bit field has a smaller size, the precision will be lost.
Programmer response: Change the assignment expression.

EQA2086E  Enumeration type clash with the "&1" operator.
Problem determination: (where &1 is a C++ operator)
Explanation: Operands from two different enumerations are used in an operation.
Programmer response: Ensure both operands are from the same enumeration.
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA2087E</td>
<td>Comparison of an unsigned value with a negative constant.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>An unsigned value is being compared to a negative number. The unsigned value will always compare greater than the negative number. This might be a programming error.</td>
</tr>
<tr>
<td>Programmer response:</td>
<td>Remove the comparison or change the type.</td>
</tr>
<tr>
<td>EQA2088E</td>
<td>Unsigned comparison is always true or always false.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The comparison is either &quot;unsigned &gt;= 0&quot;, which is always true, or &quot;unsigned &lt; 0&quot;, which is always false.</td>
</tr>
<tr>
<td>Programmer response:</td>
<td>Remove or change the comparison.</td>
</tr>
<tr>
<td>EQA2089E</td>
<td>Comparison is equivalent to &quot;unsigned value &amp;1 0&quot;.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The comparison is either &quot;unsigned &gt; 0&quot; or &quot;unsigned &lt;&lt;= 0&quot;, and could be written as &quot;unsigned != 0&quot; or &quot;unsigned == 0&quot;.</td>
</tr>
<tr>
<td>Programmer response:</td>
<td>Change the comparison.</td>
</tr>
<tr>
<td>EQA2090E</td>
<td>Argument &amp;1 for &quot;&amp;2&quot; must be of type &quot;&amp;3&quot;.</td>
</tr>
<tr>
<td>Problem determination:</td>
<td>(where &amp;1 is an argument number &amp;2 is a function name &amp;3 is a C++ type)</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The indicated function requires an argument of a particular type. However, the argument specified is of a different type than the type required.</td>
</tr>
<tr>
<td>Programmer response:</td>
<td>Ensure that the argument is of the correct type.</td>
</tr>
<tr>
<td>EQA2091E</td>
<td>Definition of &quot;&amp;1&quot; is not allowed.</td>
</tr>
<tr>
<td>Problem determination:</td>
<td>(where &amp;1 is the keyword class, struct, union or enum.)</td>
</tr>
<tr>
<td>Explanation:</td>
<td>You cannot define a type in a type cast or a conversion function declaration.</td>
</tr>
<tr>
<td>Programmer response:</td>
<td>Move the definition to a new location, or remove it.</td>
</tr>
<tr>
<td>EQA2092E</td>
<td>Reference to &quot;&amp;1&quot; is not allowed.</td>
</tr>
<tr>
<td>Problem determination:</td>
<td>(where &amp;1 is a C++ name)</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The name has a special meaning in a C++ program and cannot be referenced in this way.</td>
</tr>
<tr>
<td>Programmer response:</td>
<td>Remove the reference.</td>
</tr>
<tr>
<td>EQA2093E</td>
<td>Escape sequence &amp;1 is out of the range 0-&amp;2. Value is truncated.</td>
</tr>
<tr>
<td>Problem determination:</td>
<td>(where &amp;2 is the maximum allowed value of the escape sequence)</td>
</tr>
<tr>
<td>Programmer response:</td>
<td>Make the escape sequence small enough to fit the specified range.</td>
</tr>
<tr>
<td>EQA2094E</td>
<td>A wide character constant is larger than the size of a &quot;wchar_t&quot;. Only the last character is used.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>A wide character constant can only contain one character.</td>
</tr>
<tr>
<td>Programmer response:</td>
<td>Make the wide character constant smaller.</td>
</tr>
<tr>
<td>EQA2095E</td>
<td>A character constant is larger than the size of an &quot;int&quot;. Only the rightmost &amp;1 characters are used.</td>
</tr>
<tr>
<td>Problem determination:</td>
<td>(where &amp;1 is an integer number)</td>
</tr>
<tr>
<td>Programmer response:</td>
<td>Make the character constant smaller.</td>
</tr>
<tr>
<td>EQA2096E</td>
<td>Linkage specification must be at file scope.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>A linkage specification can only be defined at file scope, that is, outside all functions and classes.</td>
</tr>
<tr>
<td>Programmer response:</td>
<td>Move the linkage specification or remove it.</td>
</tr>
<tr>
<td>EQA2097E</td>
<td>Default initializers cannot be followed by uninitialized arguments.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>If a default initializer is specified in an argument list, all following arguments must also have default initializers.</td>
</tr>
<tr>
<td>Programmer response:</td>
<td>Remove the default initializers, or provide them for the following arguments, or move the arguments to the end of the list.</td>
</tr>
<tr>
<td>EQA2098E</td>
<td>You cannot take the address of &quot;&amp;1&quot;.</td>
</tr>
<tr>
<td>Problem determination:</td>
<td>(where &amp;1 is a C++ name)</td>
</tr>
<tr>
<td>Explanation:</td>
<td>You cannot take the address of a constructor, a destructor or a reference member.</td>
</tr>
<tr>
<td>Programmer response:</td>
<td>Remove the address operator (&amp;) from the expression or remove the expression.</td>
</tr>
<tr>
<td>EQA2099E</td>
<td>Duplicate qualifier &quot;&amp;1&quot; ignored.</td>
</tr>
<tr>
<td>Problem determination:</td>
<td>(where &amp;1 is a keyword)</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The keyword has been specified more than once. Extra occurrences are ignored.</td>
</tr>
<tr>
<td>Programmer response:</td>
<td>Remove one of the duplicate qualifiers.</td>
</tr>
<tr>
<td>EQA2100E</td>
<td>&quot;&amp;1&quot; operator cannot be overloaded.</td>
</tr>
<tr>
<td>Problem determination:</td>
<td>(where &amp;1 is an operator name)</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The specified operator cannot be overloaded using an operator function. The following operators cannot be overloaded: . * :: ?:</td>
</tr>
<tr>
<td>Programmer response:</td>
<td>Remove the overloading declaration or definition.</td>
</tr>
<tr>
<td>EQA2101E</td>
<td>At least one argument of &quot;&amp;1&quot; must be of class or enum type.</td>
</tr>
<tr>
<td>Problem determination:</td>
<td>(where &amp;1 is an operator function name)</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The nonmember operator function must have at least one argument which is of class or enum type.</td>
</tr>
<tr>
<td>Programmer response:</td>
<td>Add an argument of class or enum type.</td>
</tr>
</tbody>
</table>
EQA2102E • EQA2115E

EQA2102E  The divisor for the modulus or division operator cannot be zero.
Programmer response: Change the expression used in the divisor.

EQA2103E  The address of the bit-field "&1" cannot be taken.
Problem determination: (where &1 is a member name)
Explanation: An expression attempts to take the address of a bit-field, or to use the bit-field to initialize a reference variable or argument.
Programmer response: Remove the expression causing the error.

EQA2104E  "&1" must not have default initializers.
Problem determination: (where &1 is an operator function name or "template function")
Explanation: Default initializers are not allowed within the declaration of an operator function or a template function.
Programmer response: Remove the default initializers.

EQA2105E  The &1 "&2" cannot be initialized because it does not have a default constructor.
Problem determination: (where &1 is 'base class' or 'class member' &2 is a C++ name)
Explanation: The specified base class or member cannot be constructed since it is not initialized in the constructor initializer list and its class has no default constructor.
Programmer response: Specify a default constructor for the class or initialize it in the constructor initializer list.

EQA2106E  Template class "&1" has the wrong number of arguments.
Problem determination: (where &1 is a template class name)
Explanation: A template class instantiation has a different number of template arguments than the template declaration.
Programmer response: Ensure that the template class has the same number of declarations as the template declaration.

EQA2107E  Non-&1 member function "&2" cannot be called for a &1 object.
Problem determination: (where &2 is a function name with arguments)
Explanation: The member function is being called for a const or volatile object but the member function has not been declared with the const or volatile qualifier.
Programmer response: Supply a version of the member function with the correct set of "const" and "volatile" qualifiers.

EQA2108E  Null statement.
Explanation: Possible extraneous semicolon has been specified.
Programmer response: Check for extra semicolons in statement.

EQA2109E  Bit-field "&1" cannot be used in a conditional expression that is to be modified.
Explanation: The bit-field is part of a conditional expression that is to be modified. Only objects that can have their address taken are allowed as part of such an expression, and you cannot take the address of a bit field.
Programmer response: Remove the bit-field from the conditional expression.

EQA2110E  The "&1" qualifier cannot be applied to "&2".
Problem determination: (where &2 is a name or a type)
Explanation: The qualifier is being applied to a name or a type for which it is not valid.
Programmer response: Remove the qualifier.

EQA2111E  Local type "&1" cannot be used as a &2 argument.
Problem determination: (where &2 is either the keyword template or the keyword function)
Explanation: The type cannot be used as a function argument or in the instantiation of a template because the scope of the type is limited to the current function.
Programmer response: Remove the local type.

EQA2112E  Default initializers for nontype template arguments are only allowed for class templates.
Explanation: Default initializers have been given for nontype template arguments, but the template is not declaring a class.
Programmer response: Remove the default initializers.

EQA2113E  A function argument must not have type "void".
Explanation: A function argument can be an expression of any object type. However, "void" is not the type of any object and cannot be used as an argument type.
Programmer response: Change the type of the function argument.

EQA2114E  Insufficient memory in line &1 of file "&2".
Problem determination: (where &1 is a line number &2 is a file name)
Explanation: The debugger ran out of memory.
Programmer response: Increase your storage and rerun.

EQA2115E  Unable to initialize source conversion from codepage &1 to codepage &2.
Problem determination: (where &1 is a codepage name i.e. IBM-1047 &2 is a codepage name i.e. IBM-1047)
Explanation: An error occurred when attempting to convert source between the codepages specified.
Programmer response: Ensure the codepages are correct and that conversion between these codepages is supported.

Chapter 8. Debug Tool messages  305
EQA2116E  An object of abstract class "&1" cannot be created.

Problem determination: (where &1 is a class name)

Explanation: You cannot create instances of abstract classes. An abstract class is a class that has or inherits at least one pure virtual function.

Programmer response: Derive another object from the abstract class.

EQA2117E  Invalid use of an abstract class.

Explanation: An abstract class must not be used as an argument type, as a function return type, or as the type of an explicit conversion.

Programmer response: Derive another class from the abstract, instantiate it so it becomes a concrete object, and then use it instead.

EQA2118E  "&1" has been used more than once in the same base class list.

Problem determination: (where &1 is base class name)

Explanation: A base class can only be specified once in the base class list for a derived class.

Programmer response: Remove one of the specifications.

EQA2119E  Template argument &1 of type "&2" does not match declared type "&3".

Problem determination: (where &1 is a class name, &2 is a parameter or template, &3 is a declared class or type)

Explanation: A non-type template argument must have a type that exactly matches the type of the corresponding argument in the template declaration.

Programmer response: Ensure that the types match.

EQA2120E  Template argument &1 of type "&2" is not an allowable constant value or address.

Problem determination: (where &1 is an integer type and &2 is a C++ type)

Explanation: A non-type template argument must be a constant value or the address of an object, function, or static data member that has external linkage. String literals cannot be used as template arguments because they have no name, and therefore no linkage.

Programmer response: Change the template argument.

EQA2121E  Template argument list is empty.

Explanation: At least one template argument must be specified in a template declaration.

Programmer response: Specify a template argument in the declaration.

EQA2122E  Formal template argument &1 is of type "&2" which is not an allowable integral, enumeration, or pointer type.

Problem determination: (where &1 is an integer type and &2 is a C++ type)

Explanation: A non-type template argument must be of integral, or enumeration, or pointer type, so that it can be matched with a constant integral value.

Programmer response: Change the template argument.

EQA2123E  "&1" is defined in a template declaration but it is not a static member.

Problem determination: (where &1 is a C++ name)

Explanation: A member of a template class defined in a template declaration must be a static member.

Programmer response: Make the member static or remove it from the template declaration.

EQA2124E  Template argument "&1" is not used in the declaration of the name or the argument list of "&2".

Problem determination: (where &1 is a template argument name &2 is a C++ name)

Explanation: All template arguments for a nonclass template must be used in the declaration of the name or the function argument list.

Programmer response: Ensure all template arguments are used in the declaration of the name or the function argument list.

EQA2125E  Template declaration does not declare a class, a function, or a template class member.

Explanation: Following the template argument, a template declaration must declare a class, a function, or a static data member of a template class.

Programmer response: Change the template declaration to declare a class, a function, or a template class member.

EQA2126E  Return type "&1" for function "&2" differs from previous return type of "&3".

Problem determination: (where &1 is a C and &2 is a function name &3 is a C and &3 is a C++ type)

Explanation: The declaration of the function differs from a previous declaration in only the return type.

Programmer response: Change the return type so that it matches the previous return type.

EQA2127E  "&1" is a member of "&2" and cannot be used without qualification.

Problem determination: (where &2 is a possibly qualified class name)

Explanation: The specified name is a class member, but no class qualification has been used to reference it.

Programmer response: Add a class qualification to the class member.

EQA2128E  "&1" cannot be initialized multiple times.

Problem determination: (where &1 is a member or base class name)

Explanation: An initializer was already specified in the constructor definition.

Programmer response: Remove the additional initializer.
### EQA2129E • EQA2136E

**EQA2129E**  
No suitable copy assignment operator exists to perform the assignment.  

**Explanation:** A copy assignment operator exists but it does not accept the type of the given parameter.  

**Programmer response:** Change the copy assignment operator.

---

**EQA2136E**  
"&1" is not allowed as a conversion function type.  

**Problem determination:** (where &1 is a C and C++ type)  

**Explanation:** A conversion function cannot be declared with a function or an array as its conversion type, since the type cannot be returned from the function.  

**Programmer response:** Declare the function as converting to a pointer to the function or the array element type.

---

### EQA2130E • EQA2137E

**EQA2130E**  
Explicit call to constructor "&1" is not allowed.  

**Problem determination:** (where &1 is a constructor name)  

**Explanation:** You cannot call a constructor explicitly. It is called implicitly when an object of the class is created.  

**Programmer response:** Remove the call to the constructor.

---

**EQA2137E**  
Syntax error - "&1" is followed by "&3" but is not the name of a &2.  

**Problem determination:** (where &1 is a C++ name &2 is the keyword class or template &3 is the token ':=' or '<')  

**Explanation:** The name is not a class or template name but the context implies that it should be.  

**Programmer response:** Change the name to a class or template name.

---

### EQA2131E • EQA2138E

**EQA2131E**  
No default constructor exists for "&1".  

**Problem determination:** (where &1 is a class name)  

**Explanation:** An array of class objects must be initialized by calling the default constructor, but one has not been declared.  

**Programmer response:** Declare a default constructor for the array.

---

**EQA2138E**  
The previous &1 messages apply to the definition of template "&2".  

**Problem determination:** (where &1 is an integer number &2 is a template name)  

**Explanation:** The instantiation of the specified template caused the messages, even though the line numbers in the messages refer to the original template declaration.  

**Programmer response:** Change the template name in the original declaration to the name used in the definition.

---

### EQA2132E • EQA2139E

**EQA2132E**  
More than one default constructor exists for "&1".  

**Problem determination:** (where &1 is a class name)  

**Explanation:** An array of class objects must be initialized by calling the default constructor, but the call is ambiguous.  

**Programmer response:** Declare a default constructor for the array.

---

**EQA2139E**  
The previous message applies to the definition of template "&1".  

**Problem determination:** (where &1 is a template name)  

**Explanation:** The instantiation of the specified template caused the message, even though the line number in the message refers to the original template declaration.  

**Programmer response:** This message supplies additional information for previously emitted messages. Refer to the descriptions of those messages for recovery information.

---

### EQA2133E • EQA2140E

**EQA2133E**  
The debugger cannot generate a default copy constructor for "&1".  

**Explanation:** The default copy constructor cannot be generated for this class because there exists a member or base class that has a private copy constructor, or there are ambiguous base classes, or this class has no name.  

**Programmer response:** Ensure that only one default constructor exists.

---

**EQA2140E**  
No suitable constructor exists for conversion from "&1" to "&2".  

**Problem determination:** (where &1 is a class name &2 is a C++ type)  

**Explanation:** A constructor is required for the class but no user-defined constructor exists and the debugger could not generate one.  

**Programmer response:** Create a suitable constructor for conversion.

---

### EQA2134E • EQA2141E

**EQA2134E**  
The debugger cannot generate a default copy assignment operator for "&1".  

**Explanation:** The default copy assignment operator cannot be generated for this class because it has a const member or a reference member or a member (or base class) with a private copy assignment operator.  

**Programmer response:** Ensure there are no const members, reference members or members with a private copy assignment operator.

---

**EQA2141E**  
Class "&1" does not have a copy assignment operator.  

**Problem determination:** (where &1 is a class name)  

**Explanation:** A copy assignment operator is required for the class but no user-defined copy assignment operator exists and the debugger could not generate one.
EQA2142E • EQA2156E

Programmer response: Create a copy assignment operator.

EQA2142E "&1" cannot be used as a template name since it is already known in this scope.
Explanation: (where &1 is a C++ name) A template name must not match the name of an existing template, class, function, object, value or type.
Programmer response: Change one of the template names.

EQA2143E "&1" is expected for template argument &2.
Problem determination: (where &1 is either 'expression' or 'type name' &2 is a non-type argument)
Explanation: Either the argument is a type and the template has a non-type argument, or the argument is an expression and the template has a type argument.
Programmer response: Ensure the argument matches the template.

EQA2144E "&1" cannot be defined before the template definition of which it is an instance.
Problem determination: (where &1 is a class template name)
Explanation: An explicit definition of a template class cannot be given before the corresponding template definition.
Programmer response: Move the template definition so that it occurs before any template class definitions.

EQA2145E An ellipsis (...) cannot be used in the argument list of a template function.
Explanation: Since an exact match is needed for template functions, an ellipsis cannot be used in the function argument list.
Programmer response: Remove the ellipsis from the argument list.

EQA2146E The suffix for the floating point constant is not valid.
Explanation: You have provided an incorrect suffix for the floating point constant. Valid suffixes for floating point constants are L and F.
Programmer response: Change the suffix for the floating point constant.

EQA2147E Statement has no effect.
Explanation: The expression has no side effects and produces a result that is not used.
Programmer response: Remove the statement or use its result.

EQA2148E The suffix for the integer constant is not valid.
Explanation: The integer constant is a suffix letter that is not recognized as a valid suffix.
Programmer response: Change the suffix to either "u" or "l".

EQA2149E The expression contains a division by zero.
Programmer response: Remove the division by zero from the expression.

EQA2150E The expression contains a modulus by zero.
Programmer response: Remove the modulus by zero from the expression.

EQA2151E Static member "&1" can only be defined at file scope.
Programmer response: Move the static member so that it is defined at file scope.

EQA2152E "&1" needs a constructor because &2 "&3" needs a constructor initializer.
Problem determination: (where &1 is a class name &2 is 'class member' or 'base class' &3 is the member or base class name.)
Explanation: You have not provided a constructor for the class, because the member or base class does not have a default constructor.
Programmer response: Add a constructor.

EQA2153E Conversion from "&1" to a reference to a non-const type &2 requires a temporary.
Problem determination: (where &1 is a C++ type &2 is a C++ type)
Explanation: A temporary can only be used for conversion to a reference type when the reference is to a const type.
Programmer response: Change to a const type.

EQA2154E "&2" is too small to hold a value of type "&1".
Problem determination: (where &1 is a C++ type &2 is a C++ type)
Explanation: A conversion from a pointer type to an integral type is only valid if the integral type is large enough to hold the pointer value.
Programmer response: Remove the conversion from a pointer type to an integral type or use a larger integral type.

EQA2155E Object of type "&1" cannot be constructed from "&2" expression.
Problem determination: (where &1 is a C++ type &2 is a C++ type)
Explanation: There is no constructor taking a single argument that can be called using the given expression.
Programmer response: Change the expression.

EQA2156E The debugger cannot generate a copy constructor for conversion to "&1".
Problem determination: (where &1 is a C++ type)
Explanation: A copy constructor is required for the conversion. No suitable user-defined copy constructor exists and the debugger could not generate one.
EQA2157E • EQA2168E

Programmer response: Create a copy constructor for the conversion.

EQA2157E No suitable constructor or conversion function exists for conversion from "&1" to "&2".

Problem determination: (where &1 is a C++ type &2 is a C++ type)

Explanation: A constructor or conversion function is required for the conversion but no such constructor or function exists.

Programmer response: Create a constructor or conversion function for the conversion.

EQA2158E Syntax error - "&1" has been inserted before "&2".

Problem determination: (where &1 is a token &2 is a token)

Explanation: A syntax error was found while parsing the expression. The message identifies what the debugger expected and what it actually found.

Programmer response: Correct the syntax.

EQA2159E Call to "&1" matches some functions best in some arguments, but no function is a best match for all arguments.

Problem determination: (where &1 is a function name)

Explanation: No function matches each call argument as well as or better than all other functions.

Programmer response: Change the function call so that it matches only one function.

EQA2160E Call matches "&1".

Problem determination: (where &1 is a function name and type)

Explanation: The debugger detected an overloaded function or operator that is similar to another and is providing additional information.

Programmer response: Ensure this is the desired match.

EQA2161E Cannot adjust access of "&1::&2" because a member in "&3" hides it.

Problem determination: (where &1 is a class name &2 is a member name &3 is the name of the derived class)

Explanation: You cannot modify the access of the specified member because a member of the same name in the specified class hides it.

Programmer response: Remove the access adjustment expression or unhide the member.

EQA2162E "&1" cannot be redeclared.

Problem determination: (where &1 is a C++ name)

Explanation: The specified name cannot be redeclared because it has already been used.

Programmer response: Change or remove one of the declarations.

EQA2163E Syntax error - "&1" is not allowed; "&2" has already been specified.

Problem determination: (where &1 is a keyword &2 is a keyword)

Explanation: You cannot use both of the specified attributes in the same declaration.

Programmer response: Remove the attributes.

EQA2164E Call to "&1" matches more than one template function.

Problem determination: (where &1 is a function name and type)

Explanation: More than one template for the function matches equally well with the argument list specified on the call.

Programmer response: Change the call so that it matches only one template function.

EQA2165E "&1" is declared inline, but is undefined.

Problem determination: (where &1 is a function name and type)

Explanation: An inline function must be defined in every compilation unit in which it is used.

Programmer response: Define the inline function in this compilation unit.

EQA2166E Non-@1 member function called for a &1 object via pointer of type "&2".

Problem determination: (where &2 is a pointer or member-pointer type)

Explanation: The member function is being called indirectly for a const or volatile object but it has not been declared with the corresponding const or volatile attribute.

Programmer response: Ensure that the function call and the function declaration match.

EQA2167E "&1" cannot be a base of "&2" because "&3" contains the type name "&2".

Problem determination: (where &1 is a class name &2 is both the derived class name and a type name &3 is the class containing &2)

Explanation: A class cannot inherit a type name that is the same as the class name.

Programmer response: Change the name of either the derived class or the inherited class.

EQA2168E "&1" cannot be a base of "&2" because "&3" contains the enumerator "&2".

Problem determination: (where &1 is a class name &2 is both the derived class name and the enumerator name &3 is the class containing &2)

Explanation: A class cannot inherit an enumerator with the same name as the class name.

