# IBM z/OS Debugger
# 15.0.5

*Reference and Messages*

**Note!**

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 535.

**Sixth Edition (May 2022)**

This edition applies to IBM® z/OS® Debugger, 15.0.5 (Program Number 5724-T07 with the PTF for PH44642), which supports the following compilers:

- Open Enterprise SDK for Go 1.17 (Program Number 5655-GOZ)
- z/OS XL C/C++ Version 2 (Program Number 5650-ZOS)
- C/C++ feature of z/OS Version 1 (Program Number 5694-A01)
- C/C++ feature of OS/390® (Program Number 5647-A01)
- C/C++ for MVS/ESA Version 3 (Program Number 5655-121)
- AD/Cycle C/370 Version 1 Release 2 (Program Number 5688-216)
- Enterprise COBOL for z/OS 6.1, 6.2, 6.3, and 6.4 (Program Number 5655-EC6)
- Enterprise COBOL for z/OS Version 5 (Program Number 5655-W32)
- Enterprise COBOL for z/OS Version 4 (Program Number 5655-S71)
- Enterprise COBOL for z/OS and OS/390 Version 3 (Program Number 5655-G53)
- COBOL for OS/390 & VM Version 2 (Program Number 5648-A25)
- COBOL for MVS™ & VM Version 1 Release 2 (Program Number 5688-197)
- COBOL/370 Version 1 Release 1 (Program Number 5688-197)
- VS COBOL II Version 1 Release 3 and Version 1 Release 4 (Program Numbers 5668-958, 5688-023) - with limitations
- OS/VS COBOL, Version 1 Release 2.4 (5740-CB1) - with limitations
- High Level Assembler for MVS & VM & VSE Version 1 Release 4, Version 1 Release 5, Version 1 Release 6 (Program Number 5696-234)
- Enterprise PL/I for z/OS 6.1 (Program Number 5655-PL6)
- Enterprise PL/I for z/OS Version 5 Release 1, Release 2, and Release 3 (Program Number 5655-PL5)
- Enterprise PL/I for z/OS Version 4 (Program Number 5655-W67)
- Enterprise PL/I for z/OS and OS/390 Version 3 (Program Number 5655-H31)
- VisualAge® PL/I for OS/390 Version 2 Release 2 (Program Number 5655-B22)
- PL/I for MVS & VM Version 1 Release 1 (Program Number 5688-235)
- OS PL/I Version 2 Release 1, Version 2 Release 2, Version 2 Release 3 (Program Numbers 5668-909, 5668-910) - with limitations

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

You can find out more about IBM z/OS Debugger by visiting the following IBM Web sites:

- IBM Debug for z/OS: https://www.ibm.com/products/debug-for-zos
- IBM Developer for z/OS: https://www.ibm.com/products/developer-for-zos
- IBM Z and Cloud Modernization Stack: https://www.ibm.com/docs/z-modernization-stack

# Contents

# About this document

z/OS Debugger 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. z/OS Debugger 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 z/OS Debugger, as well as the messages that you might see as you use z/OS Debugger. Many z/OS Debugger commands are similar to statements from the supported high-level languages (HLLs). This document also describes the TEST runtime option, syntax elements that are common for all commands, and syntax elements for expressions written in assembler, disassembly, and LangX COBOL.

## Who might use this document

This document is intended for programmers using z/OS Debugger 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.

z/OS Debugger runs on the z/OS operating system and supports the following subsystems:

- CICS®
- Db2®
- IMS
- JES batch
- TSO
- UNIX System Services in remote debug mode or full-screen mode using the Terminal Interface Manager 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, (GI10-8928), 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.

## 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 runtime option.
- Chapters 2, 3, 4, and 5 describe the complete syntax of the z/OS Debugger commands.
- Chapter 6 describes the complete syntax of the EQAOPTS commands.
- Chapters 7 and 8 describe the syntax of z/OS Debugger built-in functions and variables.
- Chapters 9, 10, 11, 12, 13, 14, 15, and 16 list all the messages that z/OS Debugger and other tools shipped with z/OS Debugger might display.
- Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517 has a list of commands that are supported in remote debug mode. This topic also contains instructions on how you can enter these commands.
- Appendix B, “Changes in behavior of some commands,” on page 523 describes changes to default behavior, including a comparison of the previous behavior and the new behavior, and with which version and release of z/OS Debugger the change was introduced.
- Appendix C, “Limitations of 64-bit support in Debug Tool compatibility mode,” on page 527 describes the limitations when you debug 64-bit programs in Debug Tool compatibility mode.
- Appendix D, “Support resources and problem solving information,” on page 529 describes the resources available to help you solve any problems you might have with z/OS Debugger.
- Appendix E, “Accessibility,” on page 533 describes the features and tools available to people with physical disabilities that help them use z/OS Debugger and z/OS Debugger documents.

The last several topics list notices, glossary of terms, and bibliography.

## Terms used in this document

Because of differing terminology among the various programming languages supported by z/OS Debugger, as well as differing terminology between platforms, a group of common terms is established. The following table lists these terms and their equivalency in each language.

| z/OS Debugger term | C and C++ equivalent | COBOL or LangX COBOL equivalent | PL/I equivalent | assembler |
|---|---|---|---|---|
| Compile unit | C and C++ source file | Program | • Program<br>• PL/I source file for Enterprise PL/I<br>• A package statement or the name of the main procedure for Enterprise PL/I[1] | CSECT |
| Block | Function or compound statement | Program, nested program, method, or PERFORM group of statements | Block | CSECT |
| Label | Label | Paragraph name or section name | Label | Label |

**Note:**

1. The PL/I program must be compiled with and run in one of the following environments:
   - Compiled with Enterprise PL/I for z/OS, Version 3.6 or later, and run with the following versions of Language Environment:
     - Language Environment Version 1.9, or later
     - Language Environment Version 1.6, Version 1.7, or Version 1.8, with the PTF for APAR PK33738 applied
   - Compiled with Enterprise PL/I for z/OS, Version 3.5, with the PTFs for APARs PK35230 and PK35489 applied and run with the following versions of Language Environment:
     - Language Environment Version 1.9, or later
     - Language Environment Version 1.6, Version 1.7, or Version 1.8, with the PTF for APAR PK33738 applied

z/OS Debugger provides facilities that apply only to programs compiled with specific levels of compilers. Because of this, this document 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 supported by z/OS Debugger except the COBOL compilers described in the term *LangX COBOL*.

**Disassembly or disassembled**
Refers to high-level language programs compiled without debug information or assembler programs without debug information. The debugging support z/OS Debugger 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.

**LangX COBOL**
Refers to any of the following COBOL programs that are supported through use of the EQALANGX debug file:

- Programs compiled using the IBM OS/VS COBOL compiler.
- Programs compiled using the VS COBOL II compiler with the NOTEST compiler option.
- Programs compiled using the Enterprise COBOL for z/OS V3 and V4 compiler with the NOTEST compiler option.

When 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.

VS COBOL II programs are non-Language Environment programs when you link them with the non-Language Environment library. VS COBOL II programs are Language Environment programs when you link them with the Language Environment library.

Enterprise COBOL programs are always Language Environment programs. Note that COBOL DLL's cannot be debugged as LangX COBOL programs.

Read the information regarding non-Language Environment programs for instructions on how to start z/OS Debugger and debug non-Language Environment COBOL programs, unless information specific to LangX COBOL 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.

# 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:

**Symbol**
**Definition**

►►—
Indicates the beginning of the syntax diagram.

—►
Indicates that the syntax diagram is continued to the next line.

►—
Indicates that the syntax is continued from the previous line.

—►◄
Indicates the end of the syntax diagram.

## 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.

**Item type**
**Definition**

**Required**
Required items are displayed on the main path of the horizontal line.

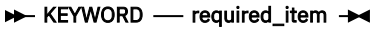
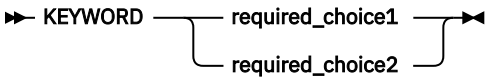
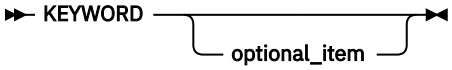
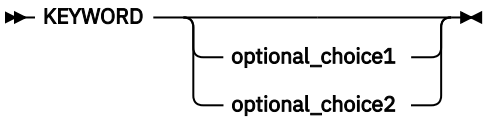
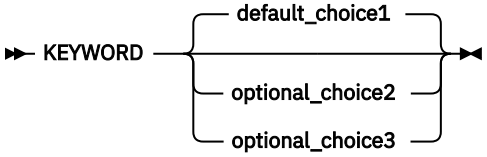
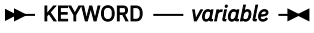
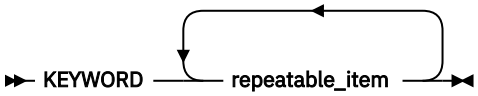
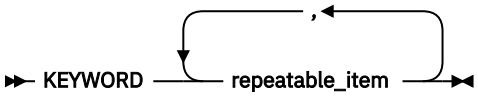
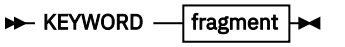
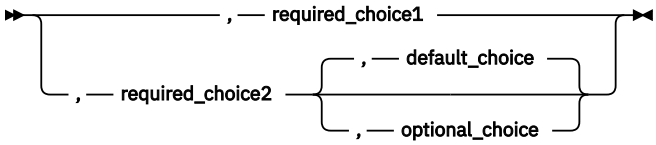
**Optional**
Optional items are displayed below the main path of the horizontal line.

**Default**
Default items are displayed above the main path of the horizontal line.

## Syntax examples

The following table provides syntax examples.

*Table 1. Syntax examples*

| Item | Syntax example |
|---|---|
| Required item.<br><br>Required items appear on the main path of the horizontal line. You must specify these items. |  |
| Required choice.<br><br>A required choice (two or more items) appears in a vertical stack on the main path of the horizontal line. You must choose one of the items in the stack. |  |
| Optional item.<br><br>Optional items appear below the main path of the horizontal line. |  |
| Optional choice.<br><br>An optional choice (two or more items) appears in a vertical stack below the main path of the horizontal line. You may choose one of the items in the stack. |  |
| Default.<br><br>Default items appear above the main path of the horizontal line. The remaining items (required or optional) appear on (required) or below (optional) the main path of the horizontal line. The following example displays a default with optional items. |  |
| Variable.<br><br>Variables appear in lowercase italics. They represent names or values. |  |
| Repeatable item.<br><br>An arrow returning to the left above the main path of the horizontal line indicates an item that can be repeated.<br><br>A character within the arrow means you must separate repeated items with that character.<br><br>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. |  <br>  |
| Fragment.<br><br>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. |  <br>**fragment**<br>  |

# How to provide 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 z/OS Debugger documentation, you can leave a comment in IBM Documentation:

- IBM Developer for z/OS and IBM Developer for z/OS Enterprise Edition: https://www.ibm.com/docs/en/developer-for-zos
- IBM Debug for z/OS: https://www.ibm.com/docs/debug-for-zos
- IBM Z and Cloud Modernization Stack: https://www.ibm.com/docs/z-modernization-stack

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

## 15.0.5

**z16, Compilers and CICS**

- Support is added for the new IBM z16 hardware.
- Support is added for CICS Transaction Server for z/OS 6.1.
- Support is added for the following compiler versions in both 31-bit mode and 64-bit mode:
  - Enterprise COBOL for z/OS 6.4
  - Enterprise PL/I for z/OS 6.1

**User-defined functions**

- User-defined functions are supported for programs compiled with Enterprise COBOL for z/OS 6.4.

**64-bit support**

- Debug Tool compatibility mode now supports delay debug.
- The following EQAOPTS commands are now supported:
  - DLAYDBG
  - DLAYDBGDSN
  - DLAYDBGTRC

  For the remaining limitations, see Appendix C, “Limitations of 64-bit support in Debug Tool compatibility mode,” on page 527.

**Interoperability**

- Interoperability is now supported between 31-bit and 64-bit COBOL programs. Use delay debug mode to improve efficiency.
- You can now debug 31-bit COBOL programs called from 64-bit Java programs. Delay debug mode is required when you debug this type of application.

  For more information, see "Using delay debug mode to delay starting of a debug session" in *IBM z/OS Debugger User's Guide*.

**Automatic Binary Optimizer for z/OS**

- Support is added for Automatic Binary Optimizer for z/OS 2.2.
- In Debug Tool compatibility mode, you can now debug programs compiled with Enterprise COBOL for z/OS Version 5 or later that have been optimized by Automatic Binary Optimizer for z/OS 2.2. The optimized programs can be debugged in the same manner as those that are not optimized.

**Code Coverage**

- You can now filter code coverage results for z/OS batch applications using existing JCL and z/OS UNIX applications by specifying a filter list file in the launch configuration. For more information, see "Filtering code coverage results during collection" in IBM Documentation.
- The module exclude list is deprecated and replaced by the filter list file.

**Debug profile**

- As a system administrator of Debug Profile Service, you can now use z/OS Debugger Profile Management to identify and delete CICS (DTCN) profiles that might interfere with z/OS system resources. For more information, see "Managing profiles with z/OS Debugger Profile Management" in *IBM z/OS Debugger Customization Guide*.

**Host configuration**

- When not run as a started task, Debug Manager now queries your security product for explicit permission to start. For more information, see "Starting Debug Manager as a user job" in *IBM z/OS Debugger Customization Guide*.