Programmer response: Change the name of either the derived class or the inherited enumerator.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
<th>Problem Determination</th>
<th>Explanation</th>
<th>Programmer Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA2169E</td>
<td>Symbol length of &quot;&amp;1&quot; exceeds limit of &amp;2 bytes.</td>
<td>(where &amp;1 is an integer type)</td>
<td>The debugger limit for the length of a symbol has been exceeded.</td>
<td>Shorten the symbol length.</td>
</tr>
<tr>
<td>EQA2170E</td>
<td>The result of this pointer to member operator can be used only as the operand of the function call operator ().</td>
<td>(where &amp;1 is an integer type)</td>
<td>If the result of the * or -&gt; is a function, that result can be used only as the operand for the function call operator ().</td>
<td>Make the result the operand of the function call operator ().</td>
</tr>
<tr>
<td>EQA2171E</td>
<td>When &quot;&amp;1&quot; is used as an operand to the arrow or dot operator, the result must be used with the function call operator ().</td>
<td>(where &amp;1 is a member name)</td>
<td>If the result of the dot or arrow operator is a function, that result can be used only as the operand for the function call operator ().</td>
<td>Make the result the operand of the function call operator ().</td>
</tr>
<tr>
<td>EQA2172E</td>
<td>A class with a reference or const member needs a constructor.</td>
<td>(where &amp;1 is a member name)</td>
<td>Const and reference members must be initialized in a constructor initializer list.</td>
<td>Add a constructor to the class.</td>
</tr>
<tr>
<td>EQA2173E</td>
<td>Base class initializers cannot contain virtual function calls.</td>
<td>(where &amp;1 is a class without a default constructor)</td>
<td>If there are fewer initializers in the list than elements in the array, the default constructor is used. If there is no default constructor the initializer list must be complete.</td>
<td>Complete the initializer list or add a default constructor to the class.</td>
</tr>
<tr>
<td>EQA2174E</td>
<td>The previous declaration of &quot;&amp;1&quot; did not have a linkage specification.</td>
<td>(where &amp;1 is a C++ type)</td>
<td>If you want to declare a linkage specification for a function, it must appear in the first declaration of the function.</td>
<td>Add a linkage specification to the first declaration of the function.</td>
</tr>
<tr>
<td>EQA2175E</td>
<td>The destructor for &quot;&amp;1&quot; does not exist. The call is ignored.</td>
<td>(where &amp;1 is a C++ type)</td>
<td>The destructor call is for a type that does not have a destructor. The call is ignored.</td>
<td>Add a destructor to the type.</td>
</tr>
<tr>
<td>EQA2176E</td>
<td>&quot;&amp;1&quot; has been added to the scope of &quot;&amp;2&quot;.</td>
<td>(where &amp;1 is the name on a friend declaration &amp;2 is a class name)</td>
<td>Because the friend class has not been declared yet, its name has been added to the scope containing the friend declaration.</td>
<td>Make the function call that calls the friend class appear before it is declared as a friend.</td>
</tr>
<tr>
<td>EQA2177E</td>
<td>The body of friend member function &quot;&amp;1&quot; cannot be defined in the member list of &quot;&amp;2&quot;.</td>
<td>(where &amp;1 is the friend member function &amp;2 is a class name)</td>
<td>A friend function that is a member of another class cannot be defined inline in the member list.</td>
<td>Define the body of the friend function at file scope.</td>
</tr>
<tr>
<td>EQA2178E</td>
<td>The initializer list must be complete because &quot;&amp;1&quot; does not have a default constructor.</td>
<td>(where &amp;1 is a class without a default constructor)</td>
<td>An array of objects of a class with constructors uses the constructors in initialization. If there are fewer initializers in the list than elements in the array, the default constructor is used. If there is no default constructor the initializer list must be complete.</td>
<td></td>
</tr>
<tr>
<td>EQA2179E</td>
<td>A pure virtual destructor needs an out-of-line definition in order for its class to be a base of another class.</td>
<td>(where &amp;1 is a class without a default constructor)</td>
<td>If there are fewer initializers in the list than elements in the array, the default constructor is used. If there is no default constructor the initializer list must be complete.</td>
<td></td>
</tr>
<tr>
<td>EQA2180E</td>
<td>The braces in the initializer are incorrect.</td>
<td>(where &amp;1 is a class without a default constructor)</td>
<td>If there are fewer initializers in the list than elements in the array, the default constructor is used. If there is no default constructor the initializer list must be complete.</td>
<td></td>
</tr>
<tr>
<td>EQA2181E</td>
<td>Invalid octal integer constant.</td>
<td>(where &amp;1 is a class without a default constructor)</td>
<td>The octal integer constant contains an '8' or a '9'. Octal numbers include 0 through 7.</td>
<td>Ensure that the octal integer constant is valid.</td>
</tr>
<tr>
<td>EQA2182E</td>
<td>All the arguments must be specified for &quot;&amp;1&quot; because its default arguments have not been checked yet.</td>
<td>(where &amp;1 is a member function name and a type)</td>
<td>For member functions, names in default argument expressions are bound at the end of the class declaration. Calling a member function as part of a second member function’s default argument is an error if the first member function’s default arguments have not been checked and the call does not specify all of the arguments.</td>
<td></td>
</tr>
</tbody>
</table>

310  Debug Tool V7.1 Reference and Messages
Programmer response: Specify all the arguments for the function.

**EQA2183E** Ellipses (...) cannot be used for "&1".
Problem determination: (where &1 is an operator name)
Explanation: An operator function has been specified with an ellipsis (...), but since the number of operands of an operator are fixed, an ellipsis is not allowed.
Programmer response: Remove the ellipsis, and specify the correct number of operands.

**EQA2184E** Syntax error - expected ",&1" or ",&2" and found ",&3".
Problem determination: (where &1 is a token &2 is a token &3 is a token)
Explanation: A syntax error was found while parsing the program. The message identifies what the debugger expected and what it actually found.
Programmer response: Correct the syntax error.

**EQA2185E** A character constant must end before the end of the line.
Explanation: The debugger detected a character constant that was not terminated before an end-of-line character was found.
Programmer response: End the character constant or use "\" to continue it on the next line. The "\" must be the last character on the line.

**EQA2186E** A pure virtual function initializer must be 0.
Explanation: To declare a pure virtual function use an initializer of 0.
Programmer response: Set the virtual function initializer to 0.

**EQA2187E** ",&1" is given ",&2" access.
Problem determination: (where &1 is a member name &2 is the keyword public, protected or private)
Explanation: Access of the class has changed.
Programmer response: Ensure this change is as intended.

**EQA2188E** ",&1" has been qualified with the "this" pointer.
Problem determination: (where &1 is a member name)
Programmer response: Ensure this qualification is intended.

**EQA2189E** Invalid escape sequence; the backslash is ignored.
Explanation: You have provided invalid character(s) after the backslash that does not represent an escape sequence. Therefore, the backslash is ignored and the rest of the escape sequence is read as is.
Programmer response: Ensure the escape sequence is valid.

**EQA2190E** The result of an address expression is being deleted.
Programmer response: Ensure this action is intended.

**EQA2191E** Conversion from ",&1" to ",&2" matches more than one conversion function.
Explanation: More than one conversion function could be used to perform the specified conversion.
Programmer response: Create a new conversion function for this conversion or change one of the types.

**EQA2192E** Conversion matches ",&1".
Problem determination: (where &1 is a function name and type)
Programmer response: Ensure this is the intended match.

**EQA2193E** ",&1" cannot be initialized with an initializer list.
Problem determination: (where &1 is a class name)
Explanation: Only an object of a class with no constructors, no private or protected members, no virtual functions and no base classes can be initialized with an initializer list.
Programmer response: Remove the class from the initializer list.

**EQA2194E** A pointer to a virtual base ",&1" cannot be converted to a pointer to a derived class ",&2".
Problem determination: (where &1 is a C++ type &2 is a C++ type)
Explanation: A pointer to a class B can be explicitly converted to a pointer to a class D that has B as a direct or indirect base class, only if an unambiguous conversion from D to B exists, and B is not a virtual base class.
Programmer response: Remove the conversion of the pointer.

**EQA2195E** The arguments passed using the ellipsis may not be accessible.
Explanation: Arguments passed using an ellipsis are only accessible if there is an argument preceding the ellipsis and the preceding argument is not passed by reference.
Programmer response: Ensure that there is an argument preceding the ellipsis and that the preceding argument is not passed by reference.

**EQA2196E** Assignment to a constant expression is not allowed.
Explanation: The left hand side of the assignment operator is an expression referring to a "const" location. For example, in "a.b", either "b" is a "const" member or "a" is a "const" variable.
Programmer response: Remove the assignment.

**EQA2197E** Assignment to const variable ",&1" is not allowed.
Problem determination: (where &1 is the variable name)
Explanation: The left hand side of the assignment operator is a variable with the "const" attribute. "const" variables can be initialized once at the point where they are declared, but cannot be
subsequently assigned new values.

Programmer response: Remove the assignment to the const variable.

---

EQA2198E The return type for the "operator->" cannot be the containing class.

Explanation: The return type for the "operator->" must be a pointer to a class type, a class type, or a reference to a class type. If it is a class or reference, the class must be previously defined and must contain an "operator->" function.

Programmer response: Change the return type for the "operator->".

---

EQA2199E The previous message applies to function argument &1.

Problem determination: (where &1 is an integer corresponding to the function argument number)

Explanation: The previous message applies to the specified argument number. This message does not indicate another error or warning, it indicates which argument of the function call is the subject of the previous message.

Programmer response: Change the reference so that it is to a const type.

---

EQA2200E Conversion from ",&1" to a reference to a non-const type ",&2" requires a temporary.

Problem determination: (where &1 is a C++ type &2 is a C++ type)

Explanation: A temporary can only be used for conversion to a reference type when the reference is to a const type. This is a warning rather than an error message because the 'compat' language level is active.

Programmer response: Change the reference so that it is to a const type.

---

EQA2201E The pointer to member function must be bound to an object when it is used with the function call operator ()

Explanation: The pointer to member function must be associated with an object or a pointer to an object when it is used with the function call operator ().

Programmer response: Remove the pointer or associate it with an object.

---

EQA2202E The direct base ",&1" of class ",&2" is ignored because ",&1" is also an indirect base of ",&2".

Problem determination: (where &1 is a base class name)

Explanation: A reference to a member of ",&1" will be ambiguous because it is inherited from two different paths.

Programmer response: Remove the indirect inheritance.

---

EQA2203E The "&1" operator cannot be applied to undefined class ",&2".

Problem determination: (where &1 is a class type)

Explanation: A class is undefined until the definition of its tag has been completed. A class tag is undefined when the list describing the name and type of its members has not been specified. The definition of the tag must be given before the operator is applied to the class.

Programmer response: Complete the definition of the class before applying an operator to it.

---

EQA2204E ",&1" hides the ",&2" ",&3".

Problem determination: (where &1 is the name of the derived class's member &2 is "pure virtual" or "virtual" &3 is the name of the hidden virtual function)

Explanation: A member in the derived class hides a virtual function member in a base class.

Programmer response: Ensure the hiding of the virtual function member is intended.

---

EQA2205E ",&1" is not the name of a function.

Problem determination: (where &1 is a C++ name)

Explanation: A function name is required in this context. The specified name has been declared but it is not the name of a function.

Programmer response: Check the spelling. If necessary, change to a function name.

---

EQA2206E The virtual functions ",&1" and ",&2" are ambiguous since they override the same function in virtual base class ",&3".

Problem determination: (where &1 is a function name and type &2 is a function name and type)

Explanation: The two functions are ambiguous and the virtual function call mechanism will not be able to choose the correct one at run time.

Programmer response: Remove one of the virtual functions.

---

EQA2207E The "this" address for ",&1" is ambiguous because there are multiple instances of ",&2".

Problem determination: (where &1 is a function name and type &2 is a class name)

Explanation: Two or more "this" addresses are possible for this virtual function. The virtual function call mechanism will not be able to determine the correct address at run time.

Programmer response: Remove the "this" expression or change the function name.

---

EQA2208E Conversion from ",&1" matches more than one conversion function.

Problem determination: (where &1 is a function name and type)

Explanation: More than one conversion function could be applied to perform the conversion from the specified type.

Programmer response: Create a new conversion function or remove the conversion.

---

EQA2209E ",&1" cannot be a base of ",&2" because ",&3" contains a member function called ",&2".

Problem determination: (where &1 is a class name &2 is both the derived class name and the member function &3 is the class containing &2)

Explanation: A class cannot inherit a function that has the same as the class.
<table>
<thead>
<tr>
<th>Problem response</th>
<th>Explanation</th>
<th>Problem determination</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmer response: Change the name of either the base class or the inherited function.</td>
<td>EQA2210E Forward declaration of the enumeration &quot;&amp;1&quot; is not allowed.</td>
<td>(where &amp;1 is the argument number &amp;2 is the function name and type)</td>
<td>EQA2221E A character string literal cannot be concatenated with a wide string literal.</td>
</tr>
<tr>
<td>Programmer response: Fully declare the enumeration.</td>
<td>Explanation: The declaration of an enumeration must contain its member list.</td>
<td>The previous message applies to argument &amp;1 of function &quot;&amp;2&quot;.</td>
<td>Explanation: A string that has a prefix L cannot be concatenated with a string that is not prefixed.</td>
</tr>
<tr>
<td>Programmer response: Create a constructor for the nested class object.</td>
<td>EQA2211E The previous message applies to argument &amp;1 of function &quot;&amp;2&quot;.</td>
<td>(where &amp;1 is the nested class name &amp;2 is the word const or reference)</td>
<td>EQA2217E All members of type &quot;&amp;1&quot; must be explicitly initialized with all default arguments specified.</td>
</tr>
<tr>
<td>Programmer response: Ensure the integer constant is in range.</td>
<td>EQA2212E The nested class object &quot;&amp;1&quot; needs a constructor so that its &amp;2 members can be initialized.</td>
<td>(where &amp;1 is the nested class name &amp;2 is the word const or reference)</td>
<td>Problem determination: (where &amp;1 is a class name &amp;2 is the member name)</td>
</tr>
<tr>
<td>Programmer response: Ensure the integer constant is in range.</td>
<td>EQA2213E The integer constant is out of range.</td>
<td>Explanation: You have provided an integer constant that is out of range. For the range of integer constants check limits.h.</td>
<td>Explanation: Default arguments for member functions are not checked until the end of the class definition. Default arguments for member functions of nested classes are not semantically checked until the containing class is defined. A call to a member function must specify all of the arguments before the default arguments have been checked.</td>
</tr>
<tr>
<td>Programmer response: Ensure the floating point constant is in range.</td>
<td>EQA2214E The floating point constant is out of range.</td>
<td>Explanation: You have provided a floating point constant that is out of range. For the range of floating point constants check float.h.</td>
<td>Problem determination: (where &amp;1 is a token &amp;2 is a token)</td>
</tr>
<tr>
<td>Programmer response: Change the type name or remove the expression.</td>
<td>EQA2215E The &amp;1 member &quot;&amp;2&quot; must be initialized in the constructor's initializer list.</td>
<td>(where &amp;1 is the word const or reference &amp;2 is the member name)</td>
<td>Explanation: The debugger detected a nontype symbol where a type is required. A type must be used to declare an object.</td>
</tr>
<tr>
<td>Programmer response: Change the type of the result.</td>
<td>EQA2216E Constructors and conversion functions are not considered when resolving an explicit cast to a reference type.</td>
<td>Using the constructor's member initializer list is the only way to initialize nonstatic const and reference members.</td>
<td>Problem determination: (where &amp;1 is the derived member's name &amp;2 is the name of the pure virtual function &amp;3 is the name of the class that contains the pure virtual)</td>
</tr>
<tr>
<td>Programmer response: Make the pure virtual function visible, or make the base it is derived from virtual.</td>
<td>Explanation: You cannot resolve an explicit cast to a reference type using constructors or conversion functions.</td>
<td>Initialize the member in the constructor's initializer list.</td>
<td>Explanation: The pure virtual function in a nonvirtual base cannot be overridden once it has been hidden.</td>
</tr>
</tbody>
</table>
### Debug Tool V7.1 Reference and Messages

- **EQA223E** The class qualifier "&1" for "&2" must be a template class that uses the template arguments.

  **Problem determination:** (where &1 is a (possibly qualified) class name. &2 is a C++ name.)

  **Explanation:** A nonclass template can only declare a global function or a member of a template class. If it declares a member of a template class, the template class arguments must include at least one of the nonclass template arguments.

  **Programmer response:** Change the template declaration so that it either declares a global function or a member of a template class that uses the nonclass template arguments.

- **EQA224E** The class "&1" cannot be passed by value because it does not have a copy constructor.

  **Problem determination:** (where &1 is a class name)

  **Explanation:** The debugger needs to generate a temporary to hold the return value of the function. To generate the temporary object, a copy constructor is needed to copy the contents of the object being returned into the temporary object.

  **Programmer response:** Create a copy constructor for the class or change the argument to pass by value.

- **EQA225E** "&1" cannot have an initializer list.

  **Problem determination:** (where &1 is a function name)

  **Explanation:** A member function that is not a constructor is defined with an initializer list.

  **Programmer response:** Remove the initializer list.

- **EQA226E** Return value of type "&1" is expected.

  **Problem determination:** (where &1 is a C and C++ type)

  **Explanation:** No return value is returned from the current function but the function is expecting a nonvoid return value.

  **Programmer response:** Ensure a value is returned, or change the return type of the function to void.

- **EQA227E** "&1" bypasses initialization of "&2".

  **Problem determination:** (where &1 is one of the keywords default, case &2 is the variable being initialized)

  **Explanation:** It is invalid to jump past a declaration with an explicit or implicit initializer unless the declaration is in an inner block that is also jumped past.

  **Programmer response:** Enclose the initialization in a block statement.

- **EQA228E** "&1" is being redeclared as a member function. It was originally declared as a data member.

  **Problem determination:** (where &1 is a variable name)

  **Explanation:** The template redeclares a data member of a class template as a member function.

  **Programmer response:** Change the original declaration of the variable to a member function, or change the redeclaration of the variable to a data member.

- **EQA2229E** "&1" is being redeclared as a nonfunction member or has syntax errors in its argument list.

  **Problem determination:** (where &1 is a (possibly qualified) class name. &2 is a C++ name.)

  **Explanation:** A nonclass template can only declare a global function or a member of a template class. If it declares a member of a template class, the template class arguments must include at least one of the nonclass template arguments.

  **Programmer response:** Change the template declaration so that it either declares a global function or a member of a template class that uses the nonclass template arguments.

- **EQA2230E** A string literal cannot be longer than &1 characters.

  **Problem determination:** (where &1 is a number. This number is system dependent.)

  **Explanation:** The debugger limit for the length of a string literal has been exceeded. The string literal is too long for the debugger to handle.

  **Programmer response:** Specify a shorter string literal.

- **EQA2231E** A wide string literal cannot be longer than &1 characters.

  **Problem determination:** (where &1 is a number. This number is system dependent.)

  **Explanation:** The debugger limit for the length of a wide string literal has been exceeded. The wide string literal is too long for the debugger to handle.

  **Programmer response:** Specify a shorter string literal.

- **EQA2232E** Invalid "multibyte character sequence character" (MBCS) character.

  **Explanation:** The debugger has detected a multibyte character sequence that it does not recognize.

  **Programmer response:** Replace the "multibyte character sequence character" (MBCS) character.

- **EQA2233E** "&1" is an undefined pure virtual function.

  **Explanation:** The user tried to call a member function that was declared to be a pure virtual function.

  **Programmer response:** Remove or define the function as pure virtual.

- **EQA2234E** Template "&1" cannot be instantiated because the actual argument for formal argument "&2" has more than one variant.

  **Problem determination:** (where &1 is the name of a function template. &2 is the name of a formal template argument.)

  **Explanation:** The argument is a function template or an overloaded function with two or more variants. The debugger cannot decide which variant to choose to bind to the argument type.

  **Programmer response:** Change the formal template argument or remove the extra variants.
EQA2235E  Pointer to a built-in function not allowed.
Explanation:  Because you cannot take the address of a built-in function, you cannot declare a pointer to a built-in function.
Programmer response:  Remove the pointer.

EQA2236E  Built-in function "&1" not recognized.
Problem determination:  (where &1 is the name of a function.)
Explanation:  The function declared as a built-in is not recognized by the debugger as being a built-in function.
Programmer response:  Ensure the function is a built-in function or remove the built-in keyword from the declaration.

EQA2237E  "&1" is not supported.
Problem determination:  (where &1 is a C++ operator)
Programmer response:  Remove the operator from the expression.

EQA2238E  Function calls are not supported.
Explanation:  You can only generate this message in the debugger, when you use an expression that includes a function call.
Programmer response:  Remove function calls from the expression.

EQA2239E  The expression is too complicated.
Programmer response:  Simplify the expression.

EQA2240E  Evaluation of the expression requires a temporary.
Programmer response:  Change the expression so that a temporary object is not required.

EQA2241E  "&1" is an overloaded function.
Problem determination:  (where &1 is the name of a function.)
Explanation:  The identifier refers to an overloaded function with two or more variants. The debugger requires a prototype argument list to decide which variant to process.
Programmer response:  Specify a prototype argument list or remove variants of the overloaded function.

EQA2242E  The bit-field length must not be negative.
Explanation:  The bit-field length must be a nonnegative integer value.
Programmer response:  Change the bit-field length to a nonnegative integer value.

EQA2243E  A zero-length bit-field must not have a name.
Explanation:  A named bit-field must have a positive length; a zero-length bit-field is used for alignment only, and must not be named.
Programmer response:  Remove the name from the zero-length bit-field.

EQA2244E  The bit-field is too small; &1 bits are needed for ";2".
Problem determination:  (where &2 is a C++ name)
Explanation:  The bit-field length is smaller than the number of bits needed to hold all values of the enum.
Programmer response:  Increase the bit-field length.

EQA2245E  The bit-field is larger than necessary; only &1 bits are needed for ";2".
Problem determination:  (where &2 is a C++ name)
Explanation:  The bit-field length is larger than the number of bits needed to hold all values of the enum.
Programmer response:  Decrease the bit-field length.

EQA2246E  A template friend declaration can only declare, not define, a class or function.
Explanation:  The class or function declared in the template friend declaration must be defined at file scope.
Programmer response:  Remove the definition from the template friend declaration.

EQA2247E  The function ";1" must not be declared ";2" at block scope.
Problem determination:  (where &2 is a C++ keyword.)
Explanation:  There can be no static or inline function declarations at block scope.
Programmer response:  Move the function so that it is not defined at block scope.

EQA2248E  The previous &1 messages apply to function argument &2.
Problem determination:  (where &1 is an integer corresponding to the function argument number)
Explanation:  The previous message applies to the specified argument number. This message does not indicate another error or warning, it indicates which argument of the function call is the subject of the previous message.

EQA2249E  The previous &1 messages apply to argument &2 of function ";3".
Problem determination:  (where &1 is the number of messages &2 is the argument number &3 is the function name and type)
Explanation:  The previous message applies to the specified argument number. This message does not indicate another error or warning, it indicates which argument of the function call is the subject of the previous message.

EQA2250E  "&1" is not a static member of ";2".
Problem determination:  (where &2 is a class name.)
Explanation:  Nonstatic data members cannot be defined outside the class definition.
Programmer response:  Make the member a static member or move it into the class definition.
EQA2251E  The initializer must be enclosed in braces.
Explanation: Array element initializers must be enclosed in braces.
Programmer response: Put braces around the initializer.

EQA2252E  &1 is an ambiguous base class of &2.
Explanation: A union can only contain one member object at any time, and therefore can be initialized to only one value.
Programmer response: Remove all but one of the initializers.

EQA2253E  You cannot override virtual function "&1" because "&3" is an ambiguous base class of &2.
Problem determination: (where &3 is the class name of an ambiguous base of &2)
Explanation: The debugger must generate code to convert the actual return type into the type that the overridden function returns (so that calls to the original overridden function is supported). However, the conversion is ambiguous.
Programmer response: Clarify the base class.

EQA2254E  "&1" is not initialized until after the base class is initialized.
Problem determination: (where &1 is the class member referenced in the base class initializer)
Explanation: First, the base classes are initialized in declaration order, then the members are initialized in declaration order, then the body of the constructor is executed.
Programmer response: Do not reference the class member in the base class initializer.

EQA2255E  The expression to the left of the "&1" operator is a relational expression ("&2"). The &3 operator may have been intended.
Problem determination: (where &1 is the bitwise operator | or &.
&2 is one of the relational operators. &3 is either the operator ||
or the operator &&.)
Explanation: The debugger has detected the mixing of relational and bitwise operators in what was determined to be a conditional expression.
Programmer response: Ensure the correct operator is being used.

EQA2256E  The expression to the left of the "&1" operator is a logical expression ("&2"). The &3 operator may have been intended.
Problem determination: (where &1 is the bitwise operator | or &.
&2 is one of the relational operators. &3 is either the operator ||
or the operator &&.)
Explanation: The debugger has detected the mixing of relational and bitwise operators in what was determined to be a conditional expression.
Programmer response: Ensure the correct operator is being used.

EQA2257E  The expression to the left of the "&1" operator is an equality expression ("&2"). The &3 operator may have been intended.
Problem determination: (where &1 is the bitwise operator | or &.
&2 is one of the relational operators. &3 is either the operator ||
or the operator &&.)
Explanation: The debugger has detected the mixing of relational and bitwise operators in what was determined to be a conditional expression.
Programmer response: Ensure the correct operator is being used.

EQA2258E  The expression to the right of the "&1" operator is a relational expression ("&2"). The &3 operator may have been intended.
Problem determination: (where &1 is the bitwise operator | or &.
&2 is one of the relational operators. &3 is either the operator ||
or the operator &&.)
Explanation: This message is generated by the /Wcnd option. This option warns of possible redundancies or problems in conditional expressions involving relational expressions and bitwise operators.
Programmer response: Ensure the correct operator is being used.

EQA2259E  Assignment to the "this" pointer is not allowed.
Explanation: The "this" pointer is a const pointer and cannot be modified.
Programmer response: Remove the assignment to the "this" pointer.

EQA2260E  "&1" must not have any arguments.
Problem determination: (where &1 is a special member function)
Programmer response: Remove all arguments from the special member function.

EQA2261E  The second operand to the "offsetof" operator is not valid.
Explanation: The second operand to the "offsetof" operator must consist only of ":" operators and "[]" operators with constant bounds.
Programmer response: Remove or change the second operand.

EQA2262E  "&1" is a member of "&2" and cannot be used without qualification.
Problem determination: (where &2 is a possibly qualified class name)
Explanation: The specified name is a class member, but no class qualification has been used to reference it.
Programmer response: Use the scope operator (::) to qualify the name.

EQA2263E  sdq.&1 is undefined. Every variable of type "&2" will assume "&1" has no virtual bases and no multiple inheritance.
Problem determination: (where &2 is a pointer to member type)
Explanation: The definition of the class is not given but the
debugger must implement the pointer to member. It will do so by assuming the class has at most one nonvirtual base class.

Programmer response: If this assumption is incorrect, define the class before declaring the member pointer.

EQA2264E "&1" is undefined. The delete operator will not call a destructor.

Problem determination: (where &1 is a name of a class, struct, or union)

Explanation: The definition of the class is not given so the debugger does not know whether the class has a destructor. No destructors will be called.

Programmer response: Define the class.

EQA2265E Label "&1" is undefined.

Problem determination: (where &1 is a C++ name)

Explanation: The specified label is used but is not defined.

Programmer response: Define the label before using it.

EQA2266E The initializer for enumerator "&1" must be an integral constant expression.

Problem determination: (where &1 is an enumerator name)

Explanation: The value of an enumerator must be a constant expression that is promotable to a signed int value. A constant expression has a value that can be determined during compilation and does not change during program execution.

Programmer response: Change the initializer to an integral constant expression.

EQA2267E Overriding virtual function "&1" may not return "&2" because class "&3" has multiple base classes or a virtual base class.

Problem determination: (where &1 is the name of a virtual function &2 is an abstract declarator &3 is the class being returned)

Explanation: Contravariant virtual functions are supported only for classes with single inheritance and no virtual bases.

Programmer response: Ensure the class has single inheritance and no virtual bases.

EQA2268E Virtual function "&1" is not a valid virtual function override because "&3" is an inaccessible base class of "&2".

Problem determination: (where &3 is the class name of an inaccessible base of &2)

Explanation: The debugger must generate code to convert the actual return type into the type that the overridden function returns (so that calls to the original overridden function is supported). However, the target type is inaccessible to the overriding function.

Programmer response: Make the base class accessible.

EQA2269E "&1" is a member of &2 classes. To reference one of these members, "&3" must be qualified.

Problem determination: (where &1 is a C++ member name &2 is an integer greater than 1 &3 is a C++ member name)

Explanation: The class member specified is defined in more than one class nested within the base class and cannot be referenced from the base class if it is not qualified. This message is generated by the /Wund option.

Programmer response: Use the scope operator (:) to qualify the name.

EQA2270E "&1" is not the name of a function.

Problem determination: (where &1 is a name)

Explanation: A function name is required in this context. The specified name has been declared but it is not the name of a function.

Programmer response: Ensure the name is the correctly spelled name of a function.

EQA2271E Enum type "&1" cannot contain both negative and unsigned values.

Explanation: The enumerator type values should fit into an integer. Specifying both unsigned and negative values will exceed this limit.

Programmer response: Remove the negative or unsigned values.

EQA2272E Cannot take the address of the machine-coded function "&1".

Explanation: Because the function is machine-coded, you cannot take its address.

Programmer response: Remove the reference to that function.

EQA2273E An initializer is not allowed for the nonvirtual function "&1".

Problem determination: (where &1 is a function name)

Explanation: The declaration of a pure virtual function must include the keyword virtual.

Programmer response: Remove the initializer.

EQA2274E A local variable or debugger temporary is being used to initialize reference member "&1".

Explanation: The local variable is only active until the end of the function, but it is being used to initialize a member reference variable.

Programmer response: Ensure that no part of your program depends on the variable or temporary.

EQA2275E "&1" is not the SOM® name of a SOM class.

Explanation: A SOM name that represents a SOM class is expected, and was not found. The SOM name of a class might differ from its C++ name.

Programmer response: Ensure that you use the correct SOM name for the class.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
<th>Problem Determination</th>
<th>Explanation</th>
<th>Programmer Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA2276E</td>
<td>Definition of &quot;&amp;1&quot; is only allowed at file scope.</td>
<td>(where &amp;1 is a C++ template class type)</td>
<td>A template class is being defined in a scope other than file scope. Because all template class names have file scope this definition is not allowed.</td>
<td>Move the template class definition to file scope.</td>
</tr>
<tr>
<td>EQA2277E</td>
<td>Class template &quot;&amp;1&quot; cannot be used until its containing template has been instantiated.</td>
<td>(where &amp;1 is a C++ class template type)</td>
<td>The class template referenced cannot be used until the template that contains it has been instantiated.</td>
<td>Declare the class template at file scope or instantiate the template that contains it.</td>
</tr>
<tr>
<td>EQA2278E</td>
<td>Invalid wchar_t value &amp;1.</td>
<td>(where &amp;1 is the value which is not valid)</td>
<td>A multibyte character or escape sequence in a literal has been converted to an invalid value for type wchar_t.</td>
<td>Change the character or escape sequence.</td>
</tr>
<tr>
<td>EQA2279E</td>
<td>The string must be terminated before the end of the line.</td>
<td></td>
<td>The debugger detected a string that was not terminated before an end-of-line character was found.</td>
<td>End the string or use &quot;&quot; to continue the string on the next line. The &quot;&quot; must be the last character on the line.</td>
</tr>
<tr>
<td>EQA2280E</td>
<td>A character constant must end before the end of the line.</td>
<td></td>
<td>The debugger detected a character constant that was not terminated before an end-of-line character was found.</td>
<td>End the character constant or use &quot;&quot; to continue it on the next line. The &quot;&quot; must be the last character on the line.</td>
</tr>
<tr>
<td>EQA2281E</td>
<td>A matching &amp;1 function named &quot;&amp;2&quot; could not be found.</td>
<td>(where &amp;1 is one of 'const', 'volatile' or 'const volatile'. &amp;2 is the name of the called function (without the argument list).)</td>
<td>The call might have failed because no member function exists that accepts the 'const/volatile' qualifications of the object.</td>
<td>Ensure the type qualifier is correct and that the function name is spelled correctly.</td>
</tr>
<tr>
<td>EQA2282E</td>
<td>&quot;&amp;1&quot; is a type name being used where a variable name is expected.</td>
<td>(where &amp;1 is a C and C++ name)</td>
<td>The identifier must be a variable name not a type name.</td>
<td>Check that the identifier is a variable name and ensure the variable is not hidden by a type name.</td>
</tr>
<tr>
<td>EQA2283E</td>
<td>Template &quot;&amp;1&quot; has a missing or incorrect template argument list.</td>
<td>(where &amp;1 is a C++ name)</td>
<td>A template friend declaration must declare a class or a function following the template arguments.</td>
<td>Complete the template argument list or change the identifier to a variable name.</td>
</tr>
<tr>
<td>EQA2284E</td>
<td>Template friend declaration does not declare a class or a function.</td>
<td></td>
<td>A template friend declaration must declare a class or a function.</td>
<td>Change the template declaration to declare a class or a function.</td>
</tr>
<tr>
<td>EQA2285E</td>
<td>The 'const' object has been cast to a non-'const' object.</td>
<td></td>
<td>A cast has been used to possibly modify a 'const' object. This might cause undefined behaviour at run time.</td>
<td>Remove the cast or make the object nonconst.</td>
</tr>
<tr>
<td>EQA2286E</td>
<td>Global friend functions may not be defined in a local class.</td>
<td></td>
<td>A local class cannot have a friend function.</td>
<td>Make the function a member function in the local class.</td>
</tr>
<tr>
<td>EQA2287E</td>
<td>The address of data member &quot;&amp;1&quot; cannot be taken because the member is being referenced through a <em>get</em> function.</td>
<td></td>
<td>An attribute is access through a &quot;<em>get</em>&quot; method if its backing data is not accessible, or if the SOMNoDataDirect pragma is in effect for the class. Since the &quot;<em>get</em>&quot; method returns the value of the member, and not its address, it isn't possible to use the address operator &quot;&amp;&quot; on the member to create an ordinary pointer. This error can also be generated if you haven't used the &quot;&amp;&quot; operator explicitly, but the debugger needs to use it to implement your code. You can create a pointer-to-member that refers to an attribute.</td>
<td>Rewrite the expression that causes the address to be taken, or remove the SOMAttribute pragma.</td>
</tr>
</tbody>
</table>
### EQA2288S • EQA2316E

<table>
<thead>
<tr>
<th>Code</th>
<th>Error Message</th>
<th>Explanations</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA2288S</td>
<td>The module &quot;module_name&quot; cannot be loaded - it was NOT found.</td>
<td>Explanation: The LOAD MODULE request failed, the module was not found.</td>
</tr>
<tr>
<td>EQA2289S</td>
<td>The module &quot;module_name&quot; cannot be loaded - there is NOT enough storage to do the load.</td>
<td>Explanation: The LOAD MODULE request failed, there is not enough storage.</td>
</tr>
<tr>
<td>EQA2290S</td>
<td>The module &quot;module_name&quot; cannot be loaded.</td>
<td>Explanation: The LOAD MODULE request was unsuccessful.</td>
</tr>
<tr>
<td>EQA2291S</td>
<td>The module &quot;module_name&quot; cannot be deleted.</td>
<td>Explanation: The DELETE MODULE request was unsuccessful.</td>
</tr>
<tr>
<td>EQA2292S</td>
<td>The module &quot;module_name&quot; was NOT loaded by the Debug Tool and therefore CANNOT be deleted.</td>
<td>Explanation: Only modules loaded by Debug Tool may be deleted.</td>
</tr>
<tr>
<td>EQA2293S</td>
<td>The module &quot;module_name&quot; cannot be loaded because it was already loaded by Debug Tool.</td>
<td>Explanation: The LOAD MODULE request was unsuccessful.</td>
</tr>
<tr>
<td>EQA2294I</td>
<td>The LOADDEBUGDATA command for CU_name has been deferred until the CU appears.</td>
<td>Explanation: The indicated CU is not currently known to Debug Tool. The LOADDEBUGDATA will be run when the CU appears in a loaded module.</td>
</tr>
<tr>
<td>EQA2295I</td>
<td>The CU specified for the LOADDEBUGDATA command is already an assembler or OS/VS COBOL CU.</td>
<td>Explanation: An L00 has already been done for the CU specified in the L00 command. This L00 may have been done previously by the user or an implicit L00 may have been done for the CU. This happens when a user-entered L00 is successful and, subsequently, the CU goes away and later reappears.</td>
</tr>
<tr>
<td>EQA2296E</td>
<td>The CU specified for the LOADDEBUGDATA command is not a disassembly CU.</td>
<td>Explanation: Only a disassembly CU can be identified as assembler CU.</td>
</tr>
<tr>
<td>EQA2297E</td>
<td>An error occurred while attempting to load the debug (EQALANGX) file for a specified CU.</td>
<td>Explanation: Either the file containing the EQALANGX debug data could not be found or there was an undetermined error loading the EQALANGX file for a specified CU.</td>
</tr>
<tr>
<td>EQA2298S</td>
<td>There is no compilation unit named compile_unit_name.</td>
<td>Explanation: The compilation unit (program) that you named could not be located by Debug Tool.</td>
</tr>
<tr>
<td>EQA2299E</td>
<td>The EQALANGX file does not match the object for object_name. The EQALANGX file cannot be used.</td>
<td>Explanation: An EQALANGX file containing the assembler debugging information does not match the object. Either the CSECT length is different, selected object code is different, or the EQALANGX file is not for the correct language.</td>
</tr>
<tr>
<td>EQA2300E</td>
<td>A pattern of &quot;*&quot; is invalid.</td>
<td>Explanation: The NAMES EXCLUDE command does not allow a pattern of &quot;.&quot;</td>
</tr>
<tr>
<td>EQA2301E</td>
<td>Value too long to display.</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>EQA2317E</td>
<td>Invalid value for update</td>
<td></td>
</tr>
<tr>
<td>EQA2318E</td>
<td>Update of value not allowed at this time</td>
<td></td>
</tr>
<tr>
<td>EQA2319E</td>
<td>Update of value not supported</td>
<td></td>
</tr>
<tr>
<td>EQA2320E</td>
<td>Operation not supported</td>
<td></td>
</tr>
<tr>
<td>EQA2321I</td>
<td>Please see log window for messages</td>
<td></td>
</tr>
<tr>
<td>EQA2322E</td>
<td>Invalid address</td>
<td></td>
</tr>
<tr>
<td>EQA2323E</td>
<td>Storage unit style is not supported</td>
<td></td>
</tr>
<tr>
<td>EQA2324E</td>
<td>Storage address style is not supported</td>
<td></td>
</tr>
<tr>
<td>EQA2325E</td>
<td>Register variable(s) out of range</td>
<td></td>
</tr>
<tr>
<td>EQA2326E</td>
<td>Invalid program name</td>
<td></td>
</tr>
<tr>
<td>EQA2327E</td>
<td>Failure loading view information</td>
<td></td>
</tr>
<tr>
<td>EQA2328E</td>
<td>Failure evaluating expression context. Contact IBM support.</td>
<td></td>
</tr>
<tr>
<td>EQA2329E</td>
<td>Frequency data is not allowed with this breakpoint type.</td>
<td></td>
</tr>
<tr>
<td>EQA2330E</td>
<td>Invalid line to set a line breakpoint</td>
<td></td>
</tr>
<tr>
<td>EQA2331E</td>
<td>Initialization failure - Debug Tool and front-end levels are incompatible</td>
<td></td>
</tr>
<tr>
<td>EQA2332E</td>
<td>Invalid storage data</td>
<td></td>
</tr>
<tr>
<td>EQA2333E</td>
<td>Incomplete update - a portion of storage is not updateable</td>
<td></td>
</tr>
<tr>
<td>EQA2334E</td>
<td>Storage is not updateable</td>
<td></td>
</tr>
<tr>
<td>EQA2335E</td>
<td>Failure updating storage contents</td>
<td></td>
</tr>
<tr>
<td>EQA2336E</td>
<td>Procedure name is an internal procedure, not an entry point</td>
<td></td>
</tr>
<tr>
<td>EQA2337E</td>
<td>Breakpoint type not supported</td>
<td></td>
</tr>
<tr>
<td>EQA2338I</td>
<td>Program at end of job</td>
<td></td>
</tr>
<tr>
<td>EQA2339E</td>
<td>A DLL load occurred. Current program location cannot be determined</td>
<td></td>
</tr>
<tr>
<td>EQA2340I</td>
<td>Target program(s) loaded - START/CALL required</td>
<td></td>
</tr>
<tr>
<td>EQA2341E</td>
<td>Insufficient storage to load view</td>
<td></td>
</tr>
<tr>
<td>EQA2342E</td>
<td>Program exception has occurred</td>
<td></td>
</tr>
<tr>
<td>EQA2343I</td>
<td>Debug session initialization complete</td>
<td></td>
</tr>
<tr>
<td>EQA2344E</td>
<td>Debug file name could not be found</td>
<td></td>
</tr>
<tr>
<td>EQA2345E</td>
<td>Debug file version not supported</td>
<td></td>
</tr>
<tr>
<td>EQA2346I</td>
<td>Maximum number of debug files (256) reached</td>
<td></td>
</tr>
<tr>
<td>EQA2347E</td>
<td>Invalid debug file name</td>
<td></td>
</tr>
<tr>
<td>EQA2348E</td>
<td>Debug file format is invalid</td>
<td></td>
</tr>
<tr>
<td>EQA2349E</td>
<td>Debug file not supported - contains multiple <code>@PROCESS</code> statements</td>
<td></td>
</tr>
<tr>
<td>EQA2350E</td>
<td>Insufficient storage to read debug file</td>
<td></td>
</tr>
<tr>
<td>EQA2351E</td>
<td>I/O error reading debug file</td>
<td></td>
</tr>
<tr>
<td>EQA2352E</td>
<td>I/O error opening debug file</td>
<td></td>
</tr>
<tr>
<td>EQA2353E</td>
<td>Debug file CSECT name does not match compile unit CSECT name</td>
<td></td>
</tr>
<tr>
<td>EQA2354I</td>
<td>USE file processing has paused - USE file is still active</td>
<td></td>
</tr>
<tr>
<td>EQA2355E</td>
<td>Altering the PSW is not supported</td>
<td></td>
</tr>
<tr>
<td>EQA2356E</td>
<td>Program not auto-started - debug file name could not be found</td>
<td></td>
</tr>
<tr>
<td>EQA2357E</td>
<td>Program not auto-started - debug file version not supported</td>
<td></td>
</tr>
<tr>
<td>EQA2358E</td>
<td>Program not auto-started - max number of debug files (256) reached</td>
<td></td>
</tr>
<tr>
<td>EQA2359E</td>
<td>Program not auto-started - invalid debug file name</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>EQA2360E</td>
<td>Program not auto-started - debug file format is invalid</td>
<td></td>
</tr>
<tr>
<td>EQA2361E</td>
<td>Program not auto-started - insufficient storage</td>
<td></td>
</tr>
<tr>
<td>EQA2362E</td>
<td>Program not auto-started - I/O error reading debug file</td>
<td></td>
</tr>
<tr>
<td>EQA2363E</td>
<td>Program not auto-started - I/O error opening debug file</td>
<td></td>
</tr>
<tr>
<td>EQA2364E</td>
<td>Program not auto-started - failure processing debug information</td>
<td></td>
</tr>
<tr>
<td>EQA2365E</td>
<td>Exit point name must be a primary entry</td>
<td></td>
</tr>
<tr>
<td>EQA2366E</td>
<td>Exit breakpoints not allowed on internal procedures</td>
<td></td>
</tr>
<tr>
<td>EQA2367E</td>
<td>Failure processing debug information. Program compiled with NOTEST</td>
<td></td>
</tr>
<tr>
<td>EQA2368I</td>
<td>Search has wrapped</td>
<td></td>
</tr>
<tr>
<td>EQA2369E</td>
<td>Invalid link pointer</td>
<td></td>
</tr>
<tr>
<td>EQA2370I</td>
<td>End of link chain reached</td>
<td></td>
</tr>
<tr>
<td>EQA2371E</td>
<td>Invalid breakpoint label name</td>
<td></td>
</tr>
<tr>
<td>EQA2372E</td>
<td>Jump to location must be within currently active program</td>
<td></td>
</tr>
<tr>
<td>EQA2373E</td>
<td>Initialization failure - extended qualification is not supported</td>
<td></td>
</tr>
<tr>
<td>EQA2374E</td>
<td>SOM object not instantiated</td>
<td></td>
</tr>
<tr>
<td>EQA2375E</td>
<td>Field is not updatable</td>
<td></td>
</tr>
<tr>
<td>EQA2376E</td>
<td>Update of field failed</td>
<td></td>
</tr>
<tr>
<td>EQA2377E</td>
<td>Invalid data</td>
<td></td>
</tr>
<tr>
<td>EQA2378I</td>
<td>Application has terminated</td>
<td></td>
</tr>
<tr>
<td>EQA2379E</td>
<td>Internal error. Please, contact IBM support</td>
<td></td>
</tr>
<tr>
<td>EQA2380E</td>
<td>Jump to Location while at initialization is not allowed</td>
<td></td>
</tr>
<tr>
<td>EQA2381E</td>
<td>The target of the Jump to Location is invalid</td>
<td></td>
</tr>
<tr>
<td>EQA2382E</td>
<td>The target of the Run to Location is invalid</td>
<td></td>
</tr>
<tr>
<td>EQA2383I</td>
<td>The environment is not yet fully initialized. Use Step or Run.</td>
<td></td>
</tr>
<tr>
<td>EQA2384E</td>
<td>The Defer option is not permitted for Line Breakpoints.</td>
<td></td>
</tr>
<tr>
<td>EQA2385E</td>
<td>The entry point is not active or debug data is not available.</td>
<td></td>
</tr>
<tr>
<td>EQA2386E</td>
<td>You are monitoring an inaccessible or uninitialized variable.</td>
<td></td>
</tr>
<tr>
<td>EQA2387E</td>
<td>Cannot modify BreakPoints. Delete existing BreakPoint then add new.</td>
<td></td>
</tr>
<tr>
<td>EQA2388E</td>
<td>Cannot monitor this type of expression or variable</td>
<td></td>
</tr>
<tr>
<td>EQA2389E</td>
<td>The C component of the LE runtime has not been initialized.</td>
<td></td>
</tr>
<tr>
<td>EQA2390E</td>
<td>Exec Imminent. Click on any button to continue.</td>
<td></td>
</tr>
<tr>
<td>EQA2391E</td>
<td>Time stamp on listing does not match time stamp on object.</td>
<td></td>
</tr>
<tr>
<td>EQA2392E</td>
<td>Unable to find requested executable module.</td>
<td></td>
</tr>
<tr>
<td>EQA2393E</td>
<td>Debug Tool has frozen this thread. Registers are not available.</td>
<td></td>
</tr>
<tr>
<td>EQA2394E</td>
<td>Stop at date field references.</td>
<td></td>
</tr>
<tr>
<td>EQA2395E</td>
<td>The entered expression is invalid, please check the expression.</td>
<td></td>
</tr>
<tr>
<td>EQA2396E</td>
<td>Expression breakpoints are not supported for this language.</td>
<td></td>
</tr>
<tr>
<td>EQA2397E</td>
<td>The expression has inaccessible or uninitialized data.</td>
<td></td>
</tr>
<tr>
<td>EQA2398E</td>
<td>No hooks present at statements, breakpoint not set.</td>
<td></td>
</tr>
<tr>
<td>EQA2399E</td>
<td>Debug Tool has frozen this thread. Call stack is not available.</td>
<td></td>
</tr>
</tbody>
</table>
EQA2400E • EQA2417I

EQA2400E
Operations involving engine settings are currently not supported.

EQA2401E
This register cannot be edited. Changes have been ignored.

EQA2402I
Program was stopped due to load occurrence breakpoint.

EQA2403I
Program was stopped due to storage change breakpoint.

EQA2404E
The debug information is not accessible for the requested thread.

EQA2405E
The PL/I component of the LE runtime has not been initialized.

EQA2406E
The Cobol component of the LE runtime has not been initialized.

EQA2407E
Too many local variables for local monitor. Use program monitor instead.

EQA2408E
Variable cannot be displayed because this compile unit was compiled without symbolic information.

Explanation: The current compile unit was compiled without symbolic information. Variable information is not accessible to Debug Tool. The CU must be recompiled with TEST to provide this information.

EQA2409E
The LAST option on an EXEC CICS SEND command has been suppressed.

Explanation: The application has issued an EXEC CICS SEND command with the LAST Option while being debugged in single terminal mode. This would end the terminal session being used by Debug Tool, so Debug Tool has suppressed the LAST option.

Programmer response: If the LAST option needs to be exercised, consider debugging the application in dual terminal mode.

EQA2410E
Search target not found.

Explanation: The search string was not found.

EQA2411E
Variable needs further qualification or qualification is invalid.

Explanation: A qualified reference is invalid. One or more qualifiers may be undefined or not in the same structure as the desired data item.

EQA2412I
You are currently on an instruction that will leave the current Compile Unit and may cause Debug Tool to lose control. You must ensure that a breakpoint is set on a subsequent instruction or statement. At location LOCN.

Explanation: While stepping through a program in the disassembly view, you are about to execute an instruction that will cause a transfer (branch) out of the current Compile Unit. A breakpoint is required where you would like to obtain control.

Programmer response: Set a breakpoint and/or enter STEP or GO to continue.

EQA2413I
You are currently on an instruction that must run without a breakpoint. Debug Tool may lose control. You must ensure that a breakpoint is set on a subsequent instruction or statement. At location LOCN.

Explanation: While stepping through a program in the disassembly view, you are about to execute an instruction that must run from the original location and therefore the breakpoint must be temporarily removed. A breakpoint is required where you would like to obtain control.

Programmer response: Set a breakpoint and/or enter STEP or GO to continue.

EQA2414I
You are currently on an instruction that is the target of an EX instruction or one that is not allowed to have a breakpoint. A breakpoint should be set on the EX or a subsequent instruction or statement. This breakpoint is removed at location LOCN.

Explanation: While stepping through a program in the disassembly view, an instruction was encountered in an unsupported location or an instruction that is not allowed to have a breakpoint was found to have a breakpoint. A breakpoint is required where you would like to obtain control. The breakpoint is automatically removed.

Programmer response: Set a breakpoint and/or enter STEP or GO to continue.

EQA2415I
Debug Tool could not stop at one or more instructions because a valid save area backchain was not found. At location LOCN.