## 15.0.4

**IBM Z® Open Debug**

- You can now connect to a debug session and start debugging when you list debug sessions that are parked on a z/OS machine.
- With the Wazi for VS Code IDE, you can now use a single action to activate a debug profile, launch and debug an application.

**Code Coverage**

- You can now merge and export code coverage results on z/OS from command line into a single file of various formats with the `ccexport.sh` command. For more information, see "Merging and exporting code coverage results from z/OS" in IBM Documentation.
- Code Coverage Service can now be started as part of Remote Debug Service. For more information, see "Generating code coverage in headless mode using Remote Debug Service" in IBM Documentation.
- When started via the headless code coverage collector, Code Coverage Service now supports secured connections, and requires authentication.
- In the **Code Coverage Results** view, you can now add a secured Code Coverage Service result location (https). You can add and clear untrusted certificates in the CCS keystore file. For more information, see "Working with result locations" in IBM Documentation.
- The code coverage output location specified in the `o,output` parameter is ignored in the startup key unless you specify `-a,allowoutputlocation=TRUE` in the command line when you start the code coverage collector and use headless code coverage. For more information, see "Specifying code coverage options in the startup key" and "Starting and stopping the headless code coverage daemon" in IBM Documentation.
- The Code Coverage API documentation is updated from version 10.1.2 to 11.0.0.

**IBM Open Enterprise SDK for Go**

- In Debug Tool compatibility mode, you can now debug Go programs compiled with IBM Open Enterprise SDK for Go 1.17. For more information, see "Debugging programs compiled with IBM Open Enterprise SDK for Go" in *IBM z/OS Debugger User's Guide*.

**Decimal point**

- In Debug Tool compatibility mode, if the `DECIMAL-POINT IS COMMA` clause is specified in a COBOL program compiled with Enterprise COBOL for z/OS Version 6 Release 3 (UI78163) or later, the debugger displays decimals as commas in the **Variables** and **Monitors** views in Eclipse IDEs, and expressions accept commas, in addition to periods, as decimal points.

**Host configuration**

- You can now configure the `eqahcc.enc` file to start Code Coverage Service as part of Remote Debug Service. For more information, see the "Customizing with the sample job EQARMTSU" topic in *IBM z/OS Debugger Customization Guide*.
- With Debug Manager, you can leverage Dynamic Virtual IP Addressing (DVIPA) available in IBM Explorer for z/OS to concurrently run identical setups on different systems in your sysplex, and have TCP/IP, optionally with the help of WLM, distribute the client connections among these systems. Ensure that each Debug Manager has a unique external port per system and the port is explicitly defined in TCP/IP definitions. For more information, see the "Distributed Dynamic VIPA" section in *IBM z/OS Debugger Customization Guide*.

- MVS data set *userid*.EQATIOUT is no longer needed when you install and configure the IMS transaction isolation extension for Eclipse IDE users.

## 15.0.3

### z/OS 2.5

- Support is added for z/OS 2.5.

### IBM Open Enterprise SDK for Go

- In Debug Tool compatibility mode, you can now debug Go programs compiled with IBM Open Enterprise SDK for Go 1.16. For more information, see "Debugging programs compiled with IBM Open Enterprise SDK for Go" in *IBM z/OS Debugger User's Guide*.

### 64-bit support

- Debug Tool compatibility mode now supports playback for 64-bit COBOL programs. For the remaining limitations, see Appendix C, “Limitations of 64-bit support in Debug Tool compatibility mode,” on page 527.

### Source Level Code Coverage for COBOL

- When you start a code coverage session with the Eclipse IDE or headless code coverage, you can now choose to use the source listing. Source level code coverage offers direct mapping between code coverage entries and the program source, to exclude the need to post process the code coverage data. Source level code coverage improves integration with tools like ZUnit and SonarQube as part of an automated pipeline. For more information, see "How does z/OS Debugger locate COBOL source during code coverage" in *IBM z/OS Debugger User's Guide*.

### Code Coverage Service API

- Code Coverage Service (CCS) RESTful API is now available to enable custom extensions. For more information, see "Code Coverage Service RESTful API Documentation" in IBM Documentation.

### z/OS Debugger Profiles view

- Remote IMS Application with Isolation launch configurations have been replaced by the IMS Isolation debug profiles. All existing IMS launch configurations are automatically migrated to the **z/OS Debugger Profiles view**. You can create and activate IMS Isolation profiles in the view to debug and run code coverage for IMS transactions in private regions.

  To use this function, ensure that the system programmer installed and configured the IMS transaction isolation extension for the ADFz Common Components server. If you want to configure the region name for the private region, ask the system programmer to update z/OS Debugger to 15.0.3 or later, with the PTF for APAR PH41774 applied.

  **Note:** IMS Isolation profiles are only available in IBM Developer for z/OS Enterprise Edition.
- You can now view the Remote System Explorer z/OS connection status in the view. A new option **Refresh z/OS Connections** is provided in the view toolbar to establish all z/OS connections and synchronize the profiles.
- You can now duplicate the content from an existing debug profile to create a new one efficiently.
- Generic profiles might trigger z/OS Debugger unexpectedly and consume unnecessary resources. When you activate a generic profile, warnings are now displayed. You can choose to hide the warnings.
- In the Debug Profile Editor, you can now save a debug profile without activating it, and leave it for future use.

For more information, see "Managing debug profiles with the z/OS Debugger Profiles view" in IBM Documentation.

**Debug Profile Service**

- You now only need to expose one port to use Debug Profile Service. A new configuration switch is added to `eqaprof.env` to select whether to use secure HTTP protocol. For more information, see "Customizing with the sample job EQAPRFSU" in *IBM z/OS Debugger Customization Guide*.

**z/OS Debugger commands**

- The following commands are now supported in Debug Tool compatibility mode for remote debugging:
  - STEP
  - GO
  - RUNTO
  - JUMPTO
  - COMMENT

**Dark theme**

- Dark theme is now supported for remote debugger in the Eclipse IDE.

**Message EQA9924U**

- Message ICH408I is now issued in the console with EQA9924U to provide you with more information to address the issue.

## 15.0.2

**IBM Z Open Debug**

- You can now debug High Level Assembler (HLASM) z/OS programs with IBM Z Open Debug.
- For Wazi for Dev Spaces, the log files are now in `/projects/.debug/logs`.

**Code Coverage**

- In the **Code Coverage Results** view, you can now export code coverage results in Cobertura format. For more information, see "Exporting code coverage results in Cobertura format" in IBM Documentation.
- You can now specify parameters in the startup key to generate code coverage results in Cobertura and SonarQube formats. In addition, short parameters values `-e,exportertype=SQ|PDF|COB` are added for you to use both in the startup key and in the headless code coverage daemon. For more information, see "Specifying code coverage options in the startup key" and "Starting and stopping the headless code coverage daemon" in IBM Documentation.
- When you view code coverage results in an editor, you can now see a code coverage summary of the included files for PL/I source files with `%INCLUDE` statements. For more information, see "Viewing code coverage results in an editor" in IBM Documentation.

**z/OS Batch Applications launches**

- In the **Remote Systems** or **z/OS Projects** view, or when you are editing the JCL source in the editor, after you choose **Debug As** or **Code Coverage As** from the menu, the following options are available:
  - **z/OS Batch Application**: Launch a debug or code coverage session without a debug profile.
  - **z/OS Batch Application with a debug profile**: Launch a debug or code coverage session with a debug profile.
  - **z/OS Batch Application ...**: Create a launch configuration to launch a debug or code coverage session.

  For more information, see "Launching a debug session for z/OS batch applications using existing JCL" in IBM Documentation.

**IBM z/OS Debugger JCL Wizard**

- The **Program/Procedure Selection List** panel is updated to include procedures, in addition to programs. Selecting a procedure will provide a panel to enter the procedure step override for the DD statements generated. The After (A) and Before (B) line commands are no longer required.
- You can now select **SVC screening** to enable SVC screening for batch non-Language Environment programs.
- You can now select **Intercept on** to show COBOL DISPLAY statements on the IBM z/OS Debugger log or Debug Console in the Eclipse IDE.
- Error messages are improved.
- The END command (**PF3**) in the **Program/Procedure Selection List** panel is modified to cancel the request. Previously, selecting **PF3** would not exit the panel.
- When the process is completed, the cursor is now placed on the command line.
- The **z/OS Debugger LDD Generation for Non-LE Programs** panel is now populated with the initial program name and subprograms selected in the **Request AT ENTRY Sub-Program Breakpoints** panel. Program names provided in this panel can be modified.
- After you select **Code Coverage** from the parameters selection panel, the EQAXOPT lines are generated to specify `CCPROGSELECTDSN`, `CCOUTPUTDSN` and `CCOUTPUTDSNALLOC`, if the `CODE_COVERAGE_SETUP` value is configured to YES in the EQAJCL REXX procedure.
- Previously the wizard would verify the program source members identified in the **z/OS Debugger LDD Generation for Non-LE Programs** panel with each library identified by the **z/OS Debugger Debug Libraries** panel to verify that the members are present. This function is removed because z/OS Debugger now flags any such members in the IBM z/OS Debugger log or Debug Console in the Eclipse IDE when the LDD command is entered.

For more information, see "IBM z/OS Debugger JCL Wizard" in the *IBM z/OS Debugger User's Guide*

**AT LABEL * command**

- For Enterprise COBOL for z/OS Version 5 and later, `AT LABEL *` now highlights the labels similar to statement breakpoints.
- You can now use PF6 or `AT LINE` to remove or add a single global label hook if `AT LABEL *` was issued. To disable this functionality, use `DISABLE AT LABEL *`.
- If you issue `AT LABEL *` again or use `ENABLE AT LABEL *`, the global label hooks are reset. The hooks that you removed are added back.

For more information, see AT LABEL command.

## 15.0.1

**64-bit support**

- Debug Tool compatibility mode now supports the following features:
  - Code coverage
  - Source entry breakpoints
  - CEETEST

  For the remaining limitations, see Appendix C, “Limitations of 64-bit support in Debug Tool compatibility mode,” on page 527.

**Code Coverage**

- With Concurrent Debug and Code Coverage, you can run code coverage collection in parallel with the active debug session in the Eclipse IDE. The code coverage data is collected during the debug run, and code coverage annotations are displayed and updated in the debug editor. Concurrent Debug and Code Coverage requires z/OS Debugger 15.0.1 or later. For more information, see the "Generating code coverage in a remote debug session" topic in IBM Documentation.

- Headless code coverage report can now be exported with a Cobertura exporter. For more information, see the "Starting and stopping the headless code coverage daemon" topic in IBM Documentation.
- Headless code coverage collector now supports filtering of module, compiler units, and files. For more information, see the "Filtering code coverage results" topic in IBM Documentation.

**Debug Profile Editor**

- In the Debug Profile Editor of the Eclipse IDE, new key bindings are available to show the error tooltip and the overall error summary. For more information, see the "Debug profile key bindings" topic in IBM Documentation.

**Debug Profile Service**

- As an alternative of a keystore file, you can now use a RACF managed key ring to enable secure communication with Debug Profile Service. For more information, see the "Enabling secure communication with a RACF managed key ring" section in *IBM z/OS Debugger Customization Guide*.
- A new optional HOST attribute is added to the CICS region configuration. For more information, see the instructions in the `/etc/debug/dtcn.ports` sample configuration file.
- The Debug Profile Service API now provides more detailed diagnostic messages when authentication fails.

**IBM Z Open Debug**

- Log files can now be found in the user's home directory.

**CICS trace entries**

- A new parameter, DNT, is added to the CICS startup parameter INITPARM to support disabling generation of z/OS Debugger trace entries. For more information, see the "Adding support for debugging under CICS" topic in *IBM z/OS Debugger Customization Guide*.

## 15.0.0

**64-bit support**

- Debug Tool compatibility mode now supports the following compiler features:
  - The 64-bit COBOL feature of z/OS for COBOL V6.3 and later
  - The 64-bit C/C++ feature of z/OS

  For the limitations, see Appendix C, “Limitations of 64-bit support in Debug Tool compatibility mode,” on page 527.

  The PTFs for z/OS Language Environment APARs PH26071 and PH28997 are required for this support.

**IBM Z Open Debug**

- IBM Z Open Debug is now also available with the Wazi for Dev Spaces IDE, in addition to the Wazi for VS Code IDE. Both IDEs are offered in IBM Wazi Developer for Red Hat CodeReady Workspaces and IBM Developer for z/OS Enterprise Edition. For a comparison of features provided in different products and IDEs, see Overview of IBM z/OS Debugger.
- You can now specify `TEST(,,,RDS:*)` for the TEST runtime option to start a debug session using Remote Debug Service for Wazi for VS Code or Wazi for Dev Spaces.

**Code Coverage**

- Headless code coverage for z/OS is now included with IBM Debug for z/OS. Use the headless code coverage collector to generate code coverage results of tests that are run as part of your DEVOPS pipeline. For more information, see the "Generating code coverage in headless mode using a daemon" section in IBM Documentation.
- Single letter parameters are now supported in the headless code coverage collector command line and in `EQA_STARTUP_KEY` when you use JCL. For more information, see topics "Starting and

stopping the headless code coverage daemon" and "Specifying code coverage options in the startup key" in IBM Knowledge Center.

- Support is added for PL/I programs compiled with `LISTVIEW(SOURCE)` to generate code coverage results for main program and all `%INCLUDE` files. For more information, see the "Supported compilers and options for code coverage in Debug Tool compatibility mode" topic in IBM Knowledge Center.
- The **Code Coverage Results** view of the Eclipse IDE now supports CCS result locations. You can add a CCS result location which collects and retrieves code coverage data by using RESTful API, and interact with the results under the CCS result location in the same way as locally stored results. CCS result locations require Headless Code Coverage 15.0.0 or later. For more information, see the "Viewing code coverage results in the Code Coverage Results view" topic in IBM Knowledge Center.
- You can now also use Remote Debug Service to collect code coverage results similar to the headless code coverage collector for IBM Wazi Developer for Red Hat CodeReady Workspaces or IBM Developer for z/OS Enterprise Edition. For more information, see the "Generating code coverage in headless mode using Remote Debug Service" topic in IBM Knowledge Center.

**Debug Profile Editor**

- In the Debug Profile Editor of the Eclipse IDE, you can now use the quick outline to navigate to a field. For more information, see the "Quick outline for the Debug Profile Editor" topic in IBM Knowledge Center.

**z/OS XL C/C++**

- Support is added for `DEBUG(NOFILE)`. For more information, see topics "Choosing DEBUG compiler suboptions for C programs" and "Choosing DEBUG compiler suboptions for C++ programs" in *IBM z/OS Debugger User's Guide*.

**Debug Tool Plug-ins**

- The following Debug Tool plug-ins of the Eclipse IDE are deprecated and will be removed in the next release:
  - DTCN Profile Manager plug-in
  - DTSP Profile Manager plug-in
  - Instrument JCL for Debug Tool Debugging plug-in
  - Debug Tool Code Coverage plug-in
  - Load Module Analyzer plug-in

  You can use the **z/OS Debugger Profiles** view to create and manage debug profiles, z/OS batch applications launches to dynamically instrument and submit JCL, and the **Code Coverage Results** view to work with compiled code coverage results. For more information, see the following topics in IBM Documentation: Managing debug profiles with the z/OS Debugger Profiles view, Launching a debug session for z/OS batch applications using existing JCL, and Viewing code coverage results in the Code Coverage Results view.

**Load Module Analyzer**

- The Load Module Analyzer is deprecated and will be removed in a future version.

**Host configuration**

- Remote Debug Service can now be configured to collect headless code coverage. For more information, see the "Adding support for Remote Debug Service" section in *IBM z/OS Debugger Customization Guide*.
- The record size for the DTCN VSAM file is increased to 3000 bytes. To use the DTCN VSAM repository with z/OS Debugger 15.0, create a new file using the `SEQASAMP(EQAWCRVS)` sample JCL. You can also convert your existing VSAM file to the new record size and format using the EQADPCNV utility. For more information, see the "Migrating a debug profiles VSAM file from an earlier release" topic in *IBM z/OS Debugger Customization Guide*.

- The IMS Transaction Isolation Facility is enhanced to utilize type 2 IMS commands for retrieving information on transactions, in cases where the type 1 commands that are normally used are disallowed. For more information, see the "Scenario F: Enabling the Transaction Isolation Facility" topic in *IBM z/OS Debugger Customization Guide*.

# Overview of IBM z/OS Debugger

IBM z/OS Debugger is the next iteration of IBM debug technology on IBM Z and consolidates the IBM Integrated Debugger and IBM Debug Tool engines into one unified technology. IBM z/OS Debugger is progressing towards one remote debug mode based on Debug Tool compatibility mode. In support of this direction, Debug Tool compatibility mode, when available in the user interface, is selected by default for V14.1.2 or later.

IBM z/OS Debugger is a host component that supports various debug interfaces, like the Eclipse and Visual Studio Code IDEs. z/OS Debugger and the supported debug interfaces are provided with the following products:

**IBM Developer for z/OS Enterprise Edition**

This product is included in IBM Application Delivery Foundation for z/OS. IBM Developer for z/OS Enterprise Edition provides all the debug features.

IBM Developer for z/OS Enterprise Edition currently provides debug functions in the following IDEs:

- IBM Developer for z/OS Eclipse
- Wazi for Dev Spaces, through IBM Z Open Debug
- Wazi for VS Code, through IBM Z Open Debug

See Table 3 on page xxvii for the debug features supported in different IDEs.

**IBM Developer for z/OS**

IBM Developer for z/OS is a subset of IBM Developer for z/OS Enterprise Edition. IBM Developer for z/OS, previously known as IBM Developer for z Systems or IBM Rational® Developer for z Systems®, is an Eclipse-based integrated development environment for creating and maintaining z/OS applications efficiently.

IBM Developer for z/OS includes all enhancements in IBM Developer for z/OS Enterprise Edition except for the debug features noted in Table 2 on page xxvi.

**IBM Debug for z/OS**

IBM Debug for z/OS is a subset of IBM Developer for z/OS Enterprise Edition. IBM Debug for z/OS focuses on debugging solutions for z/OS application developers. See Table 2 on page xxvi for the debug features supported.

IBM Debug for z/OS does not provide advanced developer features that are available in IBM Developer for z/OS Enterprise Edition.

For information about how to install the IBM Debug for z/OS Eclipse IDE, see Installation of IBM Developer for z Systems and IBM Debug for z Systems (https://developer.ibm.com/mainframe/2016/12/02/installation-of-ibm-developer-for-z-systems-and-ibm-debug-for-z-systems/).

**IBM Z and Cloud Modernization Stack**

IBM Z and Cloud Modernization Stack brings together component capabilities from IBM Z into an integrated platform that is optimized for Red Hat OpenShift Container Platform. With this solution, you can analyze the impact of application changes on z/OS, create and deploy APIs for z/OS applications, work on z/OS applications with cloud native tools, and standardize ID automation for z/OS. Starting from 2.0, Wazi Code is delivered in IBM Z and Cloud Modernization Stack. Wazi Code 1.x is still available in IBM Wazi Developer for Red Hat CodeReady Workspaces.

The debug functions are available in the IDEs provided with Wazi Code:

- Wazi for Dev Spaces, through IBM Z Open Debug
- Wazi for VS Code, through IBM Z Open Debug
- Wazi for Eclipse

See Table 2 on page xxvi and Table 3 on page xxvii for the debug features supported in the product and different IDEs.

Table 2 on page xxvi maps out the features that differ in products. Not all the available features are listed. To find the features available in different remote IDEs, see Table 3 on page xxvii.

*Table 2. Debug feature comparison*

| | **IBM Debug for z/OS** | **IBM Developer for z/OS** | **IBM Developer for z/OS Enterprise Edition** | **IBM Z and Cloud Modernization Stack (Wazi Code)** |
|---|---|---|---|---|
| **Main features** | | | | |
| 3270 interface, including z/OS Debugger Utilities | √ | | √ | |
| Eclipse IDE, see Table 3 on page xxvii for feature details.[1] | √ | √ | √ | √ |
| IBM Z Open Debug provided with the Wazi for Dev Spaces IDE, see Table 3 on page xxvii for feature details.[1] | | | √ | √ |
| IBM Z Open Debug provided with the Wazi for VS Code IDE, see Table 3 on page xxvii for feature details.[1] | | | √ | √ |
| **Code Coverage features** | | | | |
| Compiled Language Code Coverage[2] | √ | √ [3] | √ | |
| Headless Code Coverage | √ | √ | √ | |
| Java™ Code Coverage | | √ | √ | |
| ZUnit Code Coverage[4] | | √ | √ | |
| z/OS Debugger Code Coverage (3270 and remote interfaces) [5] | √ | | √ | |
| **3270 features** | | | | |
| z/OS Debugger full screen, batch or line mode | √ | | √ | |
| IMS Isolation support | √ | | √ | |

| *Table 2. Debug feature comparison (continued)* | | | | |
|---|---|---|---|---|
| | **IBM Debug for z/OS** | **IBM Developer for z/OS** | **IBM Developer for z/OS Enterprise Edition** | **IBM Z and Cloud Modernization Stack (Wazi Code)** |
| **Compiler support features** | | | | |
| Assembler support: Create EQALANGX files | √ | √ | √ | |
| Assembler support: Debugging [6] | √ | √ | √[7] | √[7] |
| LANGX COBOL support [8] | √ | √ | √ | |
| Support for Automatic Binary Optimizer (ABO) | √ | √ | √ | |
| Load Module Analyzer[9] | √ | | √ | |

**Notes:**

1. The following features are supported only in remote debug mode:
   - Support for 64-bit COBOL feature of z/OS for COBOL V6.3 and later
   - Support for 64-bit Enterprise PL/I for z/OS Version 5
   - Support for 64-bit C/C++ feature of z/OS
   - Support for IBM Open Enterprise SDK for Go 1.16.
2. Code coverage does not support Go programs.
3. IBM Developer for z/OS includes z/OS Debugger remote debug and compiled code coverage Eclipse interface, but does not include z/OS Debugger Code Coverage.
4. ZUnit Code Coverage is only supported in Debug Tool compatibility mode.
5. z/OS Debugger Code Coverage can only be enabled in the 3270 interface.
6. Debugging assembler requires that you have EQALANGX files that have been created via ADFz Common Components or a product that ships the ADFz Common Components.
7. This feature is only available with the Eclipse IDE.
8. LANGX COBOL refers to any of the following programs:
   - A program compiled with the IBM OS/VS COBOL compiler.
   - A program compiled with the IBM VS COBOL II compiler with the NOTEST compiler option.
   - A program compiled with the IBM Enterprise COBOL for z/OS Version 3 or Version 4 compiler with the NOTEST compiler option.
9. Load Module Analyzer is deprecated and will be removed in a future version.

| *Table 3. Remote IDE debug feature comparison* | | |
|---|---|---|
| **Feature** | **Eclipse-based debug interface** | **IBM Z Open Debug [1,10]** |
| Debug Tool compatibility mode[2] | √ | √ |
| Standard mode[3,10] | √[4] | |

| *Table 3. Remote IDE debug feature comparison (continued)* | | |
|---|---|---|
| **Feature** | **Eclipse-based debug interface** | **IBM Z Open Debug [1,10]** |
| Integration with Language Editors[10] | • COBOL Editor[5]<br>• PL/I Editor[5]<br>• Remote C/C++ Editor[4,5]<br>• System z LPEX Editor[4,5] | • Z Open Editor |
| Visual Debug | √[5,10] | |
| Debugging ZUnit tests | √[6,10] | |
| Debug profile management | √[4,10] | √ |
| IMS Isolation UI | √[7] | |
| Integration with CICS Explorer views | √ [4,5] | |
| Integration with property groups | √[5,10] | |
| Team Debug support | √[4,5] | |
| Integrated launch[10] | • z/OS UNIX Application launch configuration<br>• z/OS Batch Application using existing JCL<br>• z/OS Batch Application using a property group[5] | |
| Debug Tool Plug-ins | √[4, 8] | |
| Modules | √ | |
| Memory | √ | |
| **Program navigation** | | |
| Step over/Next | √ | √ |
| Step into/Step in | √ | √ |
| Step return/Step out | √ | √ |
| Jump to location | √[10] | |
| Run to location/Run to cursor | √[10] | √ |
| Resume/Continue | √ | √ |
| Terminate | √ | √ |
| Animated step | √ | |
| Playback | √[10] | |
| **Breakpoints** | | |
| Line/statement breakpoints | √ | √ |
| Entry breakpoints | √ | |
| Source entry breakpoints | √[10] | |
| Event breakpoint | √[10] | |

| *Table 3. Remote IDE debug feature comparison (continued)* | | |
|---|---|---|
| **Feature** | **Eclipse-based debug interface** | **IBM Z Open Debug [1,10]** |
| Address breakpoint | √[10] | |
| Watch breakpoint | √[10] | |
| **Variables & Registers** | | |
| Variables | √ | √ |
| Registers | √ | √[9] |
| Modifying variable and register values | √ | √ |
| Setting variable filter | √ | |
| Changing variable representation | √ | |
| Dereferencing variables | √ | |
| Displaying in memory view | √ | |
| **Monitors** | | |
| Displaying monitor | √ | √ |
| Modifying monitor value | √ | |
| Changing variable representation | √ | |
| Dereferencing variables | √ | |
| **Debug Console** | | |
| Evaluating variables and expressions | | √ |
| z/OS Debugger commands | √[10] | |

**Notes:**

1. IBM Z Open Debug is provided with Wazi for Dev Spaces and Wazi for VS Code.
2. Debug Tool compatibility mode does not support 64-bit Enterprise PL/I for z/OS Version 5.
3. Standard mode does not support 64-bit COBOL feature of z/OS for COBOL V6.3 and later. Source view for COBOL V6.2 and later is supported only in standard mode.
4. This feature is not available in Wazi for Eclipse.
5. This feature is not available in IBM Debug for z/OS.
6. Debugging ZUnit tests is only supported in Debug Tool compatibility mode.
7. This feature is only available in IBM Developer for z/OS Enterprise Edition.
8. IBM Developer for z/OS includes Debug Tool plug-ins, but does not include Load Module Analyzer and z/OS Debugger Code Coverage 3270 interfaces.
9. Registers are available in the **Variables** view.
10. Programs compiled with IBM Open Enterprise SDK for Go are not supported.