Explanation: While debugging a program, Debug Tool could not stop the application because a valid save area backchain did not exist. The backchain pointer is located at +4 in the save area pointed to by register 13. This will most likely occur when stepping through the prolog code of a Compile Unit.

EQA2416I
Debug Tool detected a missing or invalid Debug Tool SVC EQA005VC(GC0014E). The installed version is 977. DYNDEBUG is disabled.

Explanation: During initialization Debug Tool did not find a usable Dynamic Debug facility SVC. Either the SVC was not found or a downlevel version was detected. The Dynamic Debug facility is disabled.

Programmer response: Have your installer install the correct Dynamic Debug facility SVC.

EQA2417I
Not enough memory available for PLAYBACK data collection. You must DISABLE PLAYBACK.

Explanation: The run-time API did not have enough memory to save application DATA during Playback data collection.

Programmer response: Use the PLAYBACK DISABLE command to disable Playback. You can then re-start Playback and specify more memory for use by Playback on the PLAYBACK ENABLE command.
EQA2418I  Return code RC from PLAYBACK run-time API API name. You must DISABLE PLAYBACK.

Explanation: The run-time API is no longer able to collect application DATA. The return code indicates a terminal error in the run-time.

Programmer response: Use the PLAYBACK DISABLE command to disable Playback. Return code 63 indicates not enough memory was available for Playback. Restart your Debug Tool session and enter the PLAYBACK ENABLE command using the integer option. For example: PLAYBACK ENABLE * 10000.

EQA2419W  Playback data collection has wrapped. Earlier data has been overlayed.

Explanation: Playback data collection has used all available memory. The earliest collected data will be overlayed with newer data.

Programmer response: If it is necessary to retain more Playback data, specify a larger memory size on the PLAYBACK ENABLE command.

EQA2420W  The assignment was performed but the assigned value might not be used by the program, due to optimization.

Explanation: Results might be unreliable because the program might use the previous value that was saved in temporary storage or a register.

Programmer response: Recompile the program without the Optimize option.

EQA2421E  The assignment was not performed because the assigned value might not be used by the program, due to optimization.

Explanation: Results would be unreliable because the program might use the previous value that was saved in temporary storage or a register.

Programmer response: You can SET WARNING OFF to allow the update to take place or recompile the program without optimization.

EQA2422E  A breakpoint cannot be set on this statement when the STORAGE runtime option is in effect. Remove STORAGE or set the breakpoint after the next LR R13,Rx instruction.

Explanation: When the STORAGE runtime option is in effect, breakpoints are not allowed on the prologue instructions between the first BALR R14,R15 and the next LR R13,Rx. You may set a breakpoint on an instruction following the next LR R13,Rx or you may rerun your program without the STORAGE runtime option and set a breakpoint on the specified statement.

EQA2423S  A AbendCode ABEND occurred.

Explanation: The indicated System or User ABEND was detected.

Programmer response: Investigate the cause of the ABEND.

EQA2424I  Debug Tool detected a missing or invalid Debug Tool SVC EQA01SVC(IGX00051). The installed version is NFRS. DYNDEBUG is disabled for read only programs.

Explanation: During initialization Debug Tool did not find a usable Dynamic Debug facility SVCs EQA00SVC(IGC0014E) and EQA01SVC(IGX00051). Either the SVC was not found or a downlevel version was detected. The Dynamic Debug facility is disabled for read only programs. Other Dynamic Debug facilities may not operate correctly.

Programmer response: Have your installer install the correct Dynamic Debug facility SVC.

EQA2425I  Debug Tool detected a down level Debug Tool SVC EQA01SVC(IGX00051). The installed version is NFRS. Version 2 is required when using CICS with TRANISO=YES. DYNDEBUG is disabled.

Explanation: During initialization Debug Tool detected a downlevel Dynamic Debug facility SVC version. The Dynamic Debug facility is disabled for read only programs.

Programmer response: Have your installer install the correct Dynamic Debug facility SVC.

EQA2426I  Debug Tool detected a down level Debug Tool SVC EQA01SVC(IGX00051). The installed version is NFRS. Version 4 is required for read only program performance improvement.

Explanation: During initialization Debug Tool detected a downlevel Dynamic Debug facility SVC version. The Dynamic Debug facility is disabled for read only programs.

Programmer response: Have your installer install the correct Dynamic Debug facility SVC.

EQA2427I  Debug Tool detected a down level Debug Tool SVC EQA01SVC(IGX00051) or EQA00SVC(IGC0014E). EQA01SVC is version VERS. EQA00SVC is version VERS. EQA01SVC version 5 and EQA00SVC version 4 are required for NON-LE programs.

Explanation: During initialization Debug Tool detected a downlevel Dynamic Debug facility SVC version.

Programmer response: Have your installer install the correct Dynamic Debug facility SVC.

EQA2428E  This command is either invalid or unsupported.

EQA2429E  This command is not supported.

EQA2430E  This command is not supported with this UI.

EQA2431I  Automonitor is on for this debug session.

EQA2432I  Automonitor is off.
**EQA2433E** Load Debug Data failed:

**EQA2434E** Unknown CU:

**EQA2435I** This program has no statement table.

**EQA2436I** The statement table has the STMT format.

**EQA2437I** The statement table has the NUMBER format.

**EQA2438I** The statement table has the SHORT format.

**EQA2439I** The program was compiled with the following options:

**EQA2440I** Program Information for this compile unit is:

**EQA2441I** IBM Debug Tool Version 7 Release 1 Mod 0

**Explanation:** This message is used to place the Debug Tool logo, a time stamp, and copyright at the beginning of the line.

**EQA2442I** 5655-R44 and 5655-R45: (C) Copyright IBM Corp. 1992, 2006

**Explanation:** This message is used to place the Debug Tool logo, a time stamp, and copyright at the beginning of the line.

**EQA2443I** Assembler debug mode is active.

**EQA2444I** Assembler debug mode is no longer active.

**EQA2445I** Disassembly debug mode is now active.

**EQA2446I** Disassembly debug mode is no longer active.

**EQA2447I** The setting of LOG is on.

**EQA2448I** The setting of LOG is off.

**EQA2449I** Dynamic debug mode is on.

**EQA2450I** Dynamic debug mode is off.

**EQA2451I** The setting of WARNING is on.

**EQA2452I** The setting of WARNING is off.

**EQA2453E** SET DYNDEBUG can not be executed at this time. SET DYNDEBUG can only be executed at the beginning of a debugging session, before any STEP or GO commands. The DYNDEBUG status has not been changed.

**EQA2454I** This CU is not AUTOMONITOR capable for expressions.

**Programmer response:** Refer to the description of the SET AUTOMONITOR command in the Reference and Messages document to determine the requirements this CU must fulfill in order to use the SET AUTOMONITOR command.

**EQA2455E** Program uses non-standard linkage. R13 contains an invalid address.

**Explanation:** Debug Tool has stopped in a program and the value in GPR 13 is not a valid address.

**Programmer response:** Debug Tool will attempt to continue. However, some information may be missing or incorrect.

**EQA2456I** Returning from enclave where Debug Tool was initialized. Handling of non-LE events has been suspended. Debugging of non-LE programs is suspended in this Debug Tool session.

**Explanation:** The Language Environment was invoked with TEST/NOPROMPT or NOTEST causing Debug Tool to be initialized during an enclave that was not the top enclave. The enclave in which Debug Tool was initialize is now terminating. Debug Tool will no longer intercept non-LE events and, therefore, you can no longer debug non-LE programs.

**Programmer response:** If you do not need to debug higher-level, non-LE programs or to intercept non-LE events, no action is required. Otherwise, re-run the job without NOPROMPT or with the CALL to CEETEST, PLITEST or ctst() in a higher-level enclave.

**EQA2457I** This function requires Debug Tool Utilities and Advanced Functions.

**EQA2458I** SVC Screening is disabled by EQAOPTS. Handling of non-LE events is not available. Debugging of non-LE programs will be restricted in this Debug Tool session.

**Explanation:** The Debug Tool was invoked with an EQAOPTS options module that specified SVCScreen OFF. Debug Tool will not intercept non-LE events and, therefore, debugging of non-LE programs will be limited.

**Programmer response:** If you do not need to debug non-LE programs or to intercept non-LE events, no action is required. Otherwise, you must have your installer provide an EQAOPTS that specifies SVCScreen ON.

**EQA2459I** SVC Screening is in use by another product and SVC Screening CONFLICT=NOOVERRIDE is specified by EQAOPTS. Handling of non-LE events is not available. Debugging of non-LE programs will be restricted in this Debug Tool session.

**Explanation:** The Debug Tool was invoked with an EQAOPTS options module that specified CONFLICT=NOOVERRIDE. Debug Tool will not intercept non-LE events and, therefore, debugging of non-LE program will be limited.

**Programmer response:** If you do not need to debug non-LE programs or to intercept non-LE events, no action is required. Otherwise, you must terminate the prior use of SVC SCREENING (TCBVCSC, TCBVCSE, TCBVCA2) before starting Debug Tool or have your installer provide an EQAOPTS that specifies...
CONFLICT=OVERRIDE. CONFLICT=OVERRIDE allows Debug Tool to save and restore the previous use of SVC SCREENING (TCBSVCS, TCBSVCSF, TCBSVCA2).

EQA2465I SVC Screening is in use by another product and SVC Screening CONFLICT=OVERRIDE is specified by EQAOPTS. Previous use of SVC Screening will be restored at the end of this Debug Tool session.

Explanation: The Debug Tool was invoked with an EQAOPTS options module that specified CONFLICT=OVERRIDE. Debug Tool will save and restore the SVC Screening values.

SVC SCREENING is indicated by TCBSVCS, TCBSVCSF, and TCBSVCA2. These values are saved during Debug Tool startup and restored at Debug Tool termination.

EQA2461W Code page in the VADSCP suboption is not between 00001 and 99999. Default code page 00037 is assumed.

Explanation: Debug Tool was invoked with an invalid VADSCP suboption in the TEST runtime option string. Internal conversion tables for the default code page 00037 are used for translation between Debug Tool and the distributed debugger.

Programmer response: Correct the VADSCP suboption and restart the debug session.

EQA2462W Code page conversion of string failed. z/OS Unicode conversion services return code is VERS and reason code is VERS.

Explanation: Problem encountered in a code page conversion using z/OS Unicode conversion services. Internal conversion tables for the default code page 00037 are used.


EQA2463W z/OS Unicode conversion services encountered a serious problem. Default code page 00037 is used in the debug session.

Explanation: z/OS Unicode conversion services failed. Internal conversion tables for the default code page 00037 are used in the debug session.

Programmer response: See z/OS Support for Unicode: Using Conversion Services book for explanation of return code, reason code, and appropriate action in the accompanied EQA2462W message. A typical problem is that the conversion images are not available. Consult with your system programmer to see what is available on the system.

EQA2464I There are no & & & & & & names currently & & & & & & by & & & & & &.

Explanation: This message is issued by the NAMES command when there is no data to be displayed.

EQA2465I The following & & & & & & names are currently & & & & & & by & & & & & &.

Explanation: This message precedes the output of the NAMES command and indicates the type of names that follow this message.
EQA2474I • EQA2487S

EQA2474I Debug Tool could not stop at one or more instructions because the program is executing with a PSW that specifies AMODE 64. At location LOCN.
Explanation: During execution of the program Debug Tool encountered a hook or other event when the PSW specified AMODE64. Debug Tool will ignore all events that occur in this state and the program will continue to execute.
Programmer response: None.

EQA2475I An implicit CU was created for '&&&&&' in '&&&&'.
Explanation: An implicit CU was created for the specified CU in the specified load module as the result of a QUALIFY CU command for a load module that was already loaded and a COBOL CU that had not yet been executed. The CU will be destroyed when execution is resumed.
Programmer response: None.

EQA2476I An implicit LOAD was issued for module '&&&'.
Explanation: An implicit LOAD command was issued for the specified load module as the result of a QUALIFY LOAD or QUALIFY CU command for a load module that was not currently loaded. A corresponding CLEAR LOAD will be issued when execution is resumed.
Programmer response: None.

EQA2477I An implicit CLEAR LOAD was issued for module '&&&'.
Explanation: An implicit CLEAR LOAD command was issued for the specified load module. This module was previously LOADed as the result of a QUALIFY LOAD or QUALIFY CU command for a load module that was not currently loaded.
Programmer response: None.

EQA2478I Implicit CU '&&&&&' in '&&&&&' is being destroyed.
Explanation: The specified implicitly created CU is being destroyed. This module was previously created as the result of a QUALIFY CU command for a CU that had not already been created.
Programmer response: None.

EQA2479S CU '&&&&&' cannot be destroyed.
Explanation: The attempt to destroy the specified CU was unsuccessful.
Programmer response: None.

EQA2480E 'symbol' is an undefined symbol.
Explanation: The specified symbol was used in an assembler expression. However, it is not a valid symbol in the current compile unit.
Programmer response: Correct the assembler expression and retry the command.

EQA2481E Invalid syntax in expression at or near 'expression.fragment'.
Explanation: Invalid syntax was discovered at or near the part of the expression shown in the error message.
Programmer response: Correct the assembler expression and retry the command.

EQA2482E Invalid expression at or near 'expression.fragment'.
Explanation: The specified expression is invalid in the current context.
Programmer response: Correct the assembler expression and retry the command.

EQA2483E 'operator' is an invalid operator.
Explanation: The specified operator is not valid in an assembler expression.
Programmer response: Correct the assembler expression and retry the command.

EQA2484E A relational expression is not allowed in the current context or a relational expression was not found where one was expected.
Explanation: A relational expression (containing an operator such as =, ^=, <=, etc.) was found in an unexpected context or was not found where one was expected.
Programmer response: Correct the assembler expression and retry the command.

EQA2485S An internal error has occurred processing an assembler expression.
Explanation: An internal Debug Tool error has occurred processing an assembler expression.
Programmer response: Report this error to your IBM representative.

EQA2486S The source and receiver are not compatible for assignment.
Explanation: An assembler assignment contain a source and receiver that are not compatible for assignment. For example, a string longer than four bytes cannot be assigned to an arithmetic receiver.
Programmer response: Correct the assignment operands and retry the command.

EQA2487S A runtime error occurred processing an assignment statement. The source or receiver may not be addressable.
Examination: An error such as an addressing error occurred while Debug Tool was attempting to execute an assignment. Either the source or receiver may be incorrectly specified.
Programmer response: Report this error to your IBM representative.
EQA2488W • EQA2506E

EQA2488W  An LDD command was executed for a Disassembler CU that contained user breakpoints. These breakpoints have been removed.

Explanation: The Disassembler breakpoints no longer apply to an assembler program. The user should reapply any required breakpoints.

Programmer response: Set appropriate breakpoints.

EQA2489E  Symbol is an ambiguous or incorrectly qualified reference.

Explanation: The specified symbol was used in an OS/VS COBOL expression. However, either there are more than one variable by this name and the reference was not sufficiently qualified to disambiguate the reference or the qualification specified was invalid.

Programmer response: Correctly qualify the referenced variable and retry the command.

EQA2490I  The DEFAULT VIEW is now &amp;&amp;&amp;.

EQA2491I  The setting of DEFAULT VIEW is &amp;&amp;&amp;.

EQA2492I  The setting of current view is &amp;&amp;&amp;.

EQA2493I  Program was stopped due to watch breakpoint on COBOL Level-88 condition-name &amp;&amp;&amp;.

EQA2494I  Program was stopped due to watch breakpoint on COBOL Level-88 condition-name &amp;&amp;&amp; and the condition &amp;&amp;&amp; evaluated to be true.

EQA2495I  Evaluation of the conditional expression &amp;&amp;&amp; failed.

EQA2496I  The setting of LDD is &amp;&amp;&amp;.

Explanation: This message is issued by the remote interface in response to the SET LDD command.

Programmer response: None.

EQA2497I  The setting of LDD is &amp;&amp;&amp;.

Explanation: This message is issued by the remote interface in response to the QUERY LDD command.

Programmer response: None.

EQA2498I  LOAD is not supported during initialization. STEP or GO and retry LOAD command.

Explanation: A LOAD request cannot be issued until Debug Tool initialization is complete.

EQA2499I  A load module that was loaded as the result of a LOAD command contains LE CUs but LE is not active. The LE CUs will not be created.

Explanation: A LOAD command was used in a non-LE environment to load a load module that contained one or more LE CUs. These CUs cannot be created until LE is active. CUs will be created only for non-LE CUs in this load module.

EQA2500E  Incorrect or missing data

Explanation: The data at the cursor location is either incorrect or some data is missing. There could be several reasons for this:
1. Invalid combination of options specified.
2. Invalid data for field.
3. Data not entered, when required by options given.
4. Quotes specified when not allowed.

Programmer response: Correct the entry where the cursor is positioned and invoke the function again. You can use Help (PF1) to find the context sensitive help for that field.

EQA2501E  DTCN internal error

Explanation: DTCN discovered an internal error.

Programmer response: Contact IBM service.

EQA2502E  Internal CICS error

Explanation: During processing, DTCN discovered an internal CICS error.

Programmer response: Correct the error and issue the command again. If the error persists, contact your CICS system programmer and/or IBM service.

EQA2503E  Key Not Defined.

Explanation: There is no action defined with the PF key used by the user.

Programmer response: Use the keys displayed in the bottom line. For more information about the actions defined for this panel, use PF2 key for general help.

EQA2504E  Add failed - profile does not exist

Explanation: The add command failed because a profile for that terminal is already stored in the Debug Tool Profile Repository.

Programmer response: You can use Show(PF7) command to display the profile or modify the TermId+TranId and Add a new profile.

EQA2505E  Replace failed - profile does not exist

Explanation: The profile for that terminal does not exist in the Debug Tool Profile Repository and cannot be updated. Specify different terminal to update.

Programmer response: You can use Next(PF8) command to browse the Profile Repository starting from any point.

EQA2506E  Delete failed - profile does not exist

Explanation: The profile for the terminal does not exist in the Debug Tool Profile Repository and cannot be updated.

Programmer response: Specify different Terminal+Transaction Id to delete. You can use Next(PF8) command to browse the Profile Repository starting from any point.
EQA2507E • EQA3001I

EQA2507E  Show failed - profile does not exist
Explanation:  The profile for the Terminal does not exist in the Debug Tool Profile Repository.

Programmer response:  Specify different Terminal to display. You can use NEXT (PF8) command to browse the Profile Repository from any point.

EQA2508E  Next failed - profile does not exist
Explanation:  There are no more profiles in the Debug Tool Profile Repository.

EQA2510I  DTCN closed
Explanation:  DTCN deleted all profiles stored in the Debug Tool Profile Repository. This action affects all users working with that CICS region.

EQA2511E  Specify at least one resource to debug
Explanation:  DTCN needs at least one identifier to identify the resource you want to debug.

Programmer response:  Provide one or more resources to be debugged. DTCN uses a combination of resource IDs to uniquely identify a resource. You should specify adequate resource qualification to ensure that you debug only the tasks you wish to debug.

EQA2512E  TCP/IP SOCKETS for CICS is not active
Explanation:  You have tried to set up a debug session using TCP/IP, but TCP/IP SOCKETS for CICS is not active in the CICS region.

Programmer response:  Either set up a non-TCP/IP session, or refer to the TCP/IP SOCKETS for CICS publications for guidance on activating it.

EQA2514I  Debug Tool profile saved
Explanation:  A profile was saved in the Debug Tool Profile Repository.

EQA2515I  Debug Tool profile replaced
Explanation:  Existing profile was updated in the Debug Tool Profile Repository.

EQA2516I  Debug Tool profile deleted
Explanation:  Existing profile was deleted from the Debug Tool Profile Repository.

EQA2517I  Profile not saved. Press PF4, or PF3 again to exit without saving.
Explanation:  PF3 has been pressed, but the new profile has not been saved in the repository.

Programmer response:  Press PF4 to save the profile in the repository, or press PF3 again to exit from DTCN without saving the new profile.

EQA2518I  Duplicate profile exists. Specify additional debug resources.
Explanation:  An attempt has been made to save a profile in the DTCN repository, but its debug resources match an existing profile.

Programmer response:  Provide additional resource IDs to qualify your debugging needs better.

EQA2519E  Site rules require that this field be filled in.
Explanation:  Your site has specified in its EQAOPTS member that this field must be filled in. For more information, refer to DTCNFORCExxxx options in macro EQAXOPT.

Programmer response:  Enter a resource name in the field.

EQA2600E  In order to SET MONITOR COLUMN OFF, you need to first SET MONITOR WRAP ON.
Explanation:  SET MONITOR COLUMN OFF was issued while SET MONITOR WRAP is OFF. The command is rejected, because the Monitor window already shows the values in one, scrollable line. The Monitor window must stay in columnar format.

Programmer response:  Change the setting of MONITOR WRAP to ON, and then issue SET MONITOR COLUMN OFF.

EQA2601E  In order to SET MONITOR WRAP OFF, you need to first SET MONITOR COLUMN ON.
Explanation:  SET MONITOR WRAP OFF was issued while SET MONITOR COLUMN is OFF. The command is rejected, because the Monitor window can show values in one scrollable line only when the setting of MONITOR COLUMN is ON.

Programmer response:  Change the setting of MONITOR COLUMN to ON, and then issue SET MONITOR WRAP OFF.

EQA2602E  Because SET MONITOR COLUMN is ON, the monitor window width must be at least 36 characters. The window size is not changed.
Explanation:  Because SET MONITOR COLUMN is ON, the monitor window width must be at least 36 characters. The window size is not changed.

Programmer response:  To change the Monitor window size, you need to first SET MONITOR COLUMN OFF.

EQA2603E  The width of the Monitor window is less than 36 characters. SET MONITOR COLUMN ON is not allowed.
Explanation:  Columnar format in the Monitor window can be displayed only if the width of Monitor window is bigger than 36 characters.

Programmer response:  To SET MONITOR COLUMN ON, you need first change the width of the Monitor window.
Chapter 8. Debug Tool messages
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA3045E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3046E</td>
<td>Illegal statement within include group</td>
</tr>
<tr>
<td>EQA3047E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3048E</td>
<td>PEND without matching procedure invocation</td>
</tr>
<tr>
<td>EQA3049E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3050E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3051E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3052E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3053E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3054E</td>
<td>Includes nested too deeply</td>
</tr>
<tr>
<td>EQA3055E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3056E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3057E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3058E</td>
<td>Symbolic name too long</td>
</tr>
<tr>
<td>EQA3059E</td>
<td>Multiple JOB cards</td>
</tr>
<tr>
<td>EQA3060E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3061E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3062E</td>
<td>Too many parms</td>
</tr>
<tr>
<td>EQA3063E</td>
<td>Parenthesis and apostrophe nesting exceeded</td>
</tr>
<tr>
<td>EQA3064E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3065E</td>
<td>If nesting exceeds limit</td>
</tr>
<tr>
<td>EQA3066E</td>
<td>ENDF without IF</td>
</tr>
<tr>
<td>EQA3067E</td>
<td>Keyword parameter or SET statement missing =</td>
</tr>
<tr>
<td>EQA3068E</td>
<td>JCLLIB missing ORDER=</td>
</tr>
<tr>
<td>EQA3069E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3070E</td>
<td>EXEC card has null first parameter</td>
</tr>
<tr>
<td>EQA3071E</td>
<td>EXEC card PGM= has null value</td>
</tr>
<tr>
<td>EQA3072E</td>
<td>DD concatenation without DD statement</td>
</tr>
<tr>
<td>EQA3073E</td>
<td>Missing JOB card</td>
</tr>
<tr>
<td>EQA3074E</td>
<td>DD after JOB not JOBLIB or JOBCAT or PROCLIB</td>
</tr>
<tr>
<td>EQA3075E</td>
<td>Multiple JOBCAT statements</td>
</tr>
<tr>
<td>EQA3076E</td>
<td>Multiple JOBLIB statements</td>
</tr>
<tr>
<td>EQA3077E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3078E</td>
<td>Illegal statement in procedure</td>
</tr>
<tr>
<td>EQA3079E</td>
<td>PROC nesting exceeded limit</td>
</tr>
<tr>
<td>EQA3080E</td>
<td>Query empty context stack</td>
</tr>
<tr>
<td>EQA3081E</td>
<td>PROC stack underflow</td>
</tr>
<tr>
<td>EQA3082I</td>
<td>Statement:</td>
</tr>
<tr>
<td>EQA3083I</td>
<td>Substitution:</td>
</tr>
<tr>
<td>EQA3084E</td>
<td>Too many steps in job</td>
</tr>
<tr>
<td>EQA3085E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3086E</td>
<td>StepName.DDName too long</td>
</tr>
<tr>
<td>EQA3087E</td>
<td>Could not find step named on override DD</td>
</tr>
<tr>
<td>EQA3088E</td>
<td>Could not add DD; no PGM= steps found</td>
</tr>
<tr>
<td>EQA3089I</td>
<td>Added to Proc:</td>
</tr>
<tr>
<td>EQA3090I</td>
<td>Override Proc:</td>
</tr>
<tr>
<td>EQA3091E</td>
<td>Fatal Error:</td>
</tr>
<tr>
<td>EQA3092E</td>
<td>Statement label is too long</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>EQA3093E</td>
<td>Statement parameter text is too long</td>
</tr>
<tr>
<td>EQA3094E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3095E</td>
<td>Instream PROC has no name</td>
</tr>
<tr>
<td>EQA3096E</td>
<td>Instream PROC PEND not found before end of file</td>
</tr>
<tr>
<td>EQA3097I</td>
<td>Input parameters:</td>
</tr>
<tr>
<td>EQA3098E</td>
<td>Internal error</td>
</tr>
<tr>
<td>EQA3099E</td>
<td>Internal error</td>
</tr>
<tr>
<td>EQA3100W</td>
<td>Could not substitute DB2 run unit for SYSTSIN</td>
</tr>
<tr>
<td>EQA3101W</td>
<td>Could not open</td>
</tr>
<tr>
<td>EQA3102W</td>
<td>Could not find OUTPUT spec named</td>
</tr>
<tr>
<td>EQA3103W</td>
<td>Could not open DD DDITV02 for DB2/IMS</td>
</tr>
<tr>
<td>EQA3104W</td>
<td>Could not read DD DDITV02</td>
</tr>
<tr>
<td>EQA3105W</td>
<td>Could not substitute DB2 run unit for DSNMVT01</td>
</tr>
<tr>
<td>EQA3106E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3107E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3108W</td>
<td>Could not find IMS program name</td>
</tr>
<tr>
<td>EQA3109W</td>
<td>Could not find IMS PSB name</td>
</tr>
<tr>
<td>EQA3110E</td>
<td>Internal error</td>
</tr>
<tr>
<td>EQA3111E</td>
<td>Override not proceeded by named DD stmt</td>
</tr>
<tr>
<td>EQA3112E</td>
<td>Open failure on DD:ATTRBOUT, specifies MA flat file</td>
</tr>
<tr>
<td>EQA3113I</td>
<td>Open failure on DD:RDPSTART, specifies logfile</td>
</tr>
</tbody>
</table>

**Chapter 8. Debug Tool messages** 331
**EQA9934W • EQA9945S**

**EQA9934W**
**Debug Tool EQA00CIC: Error loading Program.**
See **.SEQASAMP(EQA0CCDS)**

**Explanation:**
Debug Tool Program EQA00CIC in load module EQA00CISX was unable to load the specified program.

**Programmer response:**
Ensure that the group (EQA) that contains the Debug Tool run time routines is in the group list used during CICS start-up. If required, rerun the EQACCD job and restart the CICS region. Check the Debug Tool website for any applicable updates your system might require. If the problem persists, contact Debug Tool support.

---

**EQA9935E**
**XXXXXXX**

**Explanation:**
Debug Tool has experienced a problem, and is reporting diagnostic information (usually return codes given to Debug Tool by a subsystem.)

**Programmer response:**
Check the log for further diagnostic messages. If there is no obvious cause for the problem, contact IBM Support.

---

**EQA9936I**
**EQA00CIC Bad response from EXEC CICS cmd.**

**Explanation:**
Debug Tool has issued an EXEC CICS command and has received an unexpected response.

**Programmer response:**
Check the log for further diagnostic messages. If there is no obvious cause for the problem, contact IBM Support.

---

**EQA9937W**
**XPCFTCH MEA conflict- XXXXXXXX:**

**YYYYYYYYY**

**Explanation:**
Debug Tool is reporting that another CICS XPCFTCH global user exit has set a modified entry address (MEA) and prevented Debug Tool from any possible debugging of a specific non-Language Environment program. XXXXXXXX can be either: 'Prior MEA' or 'Program' or 'Transid' and YYYYYYYYY is the data associated with each. Note this message is only issued once when this occurs the first time after the Debug Tool CICS exits are activated. Subsequent conflicts are not written to the CICS JES log.

**Programmer response:**
Multiple XPCFTCH exits running in the same CICS region which can each set the MEA and return to CICS is usually not recommended. For Debug Tool, you will be unable to debug any Non-LE programs when the MEA was changed by another XPCFTCH exit. The behavior of Debug Tool in this kind of scenario will likely be unpredictable.

---

**EQA9938E**
**Error in deactivate of NewProg exits.**

**Explanation:**
Debug Tool detected an error in attempting to deactivate the NewProg supporting exits.

**Programmer response:**
An error has likely occurred during Debug Tool CICS region initialization. Ensure that Debug Tool is properly installed in the CICS region. Also verify that the Debug Tool hlq.SEQAMOD data set is in the region DFHRPL DD and the CICS resource definitions from hlq.SEQASAMP(EQA0CCSD) have been added.

---

**EQA9939I**
**IBM Debug Tool NewProg support deactivated.**

**Explanation:**
Debug Tool is reporting that NewProg support is now disabled in the current CICS region after a DTCP transaction was issued with the 'F' parameter. This support is to allow multi-region CICS configurations (for example, TOR/AOR), where DTCN is used, to work properly when DTCN is executed in one region (TOR) and tasks to be debugged are routed to an alternate region (AOR). This is only required in the regions where DTCN does not run.

---

**EQA9940I**
**IBM Debug Tool NonLE exits enabled.**

**Explanation:**
Debug Tool is reporting that the non-Language Environment-supporting CICS exits are now enabled in the current CICS region. This was accomplished by using PLTPI program EQAOCLPLT and starting with INITPARM=(EQAOCLPLT='NLE').

---

**EQA9941I**
**IBM Debug Tool NewProg support activated.**

**Explanation:**
Debug Tool is reporting that NewProg support is now enabled in the current CICS region. This support is to allow multi-region CICS configurations (for example, TOR/AOR), where DTCN is used, to work properly when DTCN is executed in one region (TOR) and tasks to be debugged might be routed to an alternate region (AOR). This is only required in the regions where DTCN does not execute.

---

**EQA9942I**
**IBM Debug Tool Screen stack exits enabled.**

**Explanation:**
Debug Tool is reporting that its single-terminal mode screen stacking exits are now enabled. This is to support installations where starting CICS exits is restricted by an external security manager (for example, RACF) and prevents Debug Tool from starting the exits when it starts a debug session for a user. This was accomplished using PLTPI program EQAOCLPLT and starting with INITPARM=(EQAOCLPLT='STK').

---

**EQA9943E**
**Error in activate of NonLE exits.**

**Explanation:**
Debug Tool detected an error while attempting to activate the non-Language Environment supporting exits.

**Programmer response:**
The error most likely occurred during Debug Tool CICS region initialization. Verify that the Debug Tool hlq.SEQAMOD data set is the in the region DFHRPL and the CICS resource definitions from hlq.SEQASAMP(EQA0CCSD) have been added.

---

**EQA9944E**
**Error in activate of NewProg exits.**

**Explanation:**
Debug Tool detected an error in attempting to activate the NewProg supporting exits.

**Programmer response:**
An error has likely occurred during Debug Tool CICS region initialization. Ensure that Language Environment is properly installed in the CICS region. Also verify that the Debug Tool hlq.SEQAMOD data set is in the region DFHRPL DD and the CICS resource definitions from hlq.SEQASAMP(EQA0CCSD) have been added.

---

**EQA9945S**
**Debug Tool DTRCB Unavailable.**

**Explanation:**
The Debug Tool non-Language Environment CICS global user exits were made active in a CICS region where Debug Tool did not successfully initialize during CICS region startup.

**Programmer response:**
Ensure that Language Environment is installed in the CICS region and verify that the Debug Tool installation steps were executed properly. For example, the Debug Tool hlq.SEQAMOD data set is in the DFHRPL DD and that the hlq.SEQASAMP(EQA0CCSD) job was run to add Debug Tool resource definitions to the CICS region.
EQA9946S  •  EQA9957E

**EQA9946S**  EQA01SVC TCBSVCA2 invalid - xxx where xxx=start, stop, term, strX, stopX

Explanation: Internal Debug Tool SVC Screening error or Debug Tool SVC (109 extended code 51) issued outside Debug Tool. The SVC will abend.

**Programmer response:** If using Debug Tool contact your IBM representative.

**EQA9947S**  EQA01SVC EQASVCSCREEN N/T create

Explanation: Internal Debug Tool SVC Screening error or Debug Tool SVC (109 extended code 51) issued outside Debug Tool. The SVC will abend.

**Programmer response:** If using Debug Tool contact your IBM representative.

**EQA9948S**  EQA01SVC No DTRCB at InitScreen

Explanation: Internal Debug Tool SVC Screening error or Debug Tool SVC (109 extended code 51) issued outside Debug Tool. The SVC will abend.

**Programmer response:** If using Debug Tool contact your IBM representative.

**EQA9949S or EQA9949I**  TCB SVC Screening already active and NOOVERRIDE is specified by EQAOPTS.

Explanation for Severe error: SVC Screening is in use by another product and SVC Screening CONFLICT=NOOVERRIDE is specified by EQAOPTS. Handling of non-LE events is not available. Debugging of non-LE programs will be restricted in this Debug Tool session.

Explanation for Informational message: The Debug Tool was invoked with an EQAOPTS options module that specified CONFLICT=NOOVERRIDE. Debug Tool will not intercept non-LE events and, therefore debugging of no-LE programs will be limited.

**Programmer response:** If you do not need to debug no-LE programs or to intercept non-LE events, no action is required. Otherwise, you must terminate the prior use of SVC SCREENING (TCBSVCS, TCBSVCSF, TCBSVCA2) before starting Debug Tool or have your installer provide an EQAOPTS that specified CONFLICT=OVERWRITE. CONFLICT=OVERWRITE allows Debug Tool to save and restore the previous use of SVC SCREENING (TCBSVCS, TCBSVCSF, TCBSVCA2).

**EQA9950E**  Error enabling XEIN screen exit.

Explanation: Debug Tool detected an error during the ENABLE of a required CICS exit program.

**Programmer response:** Determine if the Debug Tool hlq.SEQASMP (EQACC5D) library is available in the DFHRPL concatenation of the CICS region and the resource definitions provided in hlq.SEQASMP (EQACC5D) have been added to the CICS region that is initializing.

**EQA9951E**  Error enabling XEIOUT screen exit.

Explanation: Debug Tool detected an error during the ENABLE of a required CICS exit program.

**Programmer response:** Determine if the Debug Tool hlq.SEQASMP library is available in the DFHRPL concatenation of the CICS region and the resource definitions provided in hlq.SEQASMP (EQACC5D) have been added to the CICS region that is initializing.

**EQA9952E**  Error in locate of Debug Tool RCB.

Explanation: Debug Tool CICS PLT program EQAOCLT detected an error during the search for Debug Tool region-level resources.

**Programmer response:** An error has likely occurred during Debug Tool CICS region initialization. Ensure that Language Environment is properly installed in the CICS region. Also, verify that the Debug Tool hlq.SEQASMP data set is in the region DFHRPL DD and the CICS resource definitions from hlq.SEQASMP (EQACC5D) have been added.

**EQA9953E**  NOTAUTH Error issuing CICS EXTRACT EXIT.

Explanation: Debug Tool detected a NOTAUTH condition during an EXTRACT EXIT call to CICS.

**Programmer response:** Determine if the current Debug Tool user has external security-manager (RACF) access to the EXITPROGRAM CICS CLASS. This includes the ability to issue the EXEC CICS EXTRACT/ENABLE/DISABLE EXIT commands. If this is not permitted, then use of Debug Tool PLT initialization routine, EQAOCLT, is recommended. Refer to the **Debug Tool Customization Guide** for information on EQAOCLT.

**EQA9954E**  Invalid EXIT ENABLE request.

Explanation: Debug Tool CICS PLT program EQAOCLT detected an error during the ENABLE of a required CICS exit program.

**Programmer response:** An INVREQ response was received during a call to CICS to ENABLE the Debug Tool screen-stack exits. Contact IBM Support Center and report the error.

**EQA9955E**  User not authorized for EXIT ENABLE.

Explanation: Debug Tool CICS PLT program EQAOCLT detected an error during the ENABLE of a required CICS exit program.

**Programmer response:** Determine if the CICS region user id has external security-manager (RACF) access to the EXITPROGRAM CICS CLASS. This includes the ability to issue the EXEC CICS EXTRACT/ENABLE/DISABLE EXIT commands.

**EQA9956E**  Invalid program name on EXIT ENABLE.

Explanation: Debug Tool CICS PLT program EQAOCLT detected an error during the ENABLE of a required CICS exit program.

**Programmer response:** Determine if the Debug Tool hlq.SEQASMP library is available in the DFHRPL concatenation of the CICS region and the resource definitions provided in hlq.SEQASMP (EQACC5D) have been added to the CICS region that is initializing.

**EQA9957E**  Invalid CICS release. Latest CICS used.

Explanation: Debug Tool detected an unsupported release of CICS and defaults to the latest release of CICS that this version of Debug Tool supports.

**Programmer response:** Determine if Debug Tool is starting on a supported release of CICS. See the Debug Tool Program Directory for the list of CICS releases that are supported.
**EQA9958I • EQA9968E**

EQA9958I  IBM Debug Tool CICS PLT init start.

**Explanation:** Debug Tool program EQA0CPLT is starting. This program activates various Debug Tool resources during CICS region startup. This includes starting up Debug Tool support for running in CICS multi-region configurations (INITPARM=(EQA0CPLT='NWP')) and starting Debug Tool screen stack exits once at region initialization (INITPARM=(EQA0CPLT='STK')) and starting Debug Tool non-Language Environment-supporting exits (INITPARM=(EQA0CPLT='NLE')). Combinations of these selections are also supported. For example:

```
INITPARM=(EQA0CPLT='NWP,STK,NLE')
```

---

EQA9959I  IBM Debug Tool CICS PLT init end.

**Explanation:** Debug Tool program EQA0CPLT is ending. This program activates various Debug Tool resources during CICS region startup. This includes starting up Debug Tool support for running in CICS multi-region configurations (INITPARM=(EQA0CPLT='NWP')) and starting Debug Tool screen stack exits once at region initialization (INITPARM=(EQA0CPLT='STK')) and starting Debug Tool non-Language Environment-supporting exits (INITPARM=(EQA0CPLT='NLE')).

---

EQA9960I  Program abend: Abcode Prog: Abprogram Ret@:

**Explanation:** Debug Tool has detected abend Abcode while processing program Abprogram under CICS. Ret@ is the address of the location where the abend was issued.

**Programmer response:** This message occurs when the non-Language Environment Debug Tool CICS exits are active and an abend has occurred in the application currently being debugged. It is trapped as a result of the TRAP(ON) runtime option. The default behavior for the STEP or GO command at this time is for Debug Tool to display the abend and allow the task to terminate or allow any active CICS HANDLE abend routines to run or, if applicable, allow any Language Environment user handlers or signal catchers to run. Use the TRAP(OFF) runtime option if you do not want Debug Tool to capture abends. This message is written to the CICS region's JES message log.

---

EQA9961I  Program interrupt: Intcl Prog: Abprogram Int@:

**Explanation:** Debug Tool has detected program check interrupt code Intcl while processing program Abprogram under CICS. Int@ is the address of the location where the program check occurred.

**Programmer response:** This message occurs when the non-Language Environment Debug Tool CICS exits are active and a program check has occurred in the application currently being debugged. It is trapped as a result of the TRAP(ON) runtime option. The default behavior for the STEP or GO command at this time is for Debug Tool to display the abend and allow the task to terminate or allow any active CICS HANDLE abend routines to run or, if applicable, allow any Language Environment user handlers or signal catchers to run. Use the TRAP(OFF) runtime option if you do not want Debug Tool to capture program checks. This message is written to the CICS region's JES message log.

---

EQA9962I  IBM Debug Tool Exit Activation PLT start.

**Explanation:** Debug Tool program EQANCPLT is starting. This program activates the Debug Tool non-Language Environment CICS global exits which must be executed as either a stage 2 or 3 PLT post initialization program.

---

EQA9963I  IBM Debug Tool Exit Activation PLT end.

**Explanation:** Debug Tool program EQANCPLT is ending. This program activates the Debug Tool non-Language Environment CICS global exits which must be executed as either a stage 2 or 3 PLT post initialization program.

---

EQA9964E  Create EQADTA name/token error. RC: RC

**Explanation:** Debug Tool is unable to initialize for a non-Language Environment assembler program under CICS.

**Programmer response:** Contact IBM support center and report the error. If this message occurs repeatedly, disable the non-Language Environment CICS exits using transaction DTCX (DTCXXF) or by removing the EQANCPLT from the CICS PLT.

---

EQA9965E  CEEDBGEVNETXT Error. RC: RC

**Explanation:** Debug Tool is unable to initialize for a non-Language Environment assembler program under CICS.

**Programmer response:** Contact IBM support center and report the error. If this message occurs repeatedly, disable the non-Language Environment CICS exits using transaction DTCX (DTCXXF) or by removing the EQANCPLT from the CICS PLT.

---

EQA9966E  Back-level Debug Tool SVC detected. V5R1 or later SVCs required for Non-LE support.

**Explanation:** Debug Tool is unable to initialize for a non-Language Environment assembler program under CICS due to back-level Dynamic Debug facility SVCs.

**Programmer response:** Verify that the latest version of the Debug Tool Dynamic Debug facility SVCs are installed. The level of the SVCs can be checked by running the exec in dthlnq.SEQAEXEC(EQADTSVC).

---

EQA9967I  EQA00SVC Level:Epa00svcVersion EQA01SVC Level:Eqa01svcVersion

**Explanation:** Debug Tool is unable to initialize for a non-Language Environment assembler program under CICS due to back-level Dynamic Debug facility SVCs. This message occurs with message EQA9966E and indicates the detected levels of the two Debug Tool Dynamic Debug facility SVCs.

**Programmer response:** Verify that the latest version of the Debug Tool Dynamic Debug facility SVCs are installed. The level of the SVCs can be checked by running the exec in dthlnq.SEQAEXEC(EQADTSVC). For non-Language Environment support, EQA00SVC must be at least 04 and EQA01SVC at least 05.

---

EQA9968E  Invalid Exit Type ...

**Explanation:** Debug Tool CICS exit activation transaction DTCX is unable to determine a valid exit type to start or stop.

**Programmer response:** Verify that DTCX is issued with an exit type of X=all exits or F=XPCTFCH exit or E=E=XEIN or A=XPCTA or H=XPCHAIR. Note there is no blank space between DTCX and this parameter (for example: DTCXXO = turn all exits ON and DTCXXF = turn all exits OFF).
**EQA9969E**  
**EXPLANATION:** Debug Tool CICS exit activation transaction DTCX or Newprog activation transaction DTCP is unable to determine a valid action to take. O=On or F=Off.

**PROGRAMMER RESPONSE:** Re-enter the transaction with an O or F parameter where O=On and F=Off.

---

**EQA9970I**  
**EXPLANATION:** Debug Tool CICS global user exits activated successfully.

---

**EQA9971I**  
**EXPLANATION:** Debug Tool CICS global user exits deactivated successfully.

---

**EQA9972I**  
**EXPLANATION:** Debug Tool CICS exit activation transaction DTCX successfully started the glueexitname exit where glueexitname is either XPCFTCH, XEIIIN, XEIOUT, XPCTA, or XPCHAIR.

---

**EQA9973I**  
**EXPLANATION:** Debug Tool CICS exit activation transaction DTCX successfully stopped the glueexitname exit where glueexitname is either XPCFTCH, XEIIIN, XEIOUT, XPCTA, or XPCHAIR.

---

**EQA9974E**  
**EXPLANATION:** Debug Tool CICS exit activation transaction DTCX was unable to activate glueexitname - dtexitname where glueexitname is either XPCFTCH, XEIIIN, XEIOUT, XPCTA, or XPCHAIR and dtexitname is either EQANCFTC, EQANCXEI, EQANCXAB, or EQANCXHA.

**PROGRAMMER RESPONSE:** Verify that the latest dthlg.SEQASAMP(EQACSD) CICS resource definitions are installed in the Debug Tool V5.1 dthlg.SEQMOB library is in the CICS DFRPL DD concatenation. If this has already been done, contact IBM support center and report the error.

---

**EQA9975E**  
**EXPLANATION:** Debug Tool CICS exit activation transaction DTCX was unable to deactivate glueexitname - dtexitname where glueexitname is either XPCFTCH, XEIIIN, XEIOUT, XPCTA, or XPCHAIR and dtexitname is either EQANCFTC, EQANCXEI, EQANCXAB, or EQANCXHA.

**PROGRAMMER RESPONSE:** Verify that the latest dthlg.SEQASAMP(EQACSD) CICS resource definitions are installed in the Debug Tool V5.1 dthlg.SEQMOB library is in the CICS DFRPL DD concatenation. If this has already been done, contact IBM support center and report the error.

---

**EQA9976I**  
**EXPLANATION:** The requested CICS global user exit for non-Language Environment assembler support was already active. glueexitname is either XPCFTCH, XEIIIN, XEIOUT, XPCTA, or XPCHAIR.

---

**EQA9977E**  
**EXPLANATION:** dtexitname is backlevel. Exits not enabled.

**PROGRAMMER RESPONSE:** Verify that the latest version of the Debug Tool Dynamic Debug facility SVCs are installed. The level of the SVCs can be checked by running the exec in dthlg.SEQASAMP(EQADTSVC). For non-Language Environment support, EQA00SVC must be at least 04 and EQA01SVC at least 05.

---

**EQA9978I**  
**EXPLANATION:** Debug Tool is unable to set a hook and stop in this program because the separate debug file cannot be located. This program was specified using a DTCN or CADP profile.

**PROGRAMMER RESPONSE:** Verify that the debug data file exists and make its location known to Debug Tool by using the SET DEFAULT LISTINGS or SET SOURCE command, the EQAUSEDAT user exit or the EQADEBUG DD name.

---

**EQA9979I**  
**EXPLANATION:** Debug Tool is unable to load the user program UserProgram.

**PROGRAMMER RESPONSE:** Ensure that the specified program name is spelled correctly and that the program is available in the standard search path for load modules.

---

**EQA9980E**  
**EXPLANATION:** error_description is replaced with the following text:
- Debug Tool unrecoverable CICS task error.
- CICS abend code is aaaa
- Debug Tool session ended.

(Where aaaa is the CICS abend code.)

**PROGRAMMER RESPONSE:** Look up the CICS abend code in the CICS Messages and Codes manual and take the appropriate action to resolve the CICS abend.

---

**EQA9981I**  
**EXPLANATION:** EQAx0STO Internal Error WTO Num

**PROGRAMMER RESPONSE:** Ensure that your program is not overwriting Debug Tool storage. Check the Debug Tool web site for any applicable service updates your system might require. If the problem persists, contact Debug Tool support.

---

**EQA9982E**  
**EXPLANATION:** A non-zero response code was returned from EXEC CICS 'command'. Resp value = EIBRESP

**PROGRAMMER RESPONSE:** Review the command and response to determine if CICS configuration needs to be changed. If there is no obvious cause for the error condition, contact IBM Support.
**EQA9983I • EQA9995E**

**EQA9983I  Invalid keyword value: value**

Explanation: An invalid value was specified for the indicated runtime parameter.

Programmer response: Correct the specified value.

**EQA9984I  No user program name was specified.**

Explanation: EQANMD8G was invoked without a positional parameter specifying the name of the program to be debugged.

Programmer response: Specify an initial positional parameter indicating the name of the program to be debugged.

**EQA9985I  Dynamic Debug is required for non-LE Debug Tool.**

Explanation: The non-Language Environment version of Debug Tool (EQA9988) was invoked but the Debug Tool Dynamic Debug facility SVCs required for dynamic debug support have not been installed.

Programmer response: Have your system programmer complete the installation of the required Debug Tool Dynamic Debug facility SVCs.

**EQA9986E  Error in CEE1V006 loading OSI**

Explanation: One of the required Debug Tool load modules is missing.

Programmer response: Contact your system programmer to verify the proper installation of Debug Tool.

**EQA9987I  First parameter to load_module is not addressable. Cannot debug this module.**

Explanation: A non-Language Environment program issued a **LINK SVC** for the specified load module. However, an invalid address was specified for the first positional parameter that is used to specify Language Environment runtime parameters and user parameters. This parameter must be valid, so that Debug Tool can add the TEST parameter.

Programmer response: Correct the parameter address passed to the **LINK SVC**.

**EQA9988S  Debug Tool has terminated the enclave.**

Explanation: A Debug Tool **QUIT** command was issued in a multi-enclave environment.

Programmer response: No action is necessary.