# Chapter 1. z/OS Debugger runtime options

This topic describes the runtime options that you can use to control the operation of z/OS Debugger.

"Table 10" in the IBM z/OS Debugger User's Guide describes most of the methods you can use to specify the TEST runtime options. Use that table with the information in the topic "Planning your debug session" in IBM z/OS Debugger User's Guide to select the method that works best for your site.

Some methods use the standard Language Environment runtime options. Other methods use z/OS Debugger 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 runtime 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 runtime options in any order.
- Separate them with either blanks or commas.
- Separate all runtime 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.

When you specify runtime options for a non-Language Environment program by using EQANMDBG under z/OS batch or TSO, z/OS Debugger 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 runtime options in any order following the name of the program to be debugged.
- Separate all options with commas.
- Separate the runtime options from user-program options with a slash ('/'). If you do not specify any runtime 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 topics for more information related to the material discussed in this topic.

**Related tasks**
"Planning your debug session" in IBM z/OS Debugger User's Guide

**Related references**
z/OS Language Environment Programming Reference

## Non-Language Environment positional parameter

If you use EQANMDBG to start z/OS Debugger 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 z/OS Debugger keyword runtime 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.

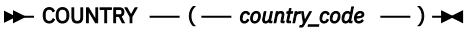
Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
"Planning your debug session" in *IBM z/OS Debugger User's Guide*.

## COUNTRY runtime option

Use the COUNTRY option to specify the country code to be used by z/OS Debugger. The default is always US.

The syntax for this option is:




***country_code***
A valid country code, one of:

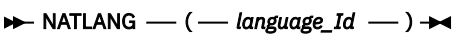
**US**
United States of America

**JP**
Japan

## NATLANG runtime option

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

The syntax for this option is:




***language_Id***
A valid national language identifier, one of:

**ENU**
English

**UEN**
Upper-case English

**JPN**
Japanese

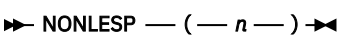
**KOR**
Korean

If you set NATLANG to JPN or KOR and you are using full-screen mode, enter the SET DBCS ON command so that z/OS Debugger displays messages in the correct format.

## NONLESP runtime option

Use the NONLESP option to direct z/OS Debugger to use a different storage subpool for its storage, in cases where the program being debugged does a FREEMAIN of subpool 1 (where z/OS Debugger places its data by default).

The syntax for this option is:




***n***
An integer with a value between 2 and 127

## TEST runtime option

The TEST runtime option gives control of your program to z/OS Debugger.

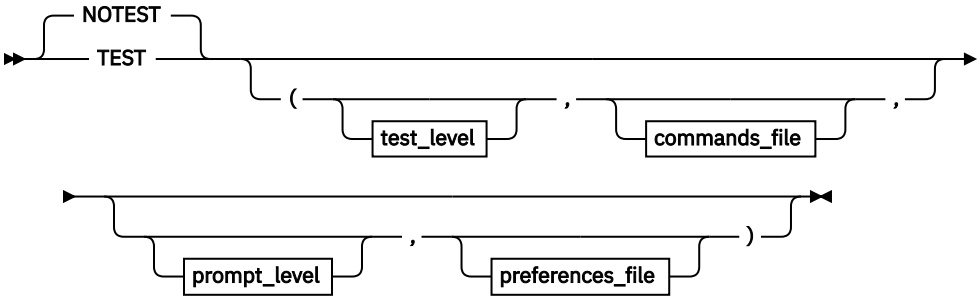
This topic describes the TEST runtime option and its suboptions. The suboptions of the TEST runtime option control how, when, and where z/OS Debugger gains control of your program. For a description of how to specify the TEST runtime option, refer to "Planning your debug session" in the IBM z/OS Debugger User's Guide.

## Syntax of the TEST runtime option

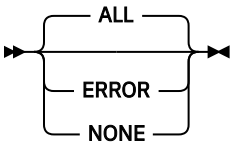
For examples of using the TEST runtime option to illustrate runtime options available for your programs, see the "Example: TEST runtime options" topic in *IBM z/OS Debugger User's Guide*.

You can combine any of the suboptions for the TEST runtime 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.

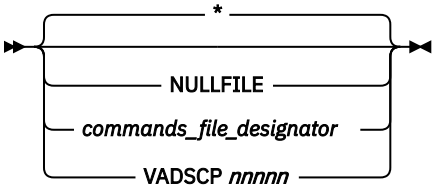
The syntax for this option is:



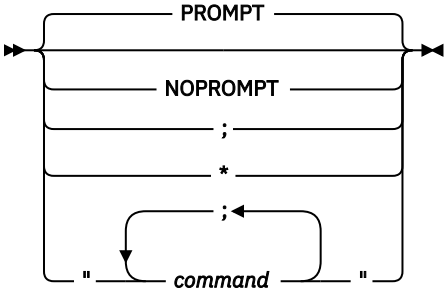

**test_level**



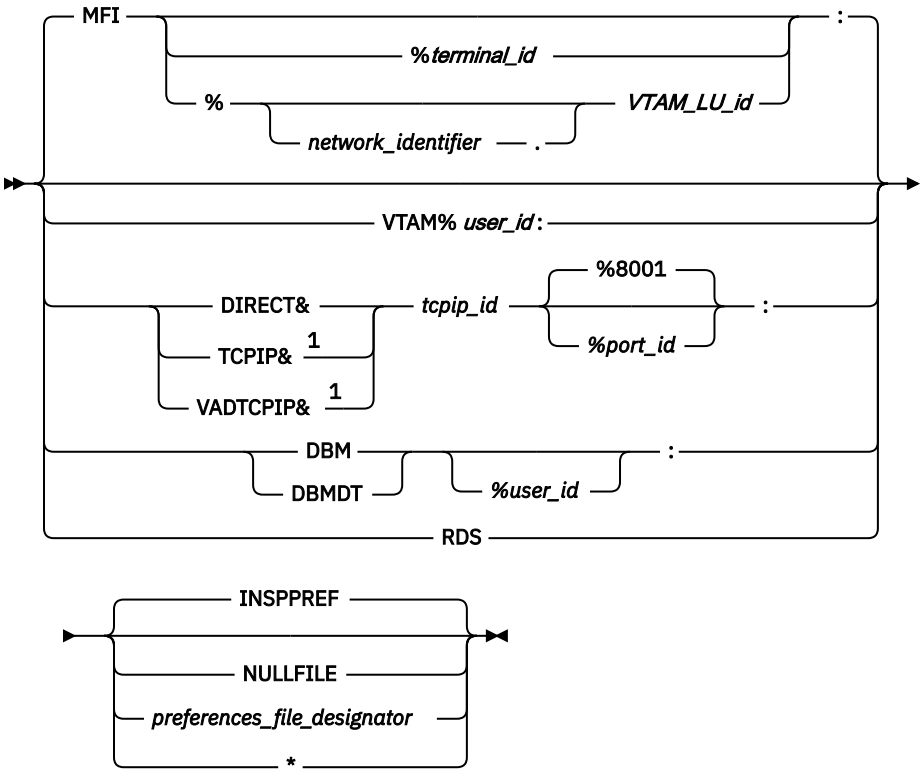

**commands_file**




**prompt_level**




**preferences_file**




Notes:

[1] Specifies remote debug mode. 64-bit COBOL, PL/I, and C/C++ programs are supported only in remote debug mode.

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

**NOTEST**
Specifies that z/OS Debugger is not started at program initialization. However, Starting z/OS Debugger 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 z/OS Debugger is started.

**TEST**
Specifies that z/OS Debugger is given control according to the specified suboptions. The TEST suboptions supplied are used if z/OS Debugger is started with CEETEST, PLITEST, or `__ctest()`.

If z/OS Debugger 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 z/OS Debugger.

**test_level**:

**ALL (or *blank*)**
Specifies that the occurrence of an attention interrupt, ABEND of a program, or any program or Language Environment condition of Severity 1 and above causes z/OS Debugger to gain control, regardless of whether a breakpoint is defined for that condition.

When a FINISH, CEE066 or CEE067 thread termination condition is raised by Language Environment, z/OS Debugger can be prevented from stopping at this condition by specifying the EQAOPTS THREADTERMCOND command. You or your system administrator can specify this command by creating an EQAOPTS load module or providing the command at run time.

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, z/OS Debugger does the following:

- In full-screen mode, z/OS Debugger reads commands from a commands file (if it exists and is available) or prompts you for commands.
- In batch mode, z/OS Debugger reads commands from the commands file. If none is available, the program runs uninterrupted.

**ERROR**
Specifies that only the following conditions cause z/OS Debugger 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, z/OS Debugger reads commands from a commands file or prompts you for them in interactive mode.

**NONE**
Specifies that z/OS Debugger 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 z/OS Debugger to gain control unless z/OS Debugger was started. To change the TEST level after you start your debug session, use the SET TEST command.

**commands_file**:

*** (or *blank*)**
Indicates that you did not supply a commands file.

In the following situation, z/OS Debugger reads commands from a default user commands file:

- You or your site specify a default naming pattern, through the EQAOPTS `COMMANDSDSN` command, identifying a user commands file.
- The user commands file exists.
- The user commands file contains a member with a name that matches the initial load module name of the first enclave.

If you or your site do not specify the name of a default user commands file or that file does not exist, and you are debugging in line mode, z/OS Debugger reads commands from the terminal.

To learn how to supply the EQAOPTS `COMMANDSDSN` command, see Chapter 6, “EQAOPTS commands,” on page 287.

**NULLFILE**
Indicates that you did not supply a commands file and z/OS Debugger ignores any specification of the EQAOPTS `COMMANDSDSN` command. If you are debugging in line mode, z/OS Debugger reads commands from the terminal.

***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.

If the designation contains non-alphanumeric characters (for example, a parenthesis), the designation must be enclosed in either quotation marks (") or apostrophes ('). However, when a data set name is enclosed in quotation marks or apostrophes, z/OS Debugger 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. z/OS Debugger 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 z/OS Debugger session as the commands file for a subsequent z/OS Debugger session.

The primary commands file is shared across multiple enclaves.

**VADSCP*nnnnn***
Specifies a CCSID (Coded Character Set Identifiers) to use when you are debugging programs in remote debug mode and the source or compiler use a code page other than 037.

If your C/C++ source contains square brackets or other special characters, you might need to specify the VADSCP*nnnnn* suboption to override the z/OS Debugger default code page (037). Consult with your system programmer to determine if he implemented the EQAOPTS CODEPAGE command to specify a code page of 1047. If not, check the code page specified when you compiled your source. The C/C++ compiler uses a default code page of 1047 if you do not explicitly specify one. If the code page used is 1047 or a code page other than 037, you need to specify the VADSCP*nnnnn* suboption specifying that code page.

The following examples show how to use VADSCP*nnnnn*:

- 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:*)
```

If a EQAOPTS CODEPAGE command exists, the code page specified in the EQAOPTS CODEPAGE command overrides the CCSID specified in VADSCP*nnnnn*.

If neither the EQAOPTS CODEPAGE command or the VADSCP*nnnnn* option are specified, the default code page is US code page (037).

**prompt_level**:

**PROMPT (or *;* or *blank*)**
Indicates that you want z/OS Debugger 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 z/OS Debugger started immediately after Language Environment initialization. Instead, your application begins running. When z/OS Debugger 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 z/OS Debugger.

***command***
One or more valid z/OS Debugger commands. z/OS Debugger 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 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 z/OS Debugger 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 z/OS Debugger screen output during dual terminal session. The corresponding terminal should be in service and acquired, ready to receive z/OS Debugger-related I/O.

***network_identifier* (full-screen mode using a dedicated terminal without Terminal Interface Manager 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 using a dedicated terminal without Terminal Interface Manager only)**
Specifies up to an eight-character VTAM® logical unit (LU) identifier for a terminal used in full-screen mode using a dedicated terminal without Terminal Interface Manager. 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.

**VTAM (full-screen mode using the Terminal Interface Manager only)**
Specifies z/OS Debugger should be started in full-screen mode using the Terminal Interface Manager for your debug sessions and that you have used the z/OS Debugger Terminal Interface Manager.

***user_id* (full-screen mode using the Terminal Interface Manager only)**
Specifies the user ID that you used to log on to the z/OS Debugger Terminal Interface Manager. Contact your system programmer to determine how to access this type of terminal at your site.

**INSPPREF (or *blank*)**
The default DD name, supplied by z/OS Debugger, for the preferences file.

In the following situation, z/OS Debugger reads commands from a default user preferences file:

- You specify INSPPREF or leave it blank, but do not allocate the DD name.
- You or your site specify a default naming pattern, through the EQAOPTS PREFERENCESDSN command, identifying a user preferences file.
- The user preferences file exists.

Any preferences file you or your site specifies to z/OS Debugger becomes the first source of z/OS Debugger commands after z/OS Debugger is started. Use preferences files to set up the z/OS Debugger environment; for example, PF key assignments or screen layout.

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

This file is read the first time z/OS Debugger is started and must contain a sequence of z/OS Debugger commands to be processed.

The designation can be either a DD name or a data set name. z/OS Debugger uses the following procedure to determine if the designation is a DD name or data set name:

- If the designation does not contain periods (.), z/OS Debugger considers it a DD name.
- Otherwise, if you are running under CICS, z/OS Debugger considers it a fully-qualified data set name.
- Otherwise, z/OS Debugger 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 z/OS Debugger to interpret the data set name as a fully-qualified name, put a minus sign (-) in front of the name. In this case, z/OS Debugger does 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 quotation marks (") or apostrophes ('). However, when a data set name is enclosed in quotation marks or apostrophes, z/OS Debugger 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 you did not supply a preferences file.

If you or your site specifies a naming pattern, through the EQAOPTS PREFERENCESDSN command, identifying a user preferences file, z/OS Debugger reads commands from that file.

To learn how to supply the EQAOPTS PREFERENCESDSN command, see Chapter 6, “EQAOPTS commands,” on page 287.

**NULLFILE**

Indicates that you did not supply a preferences file and z/OS Debugger ignores any specification of the EQAOPTS PREFERENCESDSN command.

The following TEST suboptions are for remote debug mode and code coverage:

**DIRECT&, TCPIP&, or VADTCPIP&**

Specifies that z/OS Debugger starts in remote debug mode with a client.

Use DIRECT& to start the debugger in standard mode. Use TCPIP& or VADTCPIP& to start the debugger in Debug Tool compatibility mode.

**Notes:**

1. IBM Z and Cloud Modernization Stack (Wazi Code) can only be used in Debug Tool compatibility mode.
2. Standard mode does not support commands files or preferences files. If they are specified, they are ignored.
3. Debug Tool compatibility mode does not support 64-bit PL/I programs, and standard mode does not support 64-bit COBOL programs. In Debug Tool compatibility mode, specify TCPIP& to debug 64-bit COBOL and C/C++ programs. In standard mode, specify DIRECT& for 64-bit PL/I and C/C++ programs.

***tcpip_id***

TCP/IP name or address where the remote debug daemon is running, in one of the following formats:

**IPv4**

You can specify the address as a symbolic address, such as some.name.com, or a numeric address, such as 9.112.26.333.

**IPv6**

You must specify the address as a numeric address, such as 1080:0:FF::0970:1A21.

***%port_id***

Specifies a unique TCP/IP port on your workstation that is used by the remote debug daemon. The default port number is 8001.

If you changed the default TCP/IP port settings used by the remote debugger client, you must specify the new number as the port ID in your TEST runtime options string. For example, if you changed the default TCP/IP port to 8003, your TEST runtime options string would be `TEST(ALL,'*',PROMPT,'TCPIP&9.112.26.333%8003:')`.

**DBM and DBMDT**
Specifies that z/OS Debugger uses the Debug Manager to automatically determine the client IP and port number to connect to when you start remote debug mode with one of the remote debuggers listed previously under `DIRECT&`, `TCPIP&` or `VADTCPIP&`.

Use `DBM` to start the debugger in standard mode. Use `DBMDT` to start the debugger in Debug Tool compatibility mode.

**Notes:**

- `DBM` and `DBMDT` are only supported with Eclipse IDEs.
- Standard mode does not support commands files or preferences files. If they are specified, they are ignored.
- Standard mode is not supported in IBM Z and Cloud Modernization Stack (Wazi Code).
- Debug Tool compatibility mode does not support 64-bit PL/I programs, and standard mode does not support 64-bit COBOL programs. In Debug Tool compatibility mode, specify `DBMDT%` to debug 64-bit COBOL and C/C++ programs. In standard mode, specify `DBM%` for 64-bit PL/I and C/C++ programs.

Before you start z/OS Debugger with DBM or DBMDT TEST runtime parameters, you must log on to the host via the Remote System Explorer (RSE) in IBM Explorer for z/OS.

You can start a debug session only when Debug Manager and RSE both run in secured mode or unsecured mode. To establish a secured connection between Debug Manager and RSE, they need to use the same certificates or different chained certificates of the same CA root. Otherwise, you need to import the certificates that are regarded as untrusted. For more information, see the "Encrypted communication with Debug Manager" topic in IBM Documentation.

***user_id***
Optionally specifies a user ID for routing the debug session. By default the same user ID as the job launching or running the debug session is assumed.

**RDS**
Specifies that z/OS Debugger connects to Remote Debug Service to initiate a remote debug session with IBM Z Open Debug.

**Usage notes**

- 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 topics for more information related to the material discussed in this topic.

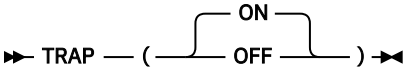
**Related references**
*z/OS Language Environment Debugging Guide*

**Related tasks**
*IBM z/OS Debugger User's Guide*

# TRAP runtime option

Use the TRAP option to specify how z/OS Debugger handles ABENDs and program interrupts.

The syntax for this option is:



**ON**
Enable z/OS Debugger to trap ABENDs.

**OFF**
Prevent z/OS Debugger from trapping ABENDs; an ABEND causes abnormal termination of both z/OS Debugger and the program under test.

# Chapter 2. Common syntax elements in z/OS Debugger commands

Several syntax elements are used in multiple z/OS Debugger 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 topics for more information related to the material discussed in this topic.

**Related references**
“block_name” on page 11
“block_spec” on page 12
“compile_unit_name” on page 13
“cu_spec” on page 13
“expression” on page 14
“load_module_name” on page 14
“load_spec” on page 15
“offset_spec” on page 15
“references” on page 15
“statement_id” on page 16
“statement_id_range and stmt_id_spec” on page 16
“statement_label” on page 17

## address

A hexadecimal address for a location in memory. An *address* can contain up to 16 hexadecimal digits. If *address* contains more than 8 significant hexadecimal digits, z/OS Debugger assumes that *address* references 64-bit addressable storage. If *address* contains 7 or 8 significant hexadecimal digits, z/OS Debugger assumes that *address* references 31-bit addressable storage. Otherwise, z/OS Debugger assumes *address* references 24-bit addressable storage.

References to code (instructions) and save areas can contain no more than 8 significant hexadecimal digits.

*address* must have one of the following formats:

- For all programming languages, x or X followed by apostrophes (') surrounding the hexadecimal value.
- For C, 0x preceding the hexadecimal value.
- For COBOL, H followed by apostrophes (') or quotation marks (") surrounding the hexadecimal value.

  For COBOL or LangX COBOL, X followed by apostrophes (') or quotation marks (") surrounding the hexadecimal value.
- For PL/I, the hexadecimal value surrounded by apostrophes (') or quotation marks ("), followed by PX.
- For assembler or disassembly, X followed by apostrophes (') or quotation marks (") surrounding the hexadecimal value.

## 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.

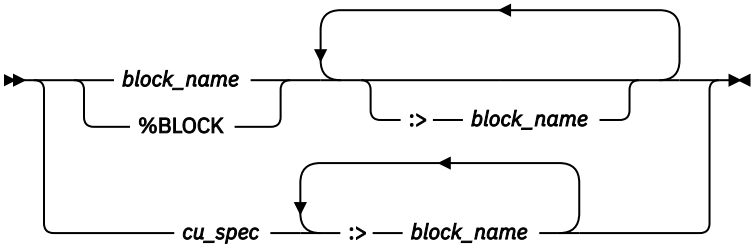
- **For COBOL only**:

  Enclose the block name in quotation marks (") or apostrophes (') if it is case sensitive. If the name is not inside quotation marks or apostrophes, z/OS Debugger will convert the name to uppercase.
  If a name contains an internal quotation mark ("), you should enclose the name in apostrophes (').
  Similarly, if the name contains an internal apostrophe ('), you should enclose the name in quotation marks (").

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 8, "z/OS Debugger variables," on page 333.

***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:

    ```
    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 topics for more information related to the material discussed in this topic.

**Related references**
"block_name" on page 11
Chapter 8, "z/OS Debugger variables," on page 333
"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 topics for more information related to the material discussed in this topic.

**Related references**
"Allowable comparisons for the IF command (COBOL)" on page 132

## compile_unit_name

A *compile_unit_name* identifies any of the following items:

- An assembler CSECT name
- A C or C++ source file
- A LangX COBOL program
- A COBOL program
- 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 one of the following compilers and running in the following environment:
  - Enterprise PL/I for z/OS, Version 3.6 or later
  - Enterprise PL/I for z/OS, Version 3.5 with the PTFs for APARs PK35230 and PK35489 applied
  - Language Environment Version 1.6 through 1.8 with the PTF for APAR PK33738 applied, or later
- 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 quotation marks ("). 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**:

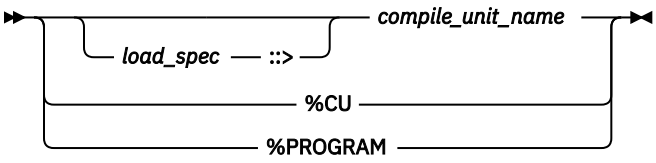
  Enclose the compile unit name in quotation marks (") or apostrophes (') if it is case sensitive. If the name is not inside quotation marks (") or apostrophes ('), z/OS Debugger converts the name to uppercase.
- **For Enterprise PL/I only**:
  - The compile unit name must be enclosed in quotation marks (") or apostrophes ('). If your program was compiled with one of the following compilers and is running in the following environment, you do not need to enclose the compile unit name in quotation marks (") or apostrophes ('):
    - Enterprise PL/I for z/OS, Version 3.6 or later
    - Enterprise PL/I for z/OS, Version 3.5, with the PTFs for APARs PK35230 and PK35489 applied
    - Language Environment Version 1.6 through 1.8 with the PTF for APAR PK33738 applied, or later

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.




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 13.

***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 topics for more information related to the material discussed in this topic.

**Related references**
“load_spec” on page 15
“compile_unit_name” on page 13
Chapter 8, “z/OS Debugger variables,” on page 333

## 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 z/OS Debugger 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 z/OS Debugger is running. For example, if you upgrade your version of the HLL compiler without upgrading your version of z/OS Debugger, certain application programming interface inconsistencies might exist.

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

Refer to the following topics for more information related to the material discussed in this topic.

**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.

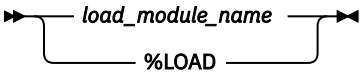
For C, escape sequences in load module names that are specified as strings are not processed if the string is part of a qualification statement.

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 z/OS Debugger, load module names must be unique.

## load_spec

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




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.

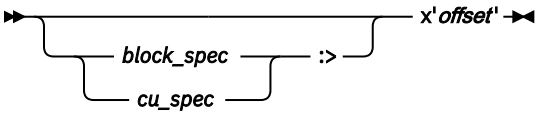
**%LOAD**
Represents the currently qualified load module.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"load_module_name" on page 14
Chapter 8, "z/OS Debugger variables," on page 333

## offset_spec

An *offset_spec* identifies an offset specification.




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

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"block_spec" on page 12
"cu_spec" on page 13

## 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 z/OS Debugger 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]`. However, in assembler,

disassembly, and LangX COBOL, variable names cannot be qualified with load module, compile unit, or block names.

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 a version earlier than Enterprise PL/I Version 3.5 with the PTFs for APARs PK35230 and PK35489 applied, z/OS Debugger 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 one of the following compilers and running in the following environment, z/OS Debugger does support the use of a qualified reference that includes *block_spec*, *cu_spec*, or *load_spec*:

- Enterprise PL/I for z/OS, Version 3.6 or later
- Enterprise PL/I for z/OS, Version 3.5 with the PTFs for APARs PK35230 and PK35489 applied
- Language Environment Version 1.6 through 1.8 with the PTF for APAR PK33738 applied, or later

If you are debugging a program that was compiled with an Enterprise PL/I compiler and z/OS Debugger is at an entry to a block, you cannot 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 z/OS Debugger is running under. For example, if you upgrade your version of the HLL compiler without upgrading your version of z/OS Debugger, certain application programming interface inconsistencies might exist.

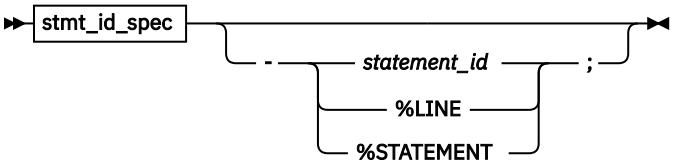
## statement_id

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.

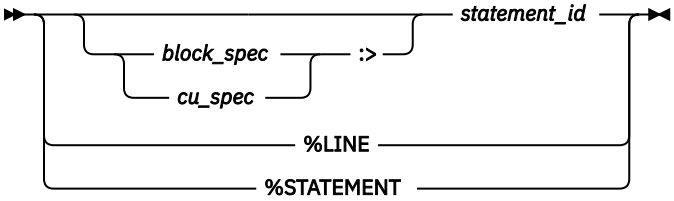
You can only use statement identifiers for statements that are known in the current enclave.

## 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***
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 8, "z/OS Debugger variables," on page 333. %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 topics for more information related to the material discussed in this topic.

**Related references**
"block_spec" on page 12
"cu_spec" on page 13
"statement_id" on page 16
Chapter 8, "z/OS Debugger variables," on page 333

# 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.

## variable_name

A contiguous text string that represents a changeable value. You can create a *variable_name* that can be used in several different programming languages. The *variable_name* must comply with the following syntax rules:

- all uppercase
- starts with one of the characters A through Z
- characters A through Z
- decimal 0 through 9
- no spaces

z/OS Debugger also supports the creation of a *variable_name* that is written to programming language-specific syntax rules. However, if you create a *variable_name* that is written to a specific programming language syntax, you cannot use that *variable_name* in programs written in a different programming language. For example, in COBOL a variable name can contain the dash character (-). If you create a *variable_name* that contains a dash, you cannot use that *variable_name* in a PL/I or C/C++ program.

# Chapter 3. Syntax for assembler and disassembly expressions

Use the syntax defined in this section to write expressions for z/OS Debugger 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:

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

***ddd.ddd*, *dd.d*E*dd*, *dd*E*dd*, *dd.d*E+*dd*, *dd*E+*dd*, *dd.d*E-*dd*, *dd*E-*dd***
A floating-point constant, where *d* is one or more decimal digits and E is the letter "E". Examples: 1.23, 0.22, 12E+10, or 2.456E-5.

**X'*xxxx*' or X"*xxxx*"**
A hexadecimal constant, where *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'*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.

***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, z/OS Debugger 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, z/OS Debugger defines the symbol R7 as the constant 7, not as register R7.

z/OS Debugger 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 z/OS Debugger 32-bit basic 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 32-bit General Purpose Register R0. These are the low-order 32 bits of the 64 bit General Purpose Register.

- RH0, RH1, RH2, RH3, RH4, RH5, RH6, RH7, RH8, RH9, RH10, RH11, RH12, RH13, RH14, RH15. These symbols are implicitly defined as z/OS Debugger 32-bit high general purpose registers. For example, RH0 is defined as %GPRH0. If you are debugging an assembler compilation unit that defines the symbol RH0 and RH0 is not used as a register, you can use the %GPRH0 variable to reference 32-bit high General Purpose Register RH0. These are the high-order 32 bits of the 64 bit General Purpose Register.
- RG0, RG1, RG2, RG3, RG4, RG5, RG6, RG7, RG8, RG9, RG10, RG11, RG12, RG13, RG14, RG15. These symbols are implicitly defined as z/OS Debugger 64-bit General Purpose Registers. For example, RG0 is defined as %GPRG0. If you are debugging an assembler compilation unit that defines the symbol RG0 and RG0 is not used as a register, you can use the %GPRG0 variable to reference 64-bit General Purpose Register R0. These symbols are available only when 64-bit General Purpose Registers are available.
- _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 following syntax:

  For example, `_STORAGE(X'1FF3C'::4)` references the four bytes of storage at address X'1FF3C'. A length of zero might be specified in which case no bytes of storage are accessed. This form is used primarily by the AUTOMONITOR command when displaying an operand of an instruction such as LA that computes an effective address but references no data at that address.

***%symbol***
A valid z/OS Debugger 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*).

**–>, =>, %>, or ==>**
Indirection operator. You can use an indirection operator as follows:

***operand1*<indirection_operator>*operand2***
Use the contents of *operand1* as the base address of the DSECT which contains *operand2*. For example, R1->DCBDDNAME instructs z/OS Debugger to use the contents of register 1 as the base address of the DSECT which contains DCBDDNAME.

***operand1*<indirection_operator> or *operand2*<indirection_operator>+*operand2***
If the <indirection_operator> is followed by a plus sign (+), use *operand2* as an offset. For example, X-> instructs z/OS Debugger to use the contents of X as the address of the storage. For a second example, R3->+X'22' instructs z/OS Debugger 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 is most commonly used 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. If you use this form in a situation where a length is required but not provided by another operand, the length defaults to four.

The following indirection operators indicate which address specification to use:

**->**
Use the current Amode specification.

**==>**
Use a 64–bit address specification.

**=>**
Use a 31–bit address specification.

**%>**
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.

**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.

If the address of the symbol is a 64-bit address, then ADDR' returns an 8-byte value. Otherwise, ADDR' returns a 4-byte value.

**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):

**=**
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.

## Arithmetic expression evaluation

Assembler and disassembly expressions are evaluated in 32-bit precision until a 64-bit operand is encountered. At that point, the precision of both operands is converted to 64-bit and all subsequent operators in the expression are evaluated in 64-bit precision. If you want the entire expression evaluated in 64-bit precision, you can use parentheses to alter the order of operations so that the first operand evaluated has at least one 64-bit operand.

If you are running your program on hardware that does not support 64-bit instructions, z/OS Debugger evaluates the 64-bit arithmetic expressions but you cannot access the 64-bit General Purpose Registers.

# Chapter 4. Syntax for LangX COBOL expressions

**Note:** This chapter is not applicable to IBM Z and Cloud Modernization Stack (Wazi Code).

You can use the syntax defined in this section to write expressions for z/OS Debugger commands while you debug LangX COBOL programs.

In general, whenever you enter a LangX COBOL expression as part of a command (for example, as the operand of the `LIST` *`expression`* command, an assignment command, or the `IF` command), you must enclose the LangX COBOL expression in apostrophes ('). The following example shows the appropriate use of apostrophes:

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

There are some z/OS Debugger commands that can be used for debugging LangX 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 a LangX COBOL program you might use the following command:

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

## Restrictions on LangX COBOL expressions

In addition to the requirement that LangX COBOL expressions be enclosed in apostrophes ('), the following restrictions apply to LangX COBOL expressions:

- The following operators are supported by z/OS Debugger in LangX 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 that non-numeric literals as a substitute for (and equivalent to) upper case letters.
- z/OS Debugger does not support the use of COBOL special registers (for example, DAY, DATE, and TIME) in LangX COBOL expressions.
- All non-numeric literals must be enclosed in quotation marks ("). Apostrophes (') cannot be used.
- You cannot list or alter level-88 variables in LangX COBOL.
- Only the following subset of figurative constants are supported in z/OS Debugger LangX 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 a LangX COBOL expression:

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

**X"xxxxx"**
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 LangX COBOL source program. Examples:

```
LASTNAME
USERVAR8
12CENTS
```

z/OS Debugger implicitly defines the _STORAGE symbol in all LangX 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" on page 24. 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 z/OS Debugger variable or built-in function. For example:

```
%ADDRESS
%HEX(expression)
```

# Operators

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

## Operators that can be used in any expression

Use the operators defined in this section to write LangX 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. z/OS Debugger 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 z/OS Debugger command syntax by either pressing PF1 or entering a question mark (?) on the command line. This lists all z/OS Debugger commands in the Log window.

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

Remote debug mode only accepts these commands (if indicated) if you run it in Debug Tool compatibility mode.

The table below summarizes the z/OS Debugger commands.

| Command | Description |
|---|---|
| "? command" on page 31 | Displays all z/OS Debugger commands in the Log window. |
| "ALLOCATE command" on page 31 | Allocates a file to an existing data set, a concatenation of existing data sets, or a temporary data set. |
| "ANALYZE command (PL/I)" on page 32 | Displays the process of evaluating an expression and the data attributes of any intermediate results. |
| "Assignment command (assembler and disassembly)" on page 33 | Assigns the value of an expression to a specified storage location or register. |
| "Assignment command (LangX COBOL)" on page 35 | Assigns the value of an expression to a specified reference. |
| "Assignment command (PL/I)" on page 36 | Assigns the value of an expression to a specified reference. |
| "AT command" on page 37 | Defines a breakpoint (gives control of your program to z/OS Debugger under the specified circumstances). |
| "BEGIN command" on page 74 | BEGIN and END delimit a sequence of one or more commands to form one longer command. |
| "block command (C and C++)" on page 75 | Allows you to group any number of z/OS Debugger commands into one command. |
| "break command (C and C++)" on page 75 | 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. |
| "CALL command" on page 76 | The `CALL` command calls either a procedure, entry name, or program name, or it requests that a utility function be run. |
| "CC command" on page 85 | Controls whether code coverage data is collected. |
| "CLEAR command" on page 86 | Removes the actions of previously issued z/OS Debugger commands (such as breakpoints). |
| "COMMENT command" on page 93 | Used to insert commentary into the session log. |
| "COMPUTE command (COBOL)" on page 94 | Assigns the value of an arithmetic expression to a specified reference. |

| Command | Description |
|---|---|
| "CURSOR command (full-screen mode)" on page 95 | Moves the cursor between the last saved position on the z/OS Debugger session panel (excluding the header fields) and the command line. |
| "Declarations (assembler, disassembly, and LangX COBOL)" on page 95 | Declares session variables that are effective during a z/OS Debugger session. |
| "Declarations (C and C++)" on page 96 | Declares session variables and tags that are effective during a z/OS Debugger session. |
| "Declarations (COBOL)" on page 99 | Declares session variables that are effective during a z/OS Debugger session. |
| "DECLARE command (PL/I)" on page 101 | Declares session variables that are effective during a z/OS Debugger session. |
| "DESCRIBE command" on page 103 | Displays the attributes of references, compile units, and the execution environment. |
| "DISABLE command" on page 107 | Makes the AT breakpoint inoperative, but does not clear it; you can ENABLE it later without typing the entire command again. |
| "do/while command (C and C++)" on page 110 | Performs a command before evaluating the test expression. |
| "DO command (PL/I)" on page 111 | Allows one or more commands to be collected into a group which can (optionally) be run repeatedly. |
| "ENABLE command" on page 113 | Makes AT breakpoints operative after they have been disabled by the DISABLE command. |
| "EVALUATE command (COBOL)" on page 115 | Provides a shorthand notation for a series of nested IF statements. |
| "Expression command (C and C++)" on page 116 | Evaluates the given expression which can be used to either assign a value to a variable or to call a function. |
| "FIND command" on page 117 | Provides full-screen and line mode searching of source and listing files, and full-screen searching of Log and Monitor windows. |
| "FINDBP command" on page 121 | Provides full-screen searching of the source for line, statement, and offset breakpoints. |
| "for command (C and C++)" on page 123 | Provides iterative looping. |
| "FREE command" on page 124 | Frees (deallocates) an allocated file. |
| "GO command" on page 124 | Causes z/OS Debugger to start or resume running your program. |
| "GOTO command" on page 125 | Causes z/OS Debugger to resume program execution at the specified statement id. |
| "GOTO LABEL command" on page 127 | Causes z/OS Debugger to resume running program at the specified statement label. |
| "%IF command (programming language neutral)" on page 129 | Lets you conditionally perform a command; use this syntax if you are constructing a command that might run in different programming languages. |
| "IF command (assembler, disassembly, and LangX COBOL)" on page 130 | Lets you conditionally perform a command. |

| Command | Description |
|---|---|
| "if command (C and C++)" on page 130 | Lets you conditionally perform a command. |
| "IF command (COBOL)" on page 131 | Lets you conditionally perform a command. |
| "IF command (PL/I)" on page 134 | Lets you conditionally perform a command. |
| "IMMEDIATE command (full-screen mode)" on page 135 | Causes a command within a command list to be performed immediately. For use with commands assigned to a PF key. |
| "INPUT command (C, C++, and COBOL)" on page 135 | Provides input for an intercepted read and is valid only when there is a read pending for an intercepted file. |
| "JUMPTO command" on page 136 | Jumps to the specified statement and then stops the program at that statement. |
| "LIST command" on page 140 | Displays information about your z/OS Debugger session. |
| "LOAD command" on page 166 | Specifies that the named module should be loaded for debugging purposes. |
| "LOADDEBUGDATA command" on page 167 | Specifies that a compile unit (CU) as an assembler CU and loads debug data. |
| "MEMORY command" on page 169 | Identifies an address in memory to display in the Memory window. |
| "MONITOR command" on page 171 | 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). |
| "MOVE command (COBOL)" on page 175 | Transfers data from one area of storage to another. |
| "NAMES command" on page 179 | Specify names of load modules or compile units to debug or ignore, and display the current setting of the NAMES command. |
| "Null command" on page 181 | A semicolon written where a command is expected. |
| "ON command (PL/I)" on page 181 | Establishes the actions to be executed when the specified PL/I condition is raised. |
| "PANEL command (full-screen mode)" on page 183 | Displays special panels (for example, to customize your full-screen session). |
| "PERFORM command (COBOL)" on page 185 | 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. |
| "PLAYBACK commands" on page 187 | Commands to start and stop recording application execution states and replay the recorded execution states. |
| "POPUP command" on page 191 | Displays the Command pop-up window, where you type in commands. |
| "POSITION command" on page 191 | Positions the cursor to a specific line in the specified window. |
| "Prefix commands (full-screen mode)" on page 192 | Apply only to source listing lines and are typed into the Source window. |
| "PROCEDURE command" on page 193 | Allows the definition of a group of commands that can be accessed using the CALL procedure command. |
| "QUALIFY RESET command" on page 194 | Resets qualification to the block of the suspended program and scrolls the Source window to display the current statement line. |

| Command | Description |
|---|---|
| “QUERY command” on page 194 | Displays the current value of z/OS Debugger settings (such as the current location in the suspended program). |
| “QUIT command” on page 199 | Ends a z/OS Debugger session (with a return code, if specified). |
| “QQUIT command” on page 200 | Ends a z/OS Debugger session (without additional prompting) |
| “RETRIEVE command (full-screen mode)” on page 202 | Displays the last command entered on the command line. |
| “RESTORE command” on page 201 | Enables explicit restoring of settings, breakpoints, and monitor specifications. |
| “RUN command” on page 203 | Causes z/OS Debugger to start or resume running your program. |
| “RUNTO command” on page 203 | Causes z/OS Debugger to run your program to a specific point (without setting a breakpoint) |
| “SCROLL command (full-screen mode)” on page 204 | Provides horizontal and vertical scrolling in full-screen mode. |
| “SELECT command (PL/I)” on page 207 | Chooses one of a set of alternate commands. |
| “SET command” on page 207 | Controls various z/OS Debugger settings. |
| “SET command (COBOL)” on page 266 | Assigns a value to a COBOL reference. |
| “SHOW prefix command (full-screen mode)” on page 269 | 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. |
| “STEP command” on page 269 | Causes z/OS Debugger to dynamically step through a program, running one or more program statements. |
| “STORAGE command” on page 271 | Enables you to alter up to eight bytes of storage. |
| “switch command (C and C++)” on page 273 | Enables you to transfer control to different commands within the switch body, depending on the value of the switch expression. |
| “SYSTEM command (z/OS)” on page 275 | Lets you issue TSO commands during a z/OS Debugger session. |
| “TRIGGER command” on page 276 | Raises the specified AT condition in z/OS Debugger, or raises the specified programming language condition in your program. |
| “TSO command (z/OS)” on page 280 | Lets you issue TSO commands during a z/OS Debugger session (this command is valid only in a TSO environment). |
| “USE command” on page 280 | Causes the z/OS Debugger commands in the specified file or data set to be either performed or syntax checked. |
| “while command (C and C++)” on page 281 | Enables you to repeatedly perform the body of a loop until the specified condition is no longer met or evaluates to false |
| “WINDOW command (full-screen mode)” on page 282 | Opens, close, resizes, or expands to full screen (zooms) the specified window on the z/OS Debugger session panel. |

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**

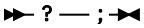
*IBM z/OS Debugger User's Guide*

**Related references**

Chapter 7, “z/OS Debugger built-in functions,” on page 327
Chapter 8, “z/OS Debugger variables,” on page 333

# ? command

The ? command displays a list of z/OS Debugger commands in the Log window.




### Usage note

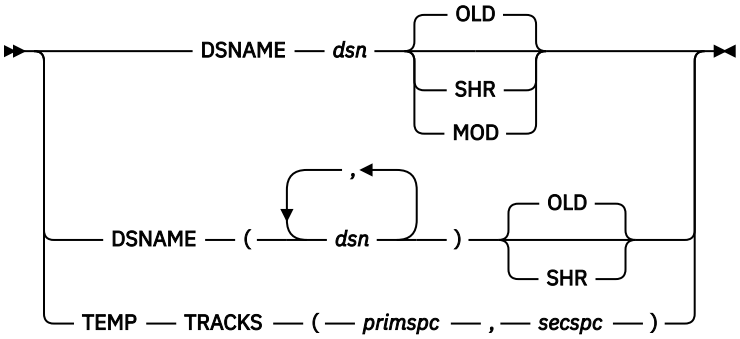
In the following cases, z/OS Debugger does not display the syntax help after you enter the ? command:

- The z/OS Debugger 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.


**attributes**




**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**

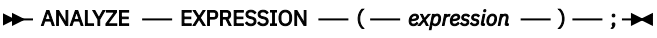
This command is not available under CICS.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“FREE command” on page 124
“DESCRIBE command” on page 103

# 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.




**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 z/OS Debugger 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, z/OS Debugger 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, z/OS Debugger 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:

```
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
```

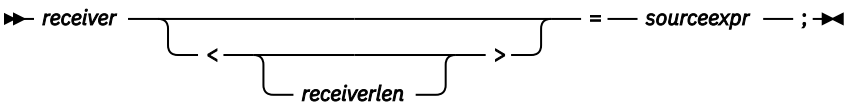
Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

“SET WARNING command (C, C++, COBOL, and PL/I)” on page 263
“PLAYBACK commands” on page 187

# Assignment command (assembler and disassembly)

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




***receiver***
A valid z/OS Debugger assembler reference or expression.

***receiverlen***
A valid z/OS Debugger 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 z/OS Debugger assembler expression.

**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 <l'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 topics for more information related to the material discussed in this topic.

**Related references**
“references” on page 15
“PLAYBACK commands” on page 187

## 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.

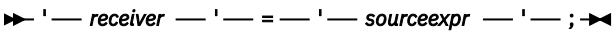
**Error**
Statement that is flagged as not valid.

*Table 4. Assignment rules depending on the source and receiver type*

| Receiver | | Source | | Assignment type | Pad or Truncate |
|---|---|---|---|---|---|
| **Type** | **Length** | **Type** | **Length** | | |
| * | 1 – * | ? | ? | Move | None |
| F, H, A, Y | 1 – 4 | F, H, A, Y, X, B, C | 1 – 4 | Arithmetic | Left |
| | | E, D, L | 4, 8, 16 | Hex Float | Right - 0 |
| | | P, Z | 1 – * | Arithmetic | |
| | | X, B, C | >4 | Error | |
| | | Other | Other | Error | |
| X | 1 – 4 | F, H, A, Y | 1 – 4 | Arithmetic | Left |
| | | P, Z | 1 – * | Arithmetic | |
| | 1 – * | X, B | 1 – * | Bit | Right – 0 |
| | | C | | Bit | Right – 0 |
| | | Other | | Error | |
| C | 1 – 4 | F, H, A, Y | 1 – 4 | Arithmetic | Left |
| | | P, Z | 1 – * | Arithmetic | |
| | 1 – * | X, B | 1 – * | Bit | Right – 0 |
| | | C | | Character | Right – blank |
| | | Other | | Error | |
| P, Z | 1 – * | P, Z | 1 – * | Packed | |
| | | F, H, A, Y, X, B, C | 1 – 4 | Packed | |
| | | E, D, L | 4, 8, 16 | Hex Float | Right - 0 |
| E, D, L | 4, 8, 16 | X | = | Move | None |
| | | E, D, L | 4, 8, 16 | Hex Float | Right - 0 |
| | | F, H, A, Y | 1 - 4 | Hex Float | Right - 0 |
| | | P, Z | 1 - * | Hex Float | Right - 0 |
| ? | 1 – 4 | F, H, A, Y | 1 – 4 | Arithmetic | Left |
| | 1 – * | X, B, C | 1 – * | Bit | Right – 0 |
| All others | | | | Error | |

## Assignment command (LangX COBOL)

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




***receiver***
A valid z/OS Debugger LangX COBOL reference enclosed in apostrophes (').

***sourceexpr***
A valid z/OS Debugger LangX COBOL expression enclosed in apostrophes (').

**Usage notes**

- When *receiver* is an arithmetic variable, then *sourceexpr* can be a hexadecimal string of the same length as *receiver*. z/OS Debugger 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 z/OS Debugger 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*.

```
'x' = '6' ;
```

- Increment the value of *X* by 5.

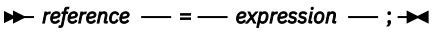
```
'X' = 'X + 5' ;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“references” on page 15
“PLAYBACK commands” on page 187

## Assignment command (PL/I)

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




***reference***
A valid z/OS Debugger PL/I reference.

***expression***
A valid z/OS Debugger PL/I expression.

**Usage notes**

- The PL/I repetition factor is not supported by z/OS Debugger.

  For example, the following is not valid: `rx = (16)'01'B;`
- If z/OS Debugger 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 z/OS Debugger cannot determine variable values within statements, only at statement boundaries.
- The PL/I assignment statement option BY NAME is not valid in the z/OS Debugger.
- If you are debugging a Enterprise PL/I program, the target of an assignment command cannot 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 z/OS Debugger variable %GPR5 the address of `name_table`.

```
%GPR5 = ADDR (name_table);
```

- Assign to the `prg_name` variable the value of z/OS Debugger variable %PROGRAM.

```
prg_name = %PROGRAM;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“references” on page 15
“PLAYBACK commands” on page 187

# AT command

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

For MVS batch, TSO, and CICS programs, the SET SAVE and SET RESTORE commands can be used to automatically save and restore breakpoints between z/OS Debugger 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, z/OS Debugger 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
- GLOBAL CALL 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.

| Command | Description |
|---|---|
| "AT ALLOCATE (PL/I) command" on page 41 | Gives z/OS Debugger control when storage for a named controlled variable or aggregate is dynamically allocated by PL/I. |
| "AT APPEARANCE command" on page 42 | Gives z/OS Debugger control:<br>• For C and PL/I, when the specified compile unit is found in storage<br>• For COBOL, the first time the specified compile unit is called |
| "AT CALL command" on page 43 | Gives z/OS Debugger control on an attempt to call the specified entry point. |
| "AT CHANGE command (full screen mode, line mode, batch mode)" on page 45 | Gives z/OS Debugger control when either the specified variable value or storage location is changed. |
| "AT CHANGE command (remote debug mode)" on page 50 | Gives z/OS Debugger control when the specified variable value is changed. |
| "AT CURSOR command (full-screen mode)" on page 52 | Defines a statement breakpoint by cursor pointing. |
| "AT DATE command (COBOL)" on page 53 | For COBOL, gives z/OS Debugger control for each date processing statement within the specified block. |
| "AT DELETE command" on page 53 | Gives z/OS Debugger control when a load module is deleted. |
| "AT ENTRY command" on page 54 or "AT ENTRY command (remote debug mode)" on page 56 | Defines a breakpoint at the specified entry point. |
| "AT EXIT command" on page 56 | Defines a breakpoint at the specified exit point. |
| "AT GLOBAL command" on page 58 | Gives z/OS Debugger control for every instance of the specified AT-condition. |
| "AT LABEL command" on page 60 | Gives z/OS Debugger control at the specified statement label. |
| "AT LINE command" on page 63 | Gives z/OS Debugger control at the specified line. |

| Command | Description |
|---|---|
| "AT LOAD command" on page 63 or "AT LOAD command (remote debug mode)" on page 65 | Gives z/OS Debugger control when the specified load module is loaded. |
| "AT OCCURRENCE command" on page 65 | Gives z/OS Debugger control on a language or Language Environment condition or exception. |
| "AT OFFSET command (disassembly)" on page 68 | Gives z/OS Debugger control at the specified offset in the disassembly view. |
| "AT PATH command" on page 69 | Gives z/OS Debugger control at a path point. |
| "AT Prefix command (full-screen mode)" on page 70 | Defines a statement breakpoint through the Source window prefix area. |
| "AT STATEMENT command" on page 70 or "AT STATEMENT command (remote debug mode)" on page 73 | Gives z/OS Debugger control at the specified statement. |
| "AT TERMINATION command" on page 73 | Gives z/OS Debugger control when the application program is terminated. |

**Usage notes**

- To set breakpoints at specific locations in a program, z/OS Debugger 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 z/OS Debugger, 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, z/OS Debugger 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 stop your program only under certain conditions, you can use a combination of the AT and IF command or the AT command with a WHEN condition 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 topics for more information related to the material discussed in this topic.

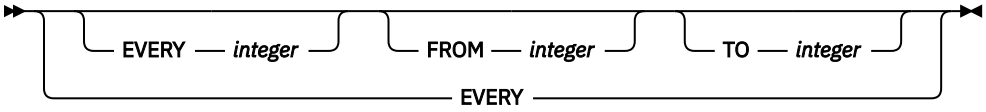
**Related tasks**
*IBM z/OS Debugger User's Guide*

**Related references**
“LIST command” on page 140

## 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***
Specifies how frequently the breakpoint is taken. For example, EVERY 5 means that z/OS Debugger is started every fifth time the AT-condition is met. The default is EVERY 1.

**FROM *integer***
Specifies when z/OS Debugger invocations are to begin. For example, FROM 8 means that z/OS Debugger 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 z/OS Debugger invocations are to end. For example, TO 20 means that after the 20th time this AT-condition is met, it should no longer start z/OS Debugger. 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.

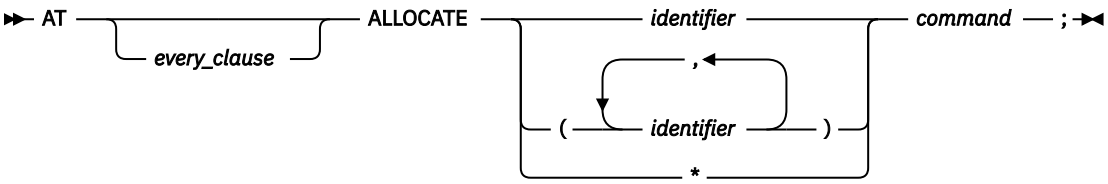
```
AT EVERY 3 FROM 48 TO 59 STATEMENT 50;
```

- 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) command

AT ALLOCATE gives z/OS Debugger 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 z/OS Debugger. 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 z/OS Debugger 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.

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

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