**EQA9989I  EQANMD8G requires Debug Tool V5R1 or later SVC.**

Explanation: The Debug Tool V5R1 (or later) non-Language Environment program was invoked but the installed version of the Debug Tool Dynamic Debug facility SVCs were from an earlier version of Debug Tool.

Programmer response: Have your system programmer install the current V5R1 or later Debug Tool Dynamic Debug facility SVCs.

**EQA9990I  LOAD detect.**

Programmer response: This is an internal Debug Tool message. No user response is required.

**EQA9991E  Error loading load_module**

Explanation: An error was encountered loading the specified load module.

Programmer response: Retry the Debug Tool session. Check the Debug Tool website for any applicable service updates your system might require. If the problem persists, contact Debug Tool support.

**EQA9992E  Internal error processing Language Environment service FunctionCode**

Explanation: An internal Debug Tool error has occurred.

Programmer response: Check the Debug Tool web site for any applicable service updates your system might require. If the problem persists, contact Debug Tool support.

**EQA9993I  EQA00OHT: Failed to set hook in R/O storage**

Explanation: The Dynamic Debug facility was unable to successfully use the Authorized Debug facility to place a hook into an application that has been loaded into protected (read only) storage.

Debug Tool Action: If the application has been compiled with hooks (eg **TEST(ALL,...)**) then you will be able to debug this application. If the application has been compiled with **TEST(NONE,...)** then you will not be able to **STEP** or set breakpoints.

Programmer response: Ensure that both the Dynamic Debug facility and Authorized Debug facility have been activated. Ensure that you have the access through your security system to resource **EQADTOOL.AUTHDEBUG** in **CLASS(FACILITY)**.

**EQA9994E  No storage for DT RCB**

Explanation: There is insufficient storage for Debug Tool to initialize.

Programmer response: Increase the region size available to the program and rerun.

**EQA9995E  REQUIRED TEXT**

Explanation: All EQA9995E messages signify a severe error has occurred in the Dynamic Debug facility SVC routine while processing an 0A9I instruction.

Programmer response:
1. Make sure none of the applications you are debugging issue the reserved 0A9I (SVC 145) instruction.
2. If you have non-IBM products installed on your system, make sure none of them issue the reserved 0A9I (SVC 145) instruction.
3. Try running the Dynamic Debug facility IVP (Installation Verification Program). This program can be found in member EQAWIVP of data set EQAW.SEQASAMP.
4. Have your system support person re-install the Dynamic Debug facility SVC using member EQAWISVC of data set EQAW.SEQASAMP and then run the IVP (see step [3].
5. Check the Debug Tool web site for any applicable service updates your system might require.
6. If the problem persists, report the error message text, return code, and reason code to your IBM representative.

---

**EQA9996E**  
**ERROR DESCRIPTION**

Explanation:  
A severe error has occurred in the Debug Tool Authorized Debug Facility SVC routine EQA01SVC. EQA01SVC is SVC 109 with extended function code 51.

Programmer response:  
Check the Debug Tool web site for any applicable service updates your system might require. If the problem persists, report the error message text, return code, and reason code to your IBM representative.

---

**EQA9997E**  
**ERROR DESCRIPTION**

Explanation:  
The ASMADOP module could not be loaded. Debugging via a Disassembly View cannot be supported.

Programmer response:  
1. Have your system support person verify that the ASMADOP module is available (in the search path).
2. Check the Debug Tool web site for any applicable service updates your system might require.
3. If the problem persists, report the error message text to your IBM representative.

---

**EQA9998I**  
**Debug Tool DTCN profile skipped.**

Explanation:  
The Debug Tool profile has been skipped because a more qualified profile has been found or an older, equally qualified, profile has been found.

Programmer response:  
Additional EQA9998I messages will follow. See the details of these additional messages for the appropriate response.

---

**EQA9999E**  
**ERROR DESCRIPTION**

Explanation:  
Severe Internal Error in Debug Tool Module  
Please contact your IBM Representative  
Failure address - xxxxxxx  
Program Check at module+offset

Programmer response:  
See details of message issued for appropriate response or indication of potential problem. Check the Debug Tool web site for any applicable service updates your system might require. If the problem persists, contact Debug Tool support.

---

**EQA9999W**  
**ERROR DESCRIPTION**

Explanation:  
Warning Message issued by Debug Tool Module

Programmer response:  
See details of message issued for appropriate response or indication of potential problem. Check the Debug Tool web site for any applicable service updates your system might require.
Chapter 9. Non-Language Environment IMS messages

Each message has a number of the form EQAInnnx, where EQAI indicates that the message is non-Language Environment IMS message, nnnn is the number of the message, and x indicates the severity level of each message. The variable x can be any of the following values:

I  An informational message calls attention to some aspect of a command response that might assist the programmer.

W  A warning message calls attention to a situation that might not be what is expected or to a situation that Debug Tool attempted to fix.

E  An error message describes an error that Debug Tool detected or cannot fix.

S  A severe error message describes an error that indicates a command referring to bad data, control blocks, program structure, or something similar.

U  An unrecoverable error message describes an error that prevents Debug Tool from continuing.

Symbols in messages

Many of the Debug Tool messages contain information that is inserted by the system when the message is issued. In this publication, such inserted information is indicated by italicized symbols, as in the following:

EQA1046I The breakpoint-id breakpoint is replaced.

The portion of Debug Tool located on the host notifies you of errors associated with debugging functions carried out by the host.

Refer to the following sections for more information related to the material discussed in this section.

Related tasks

---

EQAI1002S  CSVQUERY failure

**Explanation:** Query application MPP failed.

**User response:** Consult your system programmer.

---

EQAI1003S  Environment AIB call failed

**Explanation:** Query environment AIB call failed.

**User response:** Consult your system programmer.

---

EQAI1004S  Environment IOPCB call failed

**Explanation:** Query environment IOPCB call failed.

**User response:** Consult your system programmer.

---

EQAI1008S  Error loading EQANIDBG

---

EQAI1009S  Invalid EQASET specification

**Explanation:** Load Debug Tool non-Language Environment front end failed.

**User response:** Consult your system programmer.

---

EQAI1013S  Retrieve of token failed with rc returncode

**Explanation:** Fail to retrieve token (EQAuser_ID). The return code is returncode.

**User response:** Run the EQASET transaction from the terminal where the application transaction is invoked.
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Explanation</th>
<th>User response</th>
<th>User response</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAI1020S</td>
<td>Retrieve <em>token</em> failed</td>
<td>Fail to retrieve <em>token</em>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>User response:</strong> Start EQASET transaction with a valid keyword (MFI, TCP, VTCP, VTAM).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQAI2005I</td>
<td>DEBUG SET ON FOR MFI SETTING</td>
<td>Debugging is turned on and setting is MFI.</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td><strong>User response:</strong> None.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQAI2006I</td>
<td>DEBUG SET ON FOR TCP SETTING</td>
<td>Debugging is turned on and setting is TCP.</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td><strong>User response:</strong> None.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQAI2007I</td>
<td>DEBUG SET ON FOR VTCP SETTING</td>
<td>Debugging is turned on and setting is VTCP.</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td><strong>User response:</strong> None.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQAI2008I</td>
<td>DEBUG SET OFF</td>
<td>Debugging is turned off.</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td><strong>User response:</strong> None.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQAI2009I</td>
<td>DEBUG SET ON FOR SPECIFIED MFI LU</td>
<td>Debugging is turned on and setting is the specified MFI LU name.</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td><strong>User response:</strong> None.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQAI2010I</td>
<td>DEBUG SET ON FOR SPECIFIED TCP IP</td>
<td>Debugging is turned on and setting is the specified MFI LU name.</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td><strong>User response:</strong> None.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQAI2011I</td>
<td>DEBUG SET ON FOR SPECIFIED VTCP IP</td>
<td>Debugging is turned on and setting is the specified VTCP IP address.</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td><strong>User response:</strong> None.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQAI2012I</td>
<td>VALID KEYWORDS: ON, OFF, MFI=, TCP=, VTCP, VTAM=, STATUS</td>
<td>Valid keywords for EQASET transaction.</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td><strong>User response:</strong> None.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQAI2013E</td>
<td>USERID FROM IMS ENVIRONMENT IS BLANK</td>
<td>User ID is blank or cannot be found.</td>
<td></td>
<td>Contact your system programmer.</td>
</tr>
<tr>
<td></td>
<td><strong>User response:</strong> Contact your system programmer.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQAI2014E</td>
<td>NO DATA RECEIVED</td>
<td>EQASET transaction is entered without keyword.</td>
<td></td>
<td>Reference documentation for EQASET definition and usage.</td>
</tr>
<tr>
<td></td>
<td><strong>User response:</strong> Reference documentation for EQASET definition and usage.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQAI2015E</td>
<td>NAME/TOKEN SAVE FAILED RC= returnCode</td>
<td>Fail to save setting value.</td>
<td></td>
<td>Contact your system programmer.</td>
</tr>
<tr>
<td></td>
<td><strong>User response:</strong> Contact your system programmer.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQAI2016E</td>
<td>MFI/TCP/VTCP VALUE MUST BE SET TO USE KEYWORD ON</td>
<td>No setting exists when debugging is turned on with EQASET ON.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>User response:</strong> Set value of one of the settings (MFI, TCP, VTCP) with EQASET transaction.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQAI2017I</td>
<td>NO STATUS</td>
<td>No on/off switch exists.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>User response:</strong> Set value of one of the settings (MFI, TCP, VTCP) with EQASET transaction.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQAI2018I</td>
<td>DEBUG SET ON FOR SPECIFIED VTAM LU</td>
<td>Debugging is turned on and setting is the specified VTAM LU associated with user id.</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td><strong>User response:</strong> None.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQAI2019I</td>
<td>DEBUG SET ON FOR VTAM SETTING</td>
<td>Debugging is turned on and setting is VTAM.</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td><strong>User response:</strong> None.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 10. Load Module Analyzer Messages

All messages are shown in this section are in mixed case English. The uppercase English message text is the same, but is in uppercase letters.

Each message has a number of the form EQALMnnnx, where EQA indicates that the message is an Load Module Analyzer message, nnn is the number of the message, and x indicates the severity level of each message. The variable x can be any of the following values:

I  An informational message calls attention to some aspect of a command response that might assist the programmer.

W  A warning message calls attention to a situation that might not be what is expected or to a situation that Debug Tool attempted to fix.

E  An error message describes an error that Load Module Analyzer detected or cannot fix.

S  A severe error message describes an error that indicates a command referring to bad data, control blocks, program structure, or something similar.

U  An unrecoverable error message describes an error that prevents Load Module Analyzer from continuing.

Symbols in messages

Many of the Load Module Analyzer messages contain information that is inserted by the system when the message is issued. In this publication, such inserted information is indicated by italicized symbols, as in the following:

EQA1046I The breakpoint-id breakpoint is replaced.

The portion of Load Module Analyzer located on the host notifies you of errors associated with debugging functions carried out by the host.

Refer to the following sections for more information related to the material discussed in this section.

Related tasks

<table>
<thead>
<tr>
<th>Message Number</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQALM000S</td>
<td>**** UNKNOWN ERROR *****</td>
<td>An unexpected or unrecognized error has occurred.</td>
</tr>
<tr>
<td>EQALM001E</td>
<td>**** Unable to open filename *****</td>
<td>This indicated file cannot be opened.</td>
</tr>
<tr>
<td>EQALM002I</td>
<td>**** Unable to open EQAIN. All members will be processed. *****</td>
<td>The EQAIN file was not allocated. All members of the PDS(E) concatenation allocated to EQALIB will be processed.</td>
</tr>
<tr>
<td>EQALM003E</td>
<td>**** Unknown member specified in SELECT statement ****</td>
<td>The member specified on the SELECT statement was not found in the EQALIB concatenation.</td>
</tr>
<tr>
<td>EQALM004E</td>
<td>**** Unrecognized control statement ****</td>
<td>An unrecognized control statement was encountered while processing the EQAIN file.</td>
</tr>
<tr>
<td>EQALM005E</td>
<td>**** Work area overflow *****</td>
<td>An internal work area has overflowed.</td>
</tr>
</tbody>
</table>
EQALM006E **** Error rc-reason returned from Binder API ****

Explanation: The indicated return and reason codes were returned from the Binder interface module. This message may be accompanied by EQALM999W messages written to the Job Log indicating the associated IEBBIND return and reason codes.

EQALM007W **** text is an unrecognized option ****

Explanation: The specified text is not a supported option.

EQALM008S **** Debug Tool Utilities and Advanced Functions is required to use this function. ****

Explanation: A valid license for this program could not be found on the current system.

EQALM009S **** Unable to load EQALMER2. ****

Explanation: The indicated load module could not be found in the current STEPLIB, system link-list, etc. This program is part of the SEQMOD data set shipped with the Japanese feature of Debug Tool Utilities and Advanced Functions.

EQALM010E **** Unable to obtain list of EQALIB members. RC returned from DESERV macro. ****

Explanation: The indicated return code was generated by the DEServ function. Refer to the appropriate Data Facility Product publication for a description of the error.

EQALM011E **** Return code/reason code from IEBBIND Function: rc/reason. ****

Explanation: The specified return code was received from the Binder API's.

EQALM012S **** Unable to obtain storage ****

Explanation: Sufficient storage was not available for processing.

Programmer response: Increase the region size and rerun the application.

EQALM999W error description

Explanation: This message is issued to the Job Log via a Write To Operator (WTO). It contains information about errors returned by IEBBIND. See the appropriate Binder documentation for a description of the indicated return code and reason code.
Chapter 11. Debug Tool Language Environment user exit messages

Each message has a number of the form EQAU<nnn>, where EQAU indicates that the message is a Debug Tool Language Environment user exit message, <nnn> is the number of the message, and x indicates the severity level of each message.

There are three categories of Debug Tool Language Environment user exit messages:

- "DB2 user exit messages" begin with EQAU0
- "IMS user exit messages" on page 345 begin with EQUA1
- "Batch user exit messages" on page 346 begin with EQUA2

The variable x can be any of the following values:

- I  An informational message calls attention to some aspect of a command response that might assist the programmer.
- W  A warning message calls attention to a situation that might not be what is expected or to a situation that Debug Tool attempted to fix.
- E  An error message describes an error that Debug Tool detected or cannot fix.
- S  A severe error message describes an error that indicates a command referring to bad data, control blocks, program structure, or something similar.
- U  An unrecoverable error message describes an error that prevents Debug Tool from continuing.

Symbols in messages

Many of the Debug Tool messages contain information that is inserted by the system when the message is issued. In this publication, such inserted information is indicated by italicized symbols, as in the following:

EQAU046I The breakpoint-id breakpoint is replaced.

The portion of Debug Tool located on the host notifies you of errors associated with debugging functions carried out by the host.

**DB2 user exit messages**

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAU001E</td>
<td>CSVQuery call failed.</td>
</tr>
<tr>
<td>Explanation: CSVQuery failed to return a valid program stored external name. EQADDCXT returns without a TEST run-time option.</td>
<td></td>
</tr>
<tr>
<td>User response: Consult your system programmer.</td>
<td></td>
</tr>
<tr>
<td>EQAU002E</td>
<td>DSNWLI call failed. Return code is: returnCode.</td>
</tr>
<tr>
<td>Explanation: DSNWLI call failed. EQADDCXT returns</td>
<td></td>
</tr>
<tr>
<td>User response: Consult your system programmer.</td>
<td></td>
</tr>
<tr>
<td>EQAU003E</td>
<td>DSNWLI call failed.</td>
</tr>
<tr>
<td>Explanation: DSNWLI failed to return a valid authorization ID. EQADDCXT returns without a TEST run-time option.</td>
<td></td>
</tr>
<tr>
<td>User response: Consult your system programmer.</td>
<td></td>
</tr>
</tbody>
</table>
EQAU004W  Data set allocation failed. Data set name is: dname.

Explanation:  Data set allocation action failed. EQADDCXT returns without a TEST run-time option.

User response:  Use DTU option: Manage TEST run-time option to create a data set.

---

EQAU005W  Data set open failed. Data set name is: dname.

Explanation:  Data set open action failed. EQADDCXT returns without a TEST run-time option.

User response:  Consult your system programmer.

---

EQAU006W  Invalid program name found in data set -- pgmname.

Explanation:  A program name of more than eight characters is defined in the data set. EQADDCXT returns without a TEST run-time option.

User response:  Use DTU option: Manage TEST run-time option to edit the name on the name list.

---

EQAU007W  Data set is empty.

Explanation:  The data set is empty. EQADDCXT returns without a TEST run-time option.

User response:  Use DTU option: Manage TEST run-time option to add contents to the data set.

---

EQAU008W  No TEST run-time option in data set.

Explanation:  The data set does not contain TEST run-time option. EQADDCXT returns without a TEST run-time option.

User response:  Use DTU option: Manage TEST run-time option to add TEST run-time option to the data set.

---

EQAU009E  Data set get failed. Data set name is: dname.

Explanation:  Data set get action failed. EQADDCXT returns without a TEST run-time option.

User response:  Consult your system programmer.

---

EQAU010W  Data set name is not correct. Data set name is: dsnname.

Explanation:  Data set name is not correct. EQADDCXT returns without a TEST run-time option.

User response:  Verify and correct data set naming pattern in EQADDCXT. Assemble and re-link it into your application.

---

EQAU011W  Data set allocation failed. DD name is not valid.

Explanation:  DD name is not invalid. EQADDCXT returns without a TEST run-time option.

User response:  Verify and correct data set name pattern in EQADDCXT. Assemble and re-link it into your application. If problem persists, consult your system programmer.

---

EQAU012W  Data set is not physical sequential. Data set name is: dname.

Explanation:  Data set is not a sequential data set. EQADDCXT returns without a TEST run-time option.

User response:  Use DTU option: Manage TEST run-time option to create a sequential data set.

---

EQAU013W  First record type is not <PGM> in data set. Record is: dname.

Explanation:  The type of the first record is not <PGM>. A Program list record must be the first record in the data set. EQADDCXT returns without a TEST run-time option.

User response:  Use DTU option: Manage TEST run-time option to create a sequential data set with proper content.

---

EQAU014W  Record type is not <TST> or <RTO> in data set. Record is: dname.

Explanation:  The type of the record following <PGM> record is not <TST> TEST run-time option or <RTO> other Language Environment run-time option. EQADDCXT returns without a TEST run-time option.

User response:  Use DTU option: Manage TEST run-time option to create a sequential data set with proper content.

---

EQAU015W  Record type is out of sequence in data set. Record is: dname. User response: Use DTU option: Manage TEST run-time option to create a sequential data set with proper content.

Explanation:  The <TST> record is found after the <RTO> record. All <TST> records must be grouped together and placed before first <RTO> record. EQADDCXT returns without a TEST run-time option.

User response:  Use DTU option: Manage TEST run-time option to create a sequential data set with proper content.
IMS user exit messages

EQAU101E CSVQuery call failed.
Explanation: CSVQuery failed to return a valid program name. EQADICXT returns without a TEST run-time option.
User response: Consult your system programmer.

EQAU102E AIBTDLI call failed. Return code is: retcode.
Explanation: AIBTDLI call failed. EQADICXT returns without a TEST run-time option.
User response: Consult your system programmer.

EQAU104W Data set allocation failed. Data set name is: dname.
Explanation: Data set allocation action failed. EQADICXT returns without a TEST run-time option.
User response: Use DTU option: Manage TEST run-time option to create a data set.

EQAU105W Data set open failed. Data set name is: dname.
Explanation: Data set open action failed. EQADICXT returns without a TEST run-time option.
User response: Consult your system programmer.

EQAU106W Invalid program name found in data set - pgmname.
Explanation: A program name of more than eight characters is defined in the data set. EQADICXT returns without a TEST run-time option.
User response: Use DTU option: Manage TEST run-time option to edit the name on the name list.

EQAU107W Data set is empty.
Explanation: The data set is empty. EQADICXT returns without a TEST run-time option.
User response: Use DTU option: Manage TEST run-time option to add contents to the data set.

EQAU108W No TEST run-time option in data set.
Explanation: The data set does not contain TEST run-time option. EQADICXT returns without a TEST run-time option.
User response: Use DTU option: Manage TEST run-time option to add TEST run-time option to the data set.

EQAU109E Data set get failed. Data set name is: dname.
Explanation: Data set get action failed. EQADICXT returns without a TEST run-time option.
User response: Consult your system programmer.

EQAU110W Data set name is not correct. Data set name is: dname.
Explanation: Data set name is not correct. EQADICXT returns without a TEST run-time option.
User response: Verify and correct data set naming pattern in EQADICXT. Assemble and re-link it into your application.

EQAU111W Data set allocation failed. DD name is not valid.
Explanation: DD name is not invalid. EQADICXT returns without a TEST run-time option.
User response: Verify and correct data set name in EQADICXT. Assemble and re-link it into your application. If problem persists, consult your system programmer.

EQAU112W Data set is not physical sequential. Data set name is: dname.
Explanation: Data set is not a sequential data set. EQADICXT returns without a TEST run-time option.
User response: Use DTU option: Manage TEST run-time option to create a sequential data set.

EQAU113W First record type is not <PGM> in data set. Record is: datarecord.
Explanation: The type of the first record is not <PGM>. A Program list record must be the first record in the data set. EQADDCTX returns without a TEST run-time option.
User response: Use DTU option: Manage TEST run-time option to create a sequential data set with proper content.

EQAU114W Record type is not <TST> or <RTO> in data set. Record is: datarecord.
Explanation: The type of the record following <PGM> record is not <TST> TEST run-time option or <RTO> other Language Environment run-time option. EQADDCTX returns without a TEST run-time option.
User response: Use DTU option: Manage TEST run-time option to create a sequential data set with proper content.
EQAU115W  Record type is out of sequence in data set. Record is: datarecord.

Explanation: <TST> record is found after <RTO> record. All <TST> records must be grouped together and placed before first <RTO> record. EQADBCXT returns without a TEST run-time option.

User response: Use DTU option: Manage TEST run-time option to create a sequential data set with proper content.

Batch user exit messages

EQAU201E  CSVQuery call failed.

Explanation: CSVQuery failed to return a valid program name. EQADBCXT returns without a TEST run-time option.

User response: Consult your system programmer.

EQAU204W  Data set allocation failed. Data set name is: dname.

Explanation: Data set allocation action failed. EQADBCXT returns without a TEST run-time option.

User response: Use DTU option: Manage TEST run-time option to create a data set.

EQAU205W  Data set open failed. Data set name is: dname.

Explanation: Data set open action failed. EQADBCXT returns without a TEST run-time option.

User response: Consult your system programmer.

EQAU206W  Invalid program name found in data set – pgmname.

Explanation: A program name of more than 8 characters is defined in the data set. EQADBCXT returns without a TEST run-time option.

User response: Use DTU option: Manage TEST run-time option to edit the name on the name list.

EQAU207W  Data set is empty.

Explanation: The data set is empty. EQADBCXT returns without a TEST run-time option.

User response: Use DTU option: Manage TEST run-time option to add contents to the data set.

EQAU208W  No TEST run-time option in data set.

Explanation: The data set does not contain TEST run-time option. EQADBCXT returns without a TEST run-time option.

User response: Use DTU option: Manage TEST run-time option to add TEST run-time option to the data set.

EQAU209E  Data set get failed. Data set name is: dname.

Explanation: Data set get action failed. EQADBCXT returns without a TEST run-time option.

User response: Consult your system programmer.

EQAU210W  Data set name is not correct. Data set name is: dname.

Explanation: Data set name is not correct. EQADBCXT returns without a TEST run-time option.

User response: Verify and correct data set name in EQADBCXT. Assemble and re-link it into your application.

EQAU211W  Data set allocation failed. DD name is not valid.

Explanation: DD name is not valid. EQADBCXT returns without a TEST run-time option.

User response: Verify and correct data set name in EQADBCXT. Assemble and re-link it into your application. If problem persists, consult your system programmer.

EQAU212W  Data set is not physical sequential. Data set name is: dname.

Explanation: Data set is not a sequential data set. EQADBCXT returns without a TEST run-time option.

User response: Use DTU option: Manage TEST run-time option to create a sequential data set.

EQAU213W  First record type is not <PGM> in data set. Record is: datarecord.

Explanation: The type of the first record is not <PGM>. A Program list record must be the first record in the data set. EQADDCXT returns without a TEST run-time option.

User response: Use DTU option: Manage TEST run-time option to create a sequential data set with proper content.

EQAU214W  Record type is not <TST> or <RTO> in data set. Record is: datarecord.

Explanation: The type of the record following <PGM>
record is not <TST> TEST run-time option or <RTO> other Language Environment run-time option.
EQADDCXT returns without a TEST run-time option.

User response: Use DTU option: Manage TEST run-time option to create a sequential data set with proper content.

---

EQAU215W Record type is out of sequence in data set. Record is: datarecord.

Explanation: <TST> record is found after <RTO> record. All <TST> records must be grouped together and placed before first <RTO> record. EQADDCXT returns without a TEST run-time option.

User response: Use DTU option: Manage TEST run-time option to create a sequential data set with proper content.
Chapter 12. Debug Tool Terminal Interface Manager messages

Each message has a number of the form EQAYnnnx, where EQAY indicates that the message is a Debug Tool Terminal Interface Manager message, nnn is the number of the message, and x indicates the severity level of each message. The variable x can be any of the following values:

I  An informational message calls attention to some aspect of a command response that might assist the programmer.

W  A warning message calls attention to a situation that might not be what is expected or to a situation that Debug Tool attempted to fix.

E  An error message describes an error that Debug Tool detected or cannot fix.

S  A severe error message describes an error that indicates a command referring to bad data, control blocks, program structure, or something similar.

U  An unrecoverable error message describes an error that prevents Debug Tool from continuing.

Symbols in messages

Many of the Debug Tool messages contain information that is inserted by the system when the message is issued. In this publication, such inserted information is indicated by italicized symbols, as in the following:

EQA1046I The breakpoint-id breakpoint is replaced.

The portion of Debug Tool located on the host notifies you of errors associated with debugging functions carried out by the host.

---

<table>
<thead>
<tr>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAY999S</td>
<td>Invalid userid/password. Respecify.</td>
</tr>
<tr>
<td></td>
<td><strong>Explanation:</strong> An invalid userid or incorrect password has been specified to Debug Tool Terminal Interface Manager.</td>
</tr>
<tr>
<td></td>
<td><strong>Programmer response:</strong> Respecify the userid and/or password.</td>
</tr>
<tr>
<td>EQAY999S</td>
<td>Error receiving lu name</td>
</tr>
<tr>
<td></td>
<td><strong>Explanation:</strong> Severe internal error (VTAM 3270) in Debug Tool Terminal Interface Manager Please contact your IBM representative.</td>
</tr>
<tr>
<td>EQAY999S</td>
<td>Screen dimensions could not be determined</td>
</tr>
<tr>
<td></td>
<td><strong>Explanation:</strong> Severe internal error (VTAM 3270) in Debug Tool Terminal Interface Manager Please contact your IBM representative.</td>
</tr>
<tr>
<td>EQAY999S</td>
<td>Logon message not available</td>
</tr>
<tr>
<td></td>
<td><strong>Explanation:</strong> Severe internal error (VTAM 3270) in Debug Tool Terminal Interface Manager Please contact your IBM representative.</td>
</tr>
</tbody>
</table>
EQAY999S  No appl ids available.

Explanation: Severe internal error (VTAM 3270) in Debug Tool Terminal Interface Manager Please contact your IBM representative.

EQAY999S  Session not connected

Explanation: Severe internal error (VTAM 3270) in Debug Tool Terminal Interface Manager Please contact your IBM representative.

EQAY999I  Shutting down Debug Tool Terminal Interface Manager

Explanation: Debug Tool Terminal Interface Manager has been requested to shut down.
Chapter 13. Debug Tool Utilities messages

All messages are shown in this section are in mixed case English. The uppercase English message text is the same, but is in uppercase letters.

Each message has a number of the form EQAZnnnx, where EQAZ indicates that the message is an Debug Tool Utilities message, nnn is the number of the message, and x indicates the severity level of each message. The value of x is I, W, E, S, or U, as described below:

I An informational message calls attention to some aspect of a command response that might assist the programmer.

W A warning message calls attention to a situation that might not be what is expected or to a situation that Debug Tool Utilities attempted to fix.

E An error message describes an error that Debug Tool Utilities detected or cannot fix.

S A severe error message describes an error that indicates a command referring to bad data, control blocks, program structure, or something similar.

U An unrecoverable error message describes an error that prevents Debug Tool Utilities from continuing.

Symbols in messages

Many of the Debug Tool Utilities messages contain information that is inserted by the system when the message is issued. In this publication, such inserted information is indicated by italicized symbols, as in the following:

EQA1046I The breakpoint-id breakpoint is replaced.

---

**EQAZ005S** Install Error cmdName has no value for hlq

**Explanation:** cmdName exec detects that no value is assigned for variable hlq.

**System action:** hlq is used as the high level qualifiers for data set names.

**User response:** Follow the instructions “Customizing Debug Tool Utilities” in DT Customization Guide to modify EQASTART to customize data set names.

**EQAZ006S** Install DSN Error cmdName. Missing "dataSet"

**Explanation:** cmdName cannot find dataSet.

**System action:** cmdName exec ends.

**User response:** Follow the ‘Customizing Debug Tool Utilities’ in DT Customization Guide to modify EQASTART to customize data set names.

**EQAZ007S** libType LIBDEF Failed for lib

**Explanation:** Allocation of application library of type libType failed for data set lib.

**System action:** Debug Tool Utility ends.

**User response:** Follow the instructions “Customizing Debug Tool Utilities” in DT Customization Guide to modify EQASTART to customize data set names.

**EQAZ008S** libType ALTLIB Failed for lib

**Explanation:** Define of alternative application library of type libType failed for lib

**System action:** Debug Tool Utility ends.

**User response:** Follow the instructions “Customizing Debug Tool Utilities” in DT Customization Guide to modify EQASTART to customize data set names.

**EQAZ010W** Allocation Error: dsnName

**Explanation:** Allocation failed for dsnName.

**System action:** dsnName is not processed.

**User response:** Make sure that dsnName exists.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAZ011I</td>
<td>Invalid Command cmd for Panel pnlld</td>
<td>Invalid command cmd is entered in panel pnlld.</td>
</tr>
<tr>
<td>User response</td>
<td>Make sure that DISP is specified correctly.</td>
<td></td>
</tr>
<tr>
<td>System action</td>
<td>The command is not processed</td>
<td></td>
</tr>
<tr>
<td>User response</td>
<td>Enter a valid command.</td>
<td></td>
</tr>
<tr>
<td>EQAZ012I</td>
<td>Invalid Member name specified. Dataset dsnName is not partitioned.</td>
<td>Data set dsnName is a sequential file. A member name cannot be specified.</td>
</tr>
<tr>
<td>User response</td>
<td>Remove the member name or specify a partitioned data set</td>
<td></td>
</tr>
<tr>
<td>System action</td>
<td>You are prompted to correct the problem.</td>
<td></td>
</tr>
<tr>
<td>EQAZ013W</td>
<td>EXECIO Error for Data Set dsnName.</td>
<td>I/O error in read/write data set dsnName.</td>
</tr>
<tr>
<td>User response</td>
<td>Report the problem to the system administrator.</td>
<td></td>
</tr>
<tr>
<td>System action</td>
<td>I/O is ended.</td>
<td></td>
</tr>
<tr>
<td>EQAZ014E</td>
<td>Multiple jobs detected. Only one job is allowed. By default, only the first job in the sequence will be run.</td>
<td>Multiple jobs are found in the JCL being copied into the setup file.</td>
</tr>
<tr>
<td>System action</td>
<td>Only the first job is copied.</td>
<td></td>
</tr>
<tr>
<td>User response</td>
<td>Make sure that you select only one job when copying in the JCL.</td>
<td></td>
</tr>
<tr>
<td>EQAZ015W</td>
<td>Multiple programs detected. Only one program is allowed. By default, only the first program in the sequence will be selected.</td>
<td>Multiple steps in a job are selected in the JCL that is being copied into the setup file.</td>
</tr>
<tr>
<td>System action</td>
<td>Only the first step is copied.</td>
<td></td>
</tr>
<tr>
<td>User response</td>
<td>Make sure that you select only one step in the job.</td>
<td></td>
</tr>
<tr>
<td>EQAZ016W</td>
<td>InvalidConcatenation dsn. DISP disposition not allowed in the middle of a concatenation.</td>
<td>DISP disposition is not allowed in the middle of a concatenation.</td>
</tr>
<tr>
<td>System action</td>
<td>RUN command ends.</td>
<td></td>
</tr>
<tr>
<td>EQAZ017I</td>
<td>Program pgmName ended with Return Code rc</td>
<td>Program pgmName has been executed in the foreground with a return code of rc.</td>
</tr>
<tr>
<td>User response</td>
<td>Make sure that return code rc is what you expect.</td>
<td></td>
</tr>
<tr>
<td>System action</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>EQAZ018I</td>
<td>Specify the allocation defaults for new Setup File fileName</td>
<td>A new setup file fileName is entered.</td>
</tr>
<tr>
<td>User response</td>
<td>Enter allocation defaults in the next panel.</td>
<td></td>
</tr>
<tr>
<td>System action</td>
<td>You are prompted for allocation defaults.</td>
<td></td>
</tr>
<tr>
<td>EQAZ019W</td>
<td>cmd allowed for the first library of concatenation only.</td>
<td>Delete or Rename command is allowed for the first library of concatenation only.</td>
</tr>
<tr>
<td>System action</td>
<td>cmd is not executed.</td>
<td></td>
</tr>
<tr>
<td>User response</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>EQAZ020I</td>
<td>fileName has been actionPerformed.</td>
<td>File fileName has been actionPerformed. (such as saved)</td>
</tr>
<tr>
<td>System action</td>
<td>Processing continues.</td>
<td></td>
</tr>
<tr>
<td>User response</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>EQAZ021W</td>
<td>Member not found in copy from data set dsnName. No data has been copied.</td>
<td>dsnName is not found in a copy command processing.</td>
</tr>
<tr>
<td>System action</td>
<td>Copy command ends; no data is copied.</td>
<td></td>
</tr>
<tr>
<td>User response</td>
<td>Enter an existing data set name.</td>
<td></td>
</tr>
<tr>
<td>EQAZ022W</td>
<td>dsnName does not contain JCL or a valid setup file.</td>
<td>dsnName is not a valid JCL file or a valid setup file.</td>
</tr>
<tr>
<td>System action</td>
<td>Copy command ends; no data is copied.</td>
<td></td>
</tr>
<tr>
<td>User response</td>
<td>Enter an valid file.</td>
<td></td>
</tr>
</tbody>
</table>
EQAZ023W You must select either MFI or TCPIP session type.
Explanation: Both MFI and TCPIP session types are selected.
System action: You are prompted to select again.
User response: Select only one session type.

EQAZ024W You must specify the workstation TCP/IP identifier.
Explanation: TCP/IP identifier field is empty.
System action: You are prompted to enter the identifier.
User response: Enter TCP/IP identifier.

EQAZ025W You must specify a load module or program name to be run.
Explanation: Load Module Name field is empty.
System action: Run command ends.
User response: Enter a load module name.

EQAZ026W You must specify Directory blocks with
Data set name type dsType.
Explanation: Directory block must be greater than zero for data set name type dsType.
System action: You are prompted for the correct value.
User response: Enter an non-zero directory block.

EQAZ027E Invalid line command detected. cmd is not allowed for ddn DD statement.
Explanation: Invalid line command entered.
System action: cmd is not executed.
User response: Enter a valid command.

EQAZ029W cmd return code rc
Explanation: cmd (edit, view, or browse) command has a return code of rc.
System action: Processing continues.
User response: Refer to the ISPF documentation for return code meaning.

EQAZ030W Member specified for fileName but type is not PDSE or PDS.
Explanation: A member is specified for fileName, but its type is not PDSE or PDS.
System action: You are prompted to correct the problem.
User response: Remove the member name, or specify a PDSE or PDS.

EQAZ031W Member not specified for fileName for type PDSE or PDS
Explanation: No member is specified for fileName of type PDSE or PDS
System action: You are prompted to correct the problem.
User response: Enter a member name.

EQAZ033W An invalid data set pattern character pChar was used.
Explanation: An invalid character pChar was used in a data set pattern in the program preparation.
System action: You are prompted to correct the problem.
User response: Use field help to choose a valid pattern character. (To see field help, press the HELP key with the cursor positioned in the field.)

EQAZ034S fileName is multiply included. Circular definitions are not allowed.
Explanation: fileName is included multiple times in the settings file or files.
System action: Include statement is not processed.
User response: Contact your system administrator. Make sure that fileName is not included more than once in the same settings file or nested settings file.

EQAZ035S Too many INCLUDE statements were found in the settings file
Explanation: More than sixteen INCLUDE statements were found in the settings file or files.
System action: Include statements starting with the sixteenth are not processed.
User response: Contact your system administrator. Make sure that the number of INCLUDE statements does not exceed sixteen in the same settings file or nested settings file.

EQAZ036E End of file for member fileName while processing statement
Explanation: Incomplete statement is found in fileName settings file.
System action: The statement is not processed.
User response: Contact your system administrator.
Make sure that the statement in `fileName` is properly ended with a semicolon.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAZ037E</td>
<td>Invalid keyword <code>kwd</code> found in member <code>fileName.lineNo</code></td>
</tr>
<tr>
<td>Explanation:</td>
<td>Invalid keyword <code>kwd</code> found in line <code>lineNo</code> of settings file member <code>fileName</code>.</td>
</tr>
<tr>
<td>System action:</td>
<td>Include statement is not processed.</td>
</tr>
<tr>
<td>User response:</td>
<td>Contact your system administrator. Make sure that the statement in line <code>lineNo</code> of settings file member <code>fileName</code> is a valid statement.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAZ038I</td>
<td>You must supply a valid job card to use batch.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>A request to run the application in batch is submitted but no valid job card is found.</td>
</tr>
<tr>
<td>System action:</td>
<td>You are prompted for job card information.</td>
</tr>
<tr>
<td>User response:</td>
<td>Enter job card information.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAZ039W</td>
<td>An invalid sequence number <code>seqValue</code> was entered.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>A non-numeric value <code>seqValue</code> is entered in the sequence number field.</td>
</tr>
<tr>
<td>System action:</td>
<td><code>seqValue</code> is removed from the field.</td>
</tr>
<tr>
<td>User response:</td>
<td>Enter a numeric value.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAZ040I</td>
<td>Action started for <code>srcName</code> using <code>inName</code>.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>Action, which can be any program preparation action (compile, assemble, link-edit, create FA side file, or convert old COBOL program), is started for source <code>srcName</code> using input <code>inName</code>.</td>
</tr>
<tr>
<td>System action:</td>
<td>Processing continues.</td>
</tr>
<tr>
<td>User response:</td>
<td>None.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAZ041I</td>
<td>CICS translator <code>transName</code> invoked from <code>complib</code>.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>CICS translator <code>transName</code> is invoked from <code>complib</code> (LINKLIST or a library).</td>
</tr>
<tr>
<td>System action:</td>
<td>Processing continues.</td>
</tr>
<tr>
<td>User response:</td>
<td>None.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAZ044I</td>
<td>Converter <code>convName</code> invoked from <code>complib</code>.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>COBOL converter <code>convName</code> is invoked from <code>complib</code> (LINKLIST or a library).</td>
</tr>
<tr>
<td>System action:</td>
<td>Processing continues.</td>
</tr>
<tr>
<td>User response:</td>
<td>None.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAZ045I</td>
<td>DB2 preprocessor started for <code>fileName</code>.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>DB2 preprocessor starts to process <code>fileName</code>.</td>
</tr>
<tr>
<td>System action:</td>
<td>Processing continues.</td>
</tr>
<tr>
<td>User response:</td>
<td>None.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAZ046I</td>
<td>DB2 Preprocessor <code>db2preName</code> invoked from <code>complib</code>.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>DB2 Preprocessor <code>db2preName</code> is invoked from <code>complib</code> (LINKLIST or a library).</td>
</tr>
<tr>
<td>System action:</td>
<td>Processing continues.</td>
</tr>
<tr>
<td>User response:</td>
<td>None.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAZ047I</td>
<td>EQALANGX started for <code>fileName</code>.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>EQALANGX starts to process <code>fileName</code>.</td>
</tr>
<tr>
<td>System action:</td>
<td>Processing continues.</td>
</tr>
<tr>
<td>User response:</td>
<td>None.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAZ048I</td>
<td>EQALANGX <code>idilName</code> invoked from <code>complib</code>.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>EQALANGX <code>idilName</code> is invoked from <code>complib</code> (LINKLIST or a library).</td>
</tr>
<tr>
<td>System action:</td>
<td>Processing continues.</td>
</tr>
<tr>
<td>User response:</td>
<td>None.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAZ049I</td>
<td>CICS translator started for <code>fileName</code>.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>CICS translator starts to translate <code>fileName</code>.</td>
</tr>
<tr>
<td>System action:</td>
<td>Processing continues.</td>
</tr>
<tr>
<td>User response:</td>
<td>None.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAZ049W</td>
<td>You must specify the parameter string format.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>You must specify the parameter string format in order to modify parameters.</td>
</tr>
<tr>
<td>System action:</td>
<td>You are prompted for the correct choice.</td>
</tr>
<tr>
<td>User response:</td>
<td>Choose a format (1 or 2).</td>
</tr>
</tbody>
</table>
EQAZ050W  Allocation of a temporary data set failed.
Explanation:  The allocation of a temporary data set for the SYSIN DD statement failed when the RUN command was run.
System action:  The RUN command ends.
User response:  Report the problem to your system administrator.

EQAZ051S  Internal error in exec execName. Invalid panel = panelName.
Explanation:  execName exec is invoked with an invalid panel (panelName) in the parameter list.
System action:  execName exec ends.
User response:  Report the problem to IBM.

EQAZ052S  Internal error in exec execName. Invalid command = cmdName.
Explanation:  execName exec is invoked with an invalid command (cmdName) in the parameter list.
System action:  execName exec ends.
User response:  Report the problem to IBM.

EQAZ053W  Invalid DSN type entered - dsnType.
Explanation:  An invalid DSN type (dsnType) was entered. Valid types are PDS, PDSE, and SEQ.
System action:  You are prompted for the correct value.
User response:  Enter a valid value.

EQAZ054W  LISTDSI failed for dsnName
           Level1ErrorMsg Level2ErrorMsg
Explanation:  LISTDSI was performed on dsnName but it returned with error messages - Level1ErrorMsg and Level2ErrorMsg
System action:  The starting panel of program preparation is presented.
User response:  Report the problem to the system administrator.

EQAZ055W  Internal error in exec execName. Too many variable types, varTypeList.
Explanation:  execName exec was invoked with too many variable types (varTypeList) in the parameter list.
System action:  execName exec ends.
User response:  Report the problem to IBM.

EQAZ056S  Internal error in exec execName. Invalid variable type = varType.
Explanation:  execName exec was invoked with invalid variable type (varType) in the parameter list.
System action:  execName exec ends.
User response:  Report the problem to IBM.

EQAZ057W  Data set dsnName is not available - errorMsg.
Explanation:  The data set is not available for the reason specified in errorMsg.
System action:  The action on the data set is not performed. The program preparation completion panel is presented.
User response:  Check the program preparation return code on the panel. Some data sets might not be available if program preparation fails.

EQAZ058W  dsnUse Data set dsnName is not available - errorMsg.
Explanation:  The data set is not available for the reason specified in errorMsg.
System action:  The action on the data set is not performed. The program preparation panel is presented.
User response:  Check the program preparation return code on the panel. Report the problem to the system administrator.

EQAZ059W  No IMSPlex ID
Explanation:  IMSPlex ID is required.
System action:  You are prompted for an IMSPlex ID.
User response:  Enter a valid IMSPlex ID. Contact your system administrator if you do not have an ID.

EQAZ060E  No REXX IMS SPOC
Explanation:  REXX IMS SPOC environment is not available. Return Code = RC.
System action:  The action on LE runtime options is not performed.
User response:  Contact your system administrator and verify that IMS V8 is installed on your system and that Debug Tool Utility is properly installed and configured. See “Debug Tool Customization Guide” for details.
<table>
<thead>
<tr>
<th>Code</th>
<th>Message Description</th>
<th>Explanation</th>
<th>System action</th>
<th>User response</th>
<th>User response</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAZ061E</td>
<td>IMS SPOC command failed</td>
<td>IMS SPOC command failed. Return Code = RC.</td>
<td>The action on LE runtime options is not performed.</td>
<td>Verify that IMSplex ID is correctly specified. Contact your system administrator and verify that IMS V8 is installed on your system and that Debug Tool Utility is properly installed and configured. See &quot;Debug Tool Customization Guide&quot; for details.</td>
<td></td>
</tr>
<tr>
<td>EQAZ062E</td>
<td>IMS OM security error</td>
<td>IMS Operations Manager security check failed. SAF return Code = SAF_RC; RACF return code = RACF_RC, reason code = reason_code; Exit return code = EXIT_RC, user data = user_data.</td>
<td>The action on LE runtime options is not performed.</td>
<td>Contact your system administrator to request that your ID be authorized to use IMS QUERY LE and UPD LE commands.</td>
<td></td>
</tr>
<tr>
<td>EQAZ063E</td>
<td>Incorrect data</td>
<td>Quote is not allowed in this field.</td>
<td>You are prompted for correct data.</td>
<td>Enter a valid value for this field.</td>
<td></td>
</tr>
<tr>
<td>EQAZ064E</td>
<td>IMS command failed</td>
<td>IMS command_name command failed. Return code = IMS_RC, Reason code = IMS_reason_code.</td>
<td>The action on LE runtime options is not performed.</td>
<td>Contact your system administrator and verify that IMS V8 is installed on your system and that Debug Tool Utility is properly installed and configured. See &quot;Debug Tool Customization Guide&quot; for details.</td>
<td></td>
</tr>
<tr>
<td>EQAZ065E</td>
<td>Non-LE program cannot have load module name = EQANMDBG</td>
<td>EQANMDBG is a reserved load module name for Debug Tool when debugging a non-LE program.</td>
<td>You are prompted to correct the problem.</td>
<td>Enter a correct load module name.</td>
<td></td>
</tr>
<tr>
<td>EQAZ066E</td>
<td>Invalid DTU setup file</td>
<td>The input file is not a valid DTU setup file and may be overwritten. Press Cancel to return.</td>
<td>The original content of the input file may be overwritten if processing continues.</td>
<td>Press Cancel to return and enter a valid input file or an empty file.</td>
<td></td>
</tr>
<tr>
<td>EQAZ067W</td>
<td>Not enabled for MORE</td>
<td>This field is not enabled for additional input space.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQAZ068W</td>
<td>Cursor not in a field</td>
<td>Cursor is not in a field when MORE command is entered.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQAZ069E</td>
<td>dsnName is not a sequential data set</td>
<td>dsnName is not a sequential data set. The TEST run-time option data set must be a sequential data set.</td>
<td>The action on the data set is not performed.</td>
<td>Provide a sequential data set.</td>
<td></td>
</tr>
</tbody>
</table>
Appendix A. Support information

If you have a problem with your IBM software, you want to resolve it quickly. This section describes the following options for obtaining support for IBM software products:

- “Searching knowledge bases”
- “Obtaining fixes”
- “Receiving weekly support updates” on page 358
- “Contacting IBM Software Support” on page 358

Searching knowledge bases
You can search the available knowledge bases to determine whether your problem was already encountered and is already documented.

Searching the information center
IBM provides this documentation in an information center. You can use the search function of the information center to query conceptual information, instructions for completing tasks, and reference information.

Searching the Internet
If you cannot find an answer to your question in the information center, search the Internet for the latest, most complete information that might help you resolve your problem.

To search multiple Internet resources for your product, use the Web search topic in your information center. In the navigation frame, click Troubleshooting and support ▶ Searching knowledge bases and select Web search. From this topic, you can search a variety of resources, including the following:

- IBM technotes
- IBM downloads
- IBM Redbooks™
- IBM developerWorks®
- Forums and newsgroups
- Google

Obtaining fixes
A product fix might be available to resolve your problem. To determine what fixes are available for your IBM software product, follow these steps:

2. Click Downloads and drivers in the Support topics section.
3. Select the Software category.
4. Select a product in the Sub-category list.
5. In the Find downloads and drivers by product section, select one software category from the Category list.
6. Select one product from the Sub-category list.
7. Type more search terms in the **Search within results** if you want to refine your search.
8. Click **Search**.
9. From the list of downloads returned by your search, click the name of a fix to read the description of the fix and to optionally download the fix.

For more information about the types of fixes that are available, see the IBM Software Support Handbook at [http://techsupport.services.ibm.com/guides/handbook.html](http://techsupport.services.ibm.com/guides/handbook.html)

### Receiving weekly support updates

To receive weekly e-mail notifications about fixes and other software support news, follow these steps:

2. Click **My support** in the upper right corner of the page.
3. If you have already registered for **My support**, sign in and skip to the next step. If you have not registered, click **register now**. Complete the registration form using your e-mail address as your IBM ID and click **Submit**.
4. Click **Edit profile**.
5. In the **Products** list, select **Software**. A second list is displayed.
6. In the second list, select a product segment, for example, **Application servers**. A third list is displayed.
7. In the third list, select a product sub-segment, for example, **Distributed Application & Web Servers**. A list of applicable products is displayed.
8. Select the products for which you want to receive updates, for example, **IBM HTTP Server** and **WebSphere Application Server**.
9. Click **Add products**.
10. After selecting all products that are of interest to you, click **Subscribe to email** on the **Edit profile** tab.
11. Select **Please send these documents by weekly email**.
12. Update your e-mail address as needed.
13. In the **Documents** list, select **Software**.
14. Select the types of documents that you want to receive information about.
15. Click **Update**.

If you experience problems with the **My support** feature, you can obtain help in one of the following ways:

**Online**
Send an e-mail message to erchelp@ca.ibm.com, describing your problem.

**By phone**
Call 1-800-IBM-4You (1-800-426-4968).

### Contacting IBM Software Support

IBM Software Support provides assistance with product defects.
Before contacting IBM Software Support, your company must have an active IBM software maintenance contract, and you must be authorized to submit problems to IBM. The type of software maintenance contract that you need depends on the type of product you have:

- For IBM distributed software products (including, but not limited to, Tivoli®, Lotus®, and Rational® products, as well as DB2 and WebSphere products that run on Windows, or UNIX operating systems), enroll in Passport Advantage® in one of the following ways:

  **Online**
  
  Go to the Passport Advantage Web site at [http://www.lotus.com/services/passport.nsf/WebDocs/Passport_Advantage_Home](http://www.lotus.com/services/passport.nsf/WebDocs/Passport_Advantage_Home) and click How to Enroll.

  **By phone**
  
  For the phone number to call in your country, go to the IBM Software Support Web site at [http://techsupport.services.ibm.com/guides/contacts.html](http://techsupport.services.ibm.com/guides/contacts.html) and click the name of your geographic region.

- For customers with Subscription and Support (S & S) contracts, go to the Software Service Request Web site at [https://techsupport.services.ibm.com/ssr/login](https://techsupport.services.ibm.com/ssr/login)


- For IBM eServer™ software products (including, but not limited to, DB2 and WebSphere products that run in zSeries, pSeries, and iSeries environments), you can purchase a software maintenance agreement by working directly with an IBM sales representative or an IBM Business Partner. For more information about support for eServer software products, go to the IBM Technical Support Advantage Web site at [http://www.ibm.com/servers/eserver/techsupport.html](http://www.ibm.com/servers/eserver/techsupport.html)

If you are not sure what type of software maintenance contract you need, call 1-800-IBMSERV (1-800-426-7378) in the United States. From other countries, go to the contacts page of the IBM Software Support Handbook on the Web at [http://techsupport.services.ibm.com/guides/contacts.html](http://techsupport.services.ibm.com/guides/contacts.html) and click the name of your geographic region for phone numbers of people who provide support for your location.

To contact IBM Software support, follow these steps:

1. “Determining the business impact”
2. “Describing problems and gathering information” on page 360
3. “Submitting problems” on page 360

**Determining the business impact**

When you report a problem to IBM, you are asked to supply a severity level. Therefore, you need to understand and assess the business impact of the problem that you are reporting. Use the following criteria:

**Severity 1**

The problem has a critical business impact. You are unable to use the program, resulting in a critical impact on operations. This condition requires an immediate solution.

**Severity 2**

The problem has a significant business impact. The program is usable, but it is severely limited.
Severity 3
The problem has some business impact. The program is usable, but less significant features (not critical to operations) are unavailable.

Severity 4
The problem has minimal business impact. The problem causes little impact on operations, or a reasonable circumvention to the problem was implemented.

Describing problems and gathering information
When describing a problem to IBM, be as specific as possible. Include all relevant background information so that IBM Software Support specialists can help you solve the problem efficiently. To save time, know the answers to these questions:

- What software versions were you running when the problem occurred?
- Do you have logs, traces, and messages that are related to the problem symptoms? IBM Software Support is likely to ask for this information.
- Can you re-create the problem? If so, what steps were performed to re-create the problem?
- Did you make any changes to the system? For example, did you make changes to the hardware, operating system, networking software, and so on.
- Are you currently using a workaround for the problem? If so, be prepared to explain the workaround when you report the problem.

Submitting problems
You can submit your problem to IBM Software Support in one of two ways:

Online
Click Submit and track problems on the IBM Software Support site at [http://www.ibm.com/software/support/probsub.html](http://www.ibm.com/software/support/probsub.html). Type your information into the appropriate problem submission form.

By phone
For the phone number to call in your country, go to the contacts page of the IBM Software Support Handbook at [http://techsupport.services.ibm.com/guides/contacts.html](http://techsupport.services.ibm.com/guides/contacts.html) and click the name of your geographic region.

If the problem you submit is for a software defect or for missing or inaccurate documentation, IBM Software Support creates an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Software Support provides a workaround that you can implement until the APAR is resolved and a fix is delivered. IBM publishes resolved APARs on the Software Support Web site daily, so that other users who experience the same problem can benefit from the same resolution.
Appendix B. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The accessibility features in z/OS provide accessibility for Debug Tool.

The major accessibility features in z/OS enable users to:
• Use assistive technology products such as screen readers and screen magnifier software
• Operate specific or equivalent features by using only the keyboard
• Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products work with the user interfaces that are found in z/OS. For specific guidance information, consult the documentation for the assistive technology product that you use to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces by using TSO/E or ISPF. Refer to z/OS TSO/E Primer, z/OS TSO/E User’s Guide, and z/OS ISPF User’s Guide Volume 1 for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Accessibility of this document

Information in the following formats of this document is accessible to visually impaired individuals who use a screen reader:
• PDF format when viewed with Adobe Acrobat Reader 5.0 or later
• BookManager® format when viewed with IBM BookManager BookServer (except for syntax diagrams)

Syntax diagrams start with the word Format or the word Fragments. Each diagram is preceded by two images. For the first image, the screen reader will say "Read syntax diagram". The associated link leads to an accessible text diagram. When you return to the document at the second image, the screen reader will say "Skip visual syntax diagram" and has a link to skip around the visible diagram.

For BookManager users only: A screen reader might say the lines, symbols, and words in a diagram, but not in a meaningful way. For example, you might hear "question question dash dash MOVE dash dash plus dash dash literal-1 dash dash plus" for part of the MOVE statement. You can enter Say Next Paragraph to move quickly through syntax diagrams if your screen reader has that capability.
Notices

This information was developed for products and services offered in the U.S.A. IBM might not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with the local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement might not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.
Copyright license

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or functions of these programs.

Programming interface information

This book is intended to help you debug application programs. This publication documents intended Programming Interfaces that allow you to write programs to obtain the services of Debug Tool.

Trademarks and service marks

The following terms, denoted by an asterisk (*) on the first occurrence in this publication, are trademarks or service marks of International Business Machines Corporation in the United States or other countries:

- AD/Cycle
- AIX
- C/370
- CICS
- COBOL/370
- DB2
- DB2 Universal Database
- developerWorks
- eServer
- FFST/2
- IBM
- IBMLink
- IMS
- iSeries
- Language Environment
- Lotus
- MVS
- MVS/ESA
- OS/2
- OS/390
- Passport Advantage
- pSeries
- RACF
- Rational
- Redbooks
- Resource Link
- S/390
- SOM
- System/370
- Tivoli
- VisualAge
- VSE/ESA
- VTAM
- WebSphere
- z/OS
- z/VM
- zSeries

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Java™ and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

LINUX is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and Windows NT® are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a trademark of UNIX System Laboratories, licensed exclusively by X/Open Company, Ltd.
Glossary

This glossary defines technical terms and abbreviations used in Debug Tool Reference and Messages documentation. If you do not find the term you are looking for, refer to the IBM Glossary of Computing Terms, located at the IBM Terminology web site: http://www.ibm.com/ibm/terminology

D

DTCN. Debug Tool Control utility, a CICS transaction that enables the user to identify which CICS programs to debug.

debugging profile. Data that specifies a set of application programs which are to be debugged together.

I

index. A computer storage position or register, the contents of which identify a particular element in a table.

M

minor node. In VTAM, a uniquely defined resource within a major node.

multitasking. A mode of operation that provides for concurrent performance, or interleaved execution of two or more tasks.

O

offset. The number of measuring units from an arbitrary starting point to some other point.

S

Single Point of Control. The control interface that sends commands to one or more members of an IMSplex and receives command responses.

SPOC. See "Single Point of Control."
Bibliography

Debug Tool publications
Using CODE/370 with VS COBOL II and OS PL/I, SC09-1862

Debug Tool for z/OS
Debug Tool Summary of Commands and Built-in Functions, SC19-1074
Debug Tool Customization Guide, SC19-1075
Program Directory for Debug Tool for z/OS, GI10-8726
Debug Tool Reference and Messages, GC19-1073
Debug Tool User’s Guide, SC19-1071

Debug Tool Utilities and Advanced Functions for z/OS
Debug Tool Coverage Utility Users Guide, SC19-1072
Program Directory for Debug Tool Utilities and Advanced Functions for z/OS, GI10-8727

High level language publications
z/OS C and C++
Compiler and Run-Time Migration Guide, GC09-4913
Curses, SA22-7820
Language Reference, SC09-4815
Programming Guide, SC09-4765
Run-Time Library Reference, SA22-7821
User’s Guide, SC09-4767

Enterprise COBOL for z/OS and OS/390
Migration Guide, GC27-1409
Customization, GC27-1410
Licensed Program Specifications, GC27-1411
Language Reference, SC27-1408
Programming Guide, SC27-1412

COBOL for OS/390 & VM
Compiler and Run-Time Migration Guide, GC26-4764
Customization under OS/390, GC26-9045
Language Reference, SC26-9046
Programming Guide, SC26-9049

Enterprise PL/I for z/OS and OS/390
Diagnosis Guide, SC27-1459
Language Reference, SC27-1460
Licensed Program Specifications, GC27-1456
Messages and Codes, SC27-1461
Migration Guide, GC27-1458
Programming Guide, SC27-1457

VisualAge PL/I for OS/390
Compiler and Run-Time Migration Guide, SC26-9474
Diagnosis Guide, SC26-9475
Language Reference, SC26-9476
Licensed Program Specifications, GC26-9471
Messages and Codes, SC26-9478
Programming Guide, SC26-9473

PL/I for MVS & VM
Compile-Time Messages and Codes, SC26-3229
Compiler and Run-Time Migration Guide, SC26-3118
Diagnosis Guide, SC26-3149
Installation and Customization under MVS, SC26-3119
Language Reference, SC26-3114
Licensed Program Specifications, GC26-3116
Programming Guide, SC26-3113
Reference Summary, SX26-3821

Related publications
CICS
Application Programming Guide, SC34-6231
Application Programming Primer, SC34-0674
Application Programming Reference, SC34-6232

DB2 Universal Database™ for z/OS
Administration Guide, SC18-7413
Application Programming and SQL Guide, SC18-7415
Command Reference, SC18-7416
Data Sharing: Planning and Administration, SC18-7417
Installation Guide, GC18-7418
Messages and Codes, GC18-7422
"Reference for Remote DRDA* Requesters and Servers", SC18-7424
"Release Planning Guide", SC18-7425
"SQL Reference", SC18-7426
"Utility Guide and Reference", SC18-7427

IMS
"IMS Application Programming: Database Manager", SC27-1286
"IMS Application Programming: EXEC DLI Commands for CICS & IMS", SC27-1288
"IMS Application Programming: Transaction Manager", SC27-1289

TSO/E
"Command Reference", SA22-7782
"Programming Guide", SA22-7788
"System Programming Command Reference", SA22-7793
"User’s Guide", SA22-7794

z/OS
"MVS JCL Reference", SA22-7597
"MVS JCL User’s Guide", SA22-7598
"MVS System Commands", SA22-7627

z/OS Language Environment
"Concepts Guide", SA22-7567
"Customization", SA22-7564
"Debugging Guide", GA22-7560
"Programming Guide", SA22-7561
"Programming Reference", SA22-7562
"Run-Time Migration Guide", GA22-7565
"Vendor Interfaces", SA22-7568
"Writing Interlanguage Communication Applications", SA22-7563

Softcopy publications
Online publications are distributed on CD-ROMs and can be ordered through your IBM representative. "Debug Tool User’s Guide", "Debug Tool Customization Guide", and "Debug Tool Reference and Messages" are distributed on the following collection kit:
SK3T-4269

Online publications can also be downloaded from the IBM Web site. Visit the IBM Web site for each product to find online publications for that product.
CLEAR prefix (full-screen mode) 83
restriction with playback 83
CLOSE, WINDOW command 234
COBOL
AT DATE command 51
commands
CALL entry_name 77
COMPUTE 84
EVALUATE 102
IF 115
INPUT 121
MOVE 145
PERFORM 153
SET 216
SET INTERCEPT 191
SET, allowable moves 218
declarations 89
HEX built-in function 240
SET WARNING, to modify variables in optimized code 215
STORAGE built-in function 46
COLOR, SET command 180
COLORS, QUERY command 163
COLUMN, SET MONITOR command 197
command
syntax diagrams x
command suboption of TEST run-time option 6
commands delimiting 67
commands_file of TEST run-time option 5
commands_file_designator suboption of TEST run-time option 5
COMMENT command 84
common syntax elements 11
comparisons, allowable (IF command for COBOL) 116
compile unit
record of associations, Debug Tool 152
compile_unit_name, description 12
COMPUTE command (COBOL) 84
condition constants, for C 230
for C 60
condition, description 12
COUNTRY, QUERY command 163
COUNTRY, SET command 182
cu_spec, description 13
CURRENT VIEW, QUERY command 163
cursor commands
cursor, AT 50
cursor, LIST (full-screen mode) 128
customer support
See Software Support

D
datatype, SET MONITOR command 197
DATE, AT command 51
dbc
SET DBCS command 182
variable, assigning new value to 146
DBCS, QUERY command 163
DBCS, SET command 182
Debug Tool
termology ix
declarations
for assembler 86
for C and C++ 87
for COBOL 89
for disassembly 86
for OS/VS COBOL 86
declarations (C and C++)
restriction with playback 89
DECLARE command (PL/I) 91
restriction with playback 92
DEFAULT LISTINGS, QUERY command 163
DEFAULT LISTINGS, SET command 183
DEFAULT SCROLL, QUERY command 163
DEFAULT SCROLL, SET command 184
DEFAULT VIEW, QUERY command 163
DEFAULT VIEW, SET command 185
DEFAULT WINDOW, QUERY command 163, 164
DEFAULT WINDOW, SET command 185
DELETE, AT command 51
DESCRIBE command
description 93
restriction with playback 96
DISABLE command 96, 97
restriction with playback 97
disassembly commands not supported for GOTO 111
JUMPTO 122
PLAYBACK 155
WHERE built-in function 243
disassembly commands
AT OFFSET 63
declarations 86
DO 98
IF 113
SET DISASSEMBLY 186
while 233
DISASSEMBLY, QUERY command 163
DISASSEMBLY, SET command 186
display point of execution 205
DO command (assembler, disassembly, OS/VS COBOL) 98
DO command (PL/I) 99
restriction with playback 101
do/while command (C and C++) 98
restriction with playback 98
documents, licensed vii
DTST
See Debug Tool Setup Utility
DYNDEBUG, QUERY command 163
DYNDEBUG, SET command 186
E
ECHO, QUERY command 163
ECHO, SET command 188
ENABLE command 101, 102
restriction with playback 101
Enterprise PL/I
commands not supported for GOTO LABEL 113
Enterprise PL/I, definition of ix
entry_name, CALL command (COBOL) 77
ENTRY, AT command 52
EQUATE, SET command
description 188
EQUATES, QUERY command 163
ERROR suboption of TEST run-time option 5
EVALUATE command (COBOL)
description 102
every_clause, description 40
examples
assembler instructions with implied length 19
assembler instructions without implied length 19
compile_unit_name 13
EXECUTE, QUERY command 163
EXECUTE, SET command 189
executing
See running
EXIT, AT command 52
expression command (C and C++) 104
restriction with playback 104
expression, LIST command 129
expressions
description 14
subset, description 15
F
FIND command
description 105
fixes, obtaining 357
for command (C and C++) 109
FREE command 110
FREQUENCY, LIST command 132
FREQUENCY, QUERY command 163
FREQUENCY, SET command 190
full-screen mode
AT CURSOR command 50
AT prefix 64
CLEAR prefix 83
CURSOR command 85
DESCRIBE CURSOR 93
DISABLE prefix 97
ENABLE prefix 102
FIND 105
IMMEDIATE 120
LIST CURSOR 128
PANEL 151
PANEL LISTINGS 152
PANEL SOURCE 152
prefix commands 159
QUERY prefix 166
RETRIEVE 169
SET COLOR 180
full-screen mode (continued)
SET DEFAULT SCROLL 184
SET DEFAULT WINDOW 185
SET KEYS 192
SET LOG NUMBERS 195
SET MONITOR 196
SET MONITOR COLUMN 197
SET MONITOR DATATYPE 197
SET MONITOR NUMBERS 197
SET MONITOR WRAP 197
SET PROMPT 202
SET SAVE 207
SET SCREEN 210
SET SCROLL DISPLAY 211
SET SUFFIX 214
SHOW prefix 220
WINDOW CLOSE 234
WINDOW OPEN 235
WINDOW SIZE 235
WINDOW ZOOM 236
function, Debug Tool
%DEC 239
%GENERATION 240
%HEX 240
%INSTANCES 241
%RECURSION 242
%WHERE 243
functions, Debug Tool
%STORAGE 46
summary 239

INPUT command
restriction with playback 121
INPUT command (C, C++, COBOL) 121
INSPLLOG
using with SET LOG command 194
INSPPREF suboption of TEST run-time option 7
INSTANCES built-in function 241
INTERCEPT, QUERY command 164
Internet
searching for problem resolution 357
invoking
See starting
ISPF
SET REFRESH command 205

J
JUMP TO command 122
JUIMPTO command 122
restriction with playback 122
JUMP TO LABEL command 123
restriction with playback 124

K
KEYS, SET command 192
knowledge bases, searching for problem resolution 357

L
LABEL, AT command 55
LANGUAGE, SET NATIONAL command 198
LANGUAGE, SET PROGRAMMING command 201
LAST, LIST command 132
LDD
See LOADDEBUGDATA
LDD_QUERY command 164
LDD, SET command 193
licensed documents vii
LINE NUMBERS, LIST command 137
LINE, AT command 58, 65
LINES, LIST command 138
LIST commands
LIST (blank) 125
LIST AT 125
restriction with playback 127
LIST CALLS 128
restriction with playback 128
LIST CURSOR (full-screen mode) 128
LIST expression 129
LIST FREQUENCY 132
LIST LAST 132
LIST MONITOR 134
LIST NAMES 134
LIST ON (PL/I) 136
LIST PROCEDURES 136
LIST REGISTERS
description 137
LIST STATEMENT NUMBERS 137
LIST STATEMENTS 138
LIST commands (continued)
LIST STORAGE description 139
summary 124
LIST, QUERY command 164
LIST, SET command 193
listing
SET DEFAULT LISTINGS command 183
LOAD command 140
load_module_name, description 14
load_spec, description 15
LOAD, AT command 58
LOADDEBUGDATA command 141
loaded modules
clearing 79
LOCATION, QUERY command 164
LOG NUMBERS, QUERY command 164
LOG NUMBERS, SET command 195
LOG, QUERY command 164
log, session
clearing 79
LOG, SET command 194
Logical Unit Name 7
LONGCUNAME, QUERY command 164
LookAt message retrieval tool viii
loops
for command (C and C++) 109
LU Name 7
lvalue for C 15

M
message retrieval tool, LookAt viii
MFI suboption of TEST run-time option 7
MONITOR COLUMN, QUERY command 164
MONITOR COLUMN, SET command 197
MONITOR command
description 142
MONITOR DATATYPE, QUERY command 164
MONITOR DATATYPE, SET command 197
MONITOR NUMBERS, QUERY command 164
MONITOR NUMBERS, SET command 197
MONITOR WRAP, QUERY command 164
MONITOR, SET command 197
MONITOR, LIST command 134
MONITOR, SET command 196
monitors
clearing 79
MOVE command (COBOL) 145
allowable moves 146
MSGID, QUERY command 164
MSGID, SET command 198

Index 371
NAMES, LIST command 134
NATIONAL LANGUAGE, QUERY command 164
NATIONAL LANGUAGE, SET command 198
NATLANG run-time option 2
NONE suboption of TEST run-time option 5
NOPROMPT suboption of TEST run-time option 6
NOTEST suboption of TEST run-time option 4
null command 149
NUMBERS, LIST STATEMENT command 137
NUMBERS, SET LOG command 195
NUMBERS, SET MONITOR command 197

OCCURRENCE, AT command 60
OFFSET, AT command (disassembly) 63
ON command (PL/I) 149
restriction with playback 150
ON, LIST command (PL/I) 136
restriction with playback 136
OPEN, WINDOW command 235
optimized COBOL commands not supported for
GOTO LABEL 113
JUMPTO 122
JUMPTO LABEL 124
OS/VS COBOL commands
assignment 36
commands not supported for
AT CALL 44, 54
OS/VS COBOL commands declarations 86
DO 98
IF 113

PACE, QUERY command 164
PACE, SET command 199
panel
Source Identification 152
PANEL command (full-screen mode) description 151
path point
differences between languages 64
PATH, AT command 63
PERFORM command (COBOL) 153
PFKEY, SET command 200
PFKEYS, QUERY command 164
PL/I
%INSTANCES built-in function 241
commands
ANALYZE 32
assignment 36
AT ALLOCATE 40
BEGIN 67
DECLARE 91

PL/I (continued)
commands (continued)
DO 99
IF 119
LIST ON 136
ON 149
SELECT 173
SET LONGCUNAME 195
HEX built-in function 240
RECURSION built-in function 242
SET WARNING 215
STORAGE built-in function 46
PL/I, definition of x
playback
available commands 157
PLAYBACK commands
BACKWARD 158
DISABLE 159
ENABLE 165
FORWARD 158
START 157
STOP 159
summary 155
PLAYBACK LOCATION, QUERY command 164
PLAYBACK, QUERY command 164
preferences_file_designator suboption of
TEST run-time option 7
prefix commands
AT 64
clear 83
description 159
disable 97
enable 102
query 166
show 220
problem determination
describing problems 360
determining business impact 359
submitting problems 360
PROCEDURE command 160
procedure, CALL 79
PROCEDURES, LIST command 136
PROGRAMMING LANGUAGE, QUERY command 164
PROGRAMMING LANGUAGE, SET command 201
PROMPT suboption of TEST run-time option 6
PROMPT, QUERY command 165
PROMPT, SET command 202

QUICK command 167
QUALIFY, RESET command 161
QUALIFY, QUERY command 165
QUERY command 161
QUERY prefix 166
QUIT command 166

range of statements, specifying 17
RECURSION built-in function 242

reference
C value 15
COBOL data name 16
COBOL special register 16
description 15
REFRESH, QUERY command 165
REGISTERS, LIST command 137
remote debug mode 8
tcpip_workstation_id suboption 8
tcpip& suboption 8
VADTCP& suboption 8
repeating breakpoints 40
RESTORE command 168
RESTORE, QUERY command 165
RESTORE, SET command 205
RETRIEVE command
description 169
return to point of execution 205
REWRITE, QUERY command 165
REWRITE, SET command 207
RUNTO command 170

SAVE, QUERY command 165
SAVE, SET command 207
SCREEN, QUERY command 165
SCREEN, SET command 210
SCROLL command
description 171
SCROLL DISPLAY, QUERY command 165
SCROLL, SET DEFAULT command 184
SELECT command (PL/I) 173
SEQUENCE, QUERY command 165
SEQUENCE, SET command 211
SET commands
SET ASSEMBLER 176
SET AUTOMONITOR 177
SET CHANGE 179
SET COLOR 180
SET command (COBOL)
description 216
SET COUNTRY 182
SET DBCS 182
SET DEFAULT LISTINGS 183
SET DEFAULT SCROLL
description 184
SET DEFAULT VIEW 185
SET DEFAULT WINDOW 185
SET DISASSEMBLY 186
SET DYNDEBUG 186
SET ECHO 188
SET EQUATE
description 188
SET EXECUTE 189
SET FREQUENCY 190
SET HISTORY 191
SET INTERCEPT
description 191
restriction with playback 192
SET KEYS 192
SET LDD 193
SET LIST TABULAR 193
SET LOG 194
SET LOG NUMBERS 195
Readers’ Comments — We’d Like to Hear from You

Debug Tool for z/OS
Debug Tool Utilities and Advanced Functions for z/OS
Reference and Messages
Version 7.1

Publication No. GC19-1073-02

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.
Submit your comments using one of these channels:
• Send your comments to the address on the reverse side of this form.
• Send your comments via e-mail to: COMMENTS@US.IBM.COM

If you would like a response from IBM, please fill in the following information:

Name

Address

Company or Organization

Phone No.

E-mail address