```
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.';
```

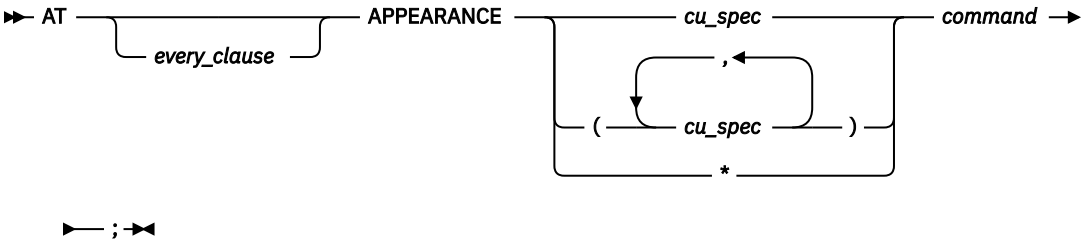
When temp is allocated the value of temp has not yet been initialized. When it is initialized to 'abc' by the INITIAL phrase, the first AT CHANGE is recognized and 'abc' is listed. The three assignments to temp cause the value to be set again but the third assignment doesn't change the value. This example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

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“every_clause syntax” on page 40
“PLAYBACK commands” on page 187

# AT APPEARANCE command

Gives z/OS Debugger 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.




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

***command***
A valid z/OS Debugger 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 `cux` that is in load module `loady` appears, the breakpoint `AT APPEARANCE loadx::>cux` will not be triggered.
- If the compile unit is *not* qualified with a load module name, the current load module qualification is not used.
- z/OS Debugger gains control when the specified compile unit is first recognized by z/OS Debugger. 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 z/OS Debugger 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, C++, and Enterprise COBOL for z/OS Version 5 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++, Enterprise COBOL for z/OS Version 5, and PL/I only:** An APPEARANCE breakpoint is triggered when z/OS Debugger finds the specified compile unit in storage. To be triggered, however, the APPEARANCE breakpoint must be set before the compile unit is loaded.
- **For Enterprise COBOL for z/OS Version 4 or earlier:** An APPEARANCE breakpoint is triggered when z/OS Debugger 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;
  GO;
}
```

- 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;
```

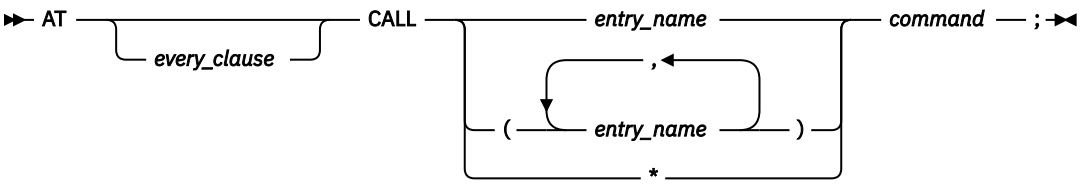
If cuy is later deleted from storage, the breakpoints that are dependent on cuy are automatically cleared. However, if cuy is then loaded again, the APPEARANCE breakpoint for cuy is triggered and the AT EXIT and AT LABEL breakpoints are redefined.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“every_clause syntax” on page 40
“cu_spec” on page 13
“PLAYBACK commands” on page 187

## AT CALL command

Gives z/OS Debugger 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.




***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 z/OS Debugger command.

**Usage notes**

- AT CALL intercepts the call itself, not the subroutine entry point. C, COBOL, and PL/I programs compiled with the PATH suboption of the TEST or DEBUG 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 z/OS Debugger 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.
- z/OS Debugger does not support the AT CALL command while you debug a LangX COBOL or any 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.
    - 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 quotation marks (") or apostrophes (').
  - To be able to set breakpoints in a COBOL program by using the AT CALL command, you must compile your program with the correct TEST compiler suboptions. The following list describes the TEST compiler suboptions to use for the corresponding version of the COBOL compiler:
    - Specify the HOOK or NOHOOK suboption of the TEST compiler option for Enterprise COBOL for z/OS, Version 4
    - Specify the PATH, ALL, or NONE suboption of the TEST compiler option for the following compilers:
      - Enterprise COBOL for z/OS and OS/390, Version 3
      - COBOL for OS/390 & VM, Version 2

    If you compile your program with one of the following compilers and suboptions, you cannot use the AT CALL *entry_name* command:
    - It is not supported for Enterprise COBOL for z/OS Version 5.
    - NOHOOK suboption of the TEST compiler option for Enterprise COBOL for z/OS, Version 4.
    - NONE suboption of the TEST compiler option for the following compilers:
      - Enterprise COBOL for z/OS and OS/390, Version 3.
      - COBOL for OS/390 & VM, Version 2.

    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 z/OS Debugger 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;
}
```

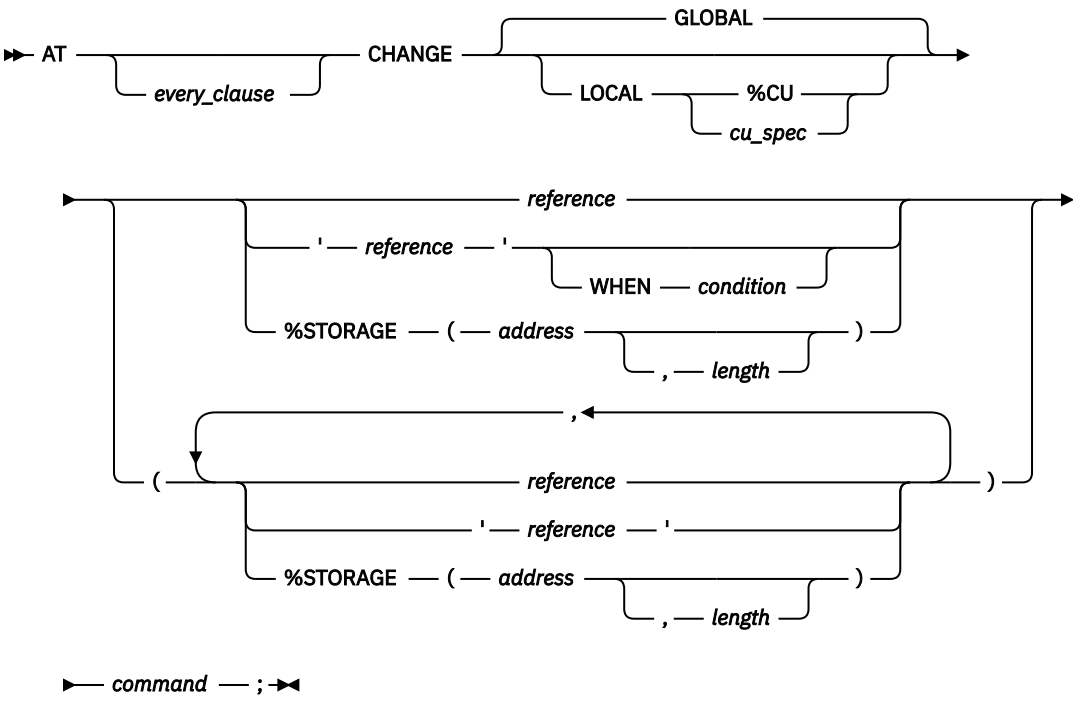
Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
*IBM z/OS Debugger User's Guide*

**Related references**
“every_clause syntax” on page 40
“PLAYBACK commands” on page 187

## AT CHANGE command (full screen mode, line mode, batch mode)

Gives z/OS Debugger control when either the program or z/OS Debugger command changes the specified variable value or storage location.




**GLOBAL**
Specifies that the AT CHANGE breakpoint is global. The AT CHANGE breakpoint is not limited to a specific compile unit; it spans the entire application. This is the default.

**LOCAL**
Specifies that the AT CHANGE breakpoint is limited to a specific compile unit.

***cu_spec***
A valid compile unit specification. Specifies that the AT CHANGE breakpoint is limited to this compile unit.

***condition***
A valid, simple z/OS Debugger conditional expression. Simple means that you use only one operator; for example, a < b.

***reference***
A valid z/OS Debugger reference in the current programming language.

***'reference'***
A valid z/OS Debugger reference when the current programming language is LangX 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.

***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 z/OS Debugger command. If you are using remote debug mode, you can specify only commands that are supported in remote debug mode.

**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 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 z/OS Debugger 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 z/OS Debugger 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, z/OS Debugger 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 z/OS Debugger built-in function that is available only with the AT CHANGE command.
- For a CICS application on z/OS Debugger, the CHANGE %STORAGE breakpoint is cleared at the end of the last process in the application. For a non-CICS application on z/OS Debugger, 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 z/OS Debugger 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`.
  - 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.

  - If *reference* is a pointer, z/OS Debugger stops when the contents of storage at the address given by that pointer changes.
- 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.
  - 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/I 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.
- For PL/I and C/C++, when you specify a *reference*, z/OS Debugger calculates the address of the *reference* only once, when it runs the AT CHANGE command the first time. Thereafter, z/OS Debugger monitors the storage location indicated by that address.

  For the following items, z/OS Debugger recalculates the address of *reference* each time it monitors the storage location. If the address of *reference* changes, z/OS Debugger uses the new storage location as the address to monitor:
  - COBOL variables whose address can change
  - Assembler DSECT items that are in the range of an active USING when you enter the AT CHANGE command
  - Assembler absolute locations that are in the range of an active USING when you enter the AT CHANGE command
- 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 COBOL level-88 variable on an AT CHANGE command, the current setting of the value is saved. z/OS Debugger stops at the breakpoint only if the setting of the COBOL 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, z/OS Debugger stops at the breakpoint. Note that level-88 variables cannot be listed in LangX COBOL.
- To use a COBOL level-88 variable with the AT CHANGE command, you (through a z/OS Debugger 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, z/OS Debugger 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, z/OS Debugger stops and runs the command associated with the breakpoint.
- When z/OS Debugger evaluates the *condition* and the *condition* is invalid, z/OS Debugger does one of the following actions:
  - If `SET WARNING` is set to `ON`, z/OS Debugger stops and displays a message that it could not evaluate the *condition*. You need to enter a command to indicate what action you want z/OS Debugger to take.
  - If `SET WARNING` is set to `OFF`, z/OS Debugger does not stop nor display a message that it could not evaluate the *condition*. z/OS Debugger continues running the program.
- If you specify *address* with more than 8 significant digits or if *reference* references 64-bit addressable storage, z/OS Debugger assumes that the storage location is 64-bit addressable storage. Otherwise, z/OS Debugger assumes that the storage location is 31-bit addressable storage.

**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 (0x00001000, 20)     /* Breakpoint 1 set */
AT CHANGE %STORAGE (0x00001000, 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.

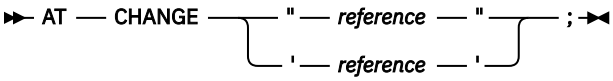
Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
"Controlling how z/OS Debugger handles invalid comparisons" in the IBM z/OS Debugger User's Guide

**Related references**
“address” on page 11
“every_clause syntax” on page 40
“references” on page 15
“PLAYBACK commands” on page 187
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

# AT CHANGE command (remote debug mode)

Gives z/OS Debugger control when the program changes the specified variable value.




**'*reference*' or "*reference*"**
A valid z/OS Debugger reference in the current programming language.

**Usage notes**

- When you enter an AT CHANGE command, the breakpoint is set relative to the location the program is stopped, which might not be the program displayed in the source view. For example, your program is stopped at program SUB1, which was called by program MAIN1, and the source view displays the source for program SUB1. Then, you click on MAIN1 in the Debug view so that the source view displays the source for MAIN1. If you enter the command AT CHANGE "Var1", a breakpoint is set to monitor any changes to a variable called "Var1" in SUB1, not a variable called "Var1" in MAIN1.
- 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 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 z/OS Debugger 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 z/OS Debugger 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, z/OS Debugger 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).

- The referenced variable must exist when the AT CHANGE breakpoint is defined.
- 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 or variable values.
- 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`.
  - 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.
  - If *reference* is a pointer, z/OS Debugger stops when the contents of storage at the address given by that pointer changes.
- 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.
  - 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/I 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.
- 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.

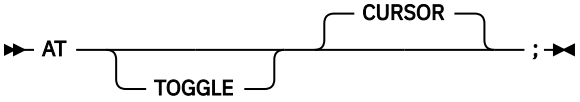
- For optimized COBOL programs, the specified variable cannot be a variable that was discarded due to compiler optimization.
- When you use a COBOL level-88 variable on an AT CHANGE command, the current setting of the value is saved. z/OS Debugger stops at the breakpoint only if the setting of the COBOL 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, z/OS Debugger stops at the breakpoint.
- To use a COBOL level-88 variable with the AT CHANGE command, you (through a z/OS Debugger 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, z/OS Debugger behavior becomes unpredictable.
- If *reference* references 64-bit addressable storage, z/OS Debugger assumes that the storage location is 64-bit addressable storage. Otherwise, z/OS Debugger assumes that the storage location is 31-bit addressable storage.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"references" on page 15
"AT CHANGE command (full screen mode, line mode, batch mode)" on page 45
Appendix A, "z/OS Debugger commands supported in Debug Tool compatibility mode," on page 517

## AT CURSOR command (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.

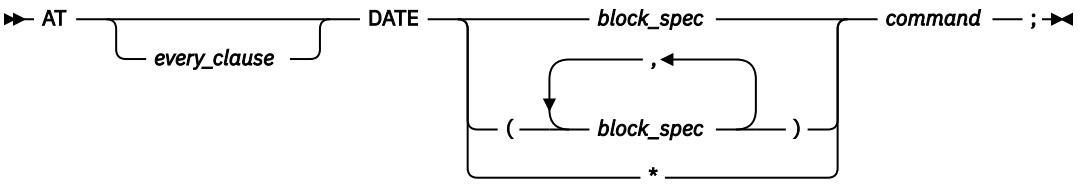
```
SET PF10 = AT TOGGLE CURSOR;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"PLAYBACK commands" on page 187

# AT DATE command (COBOL)

Gives z/OS Debugger 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.




*****
Sets a breakpoint at every date processing statement.

***command***
A valid z/OS Debugger 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 subrx, 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.

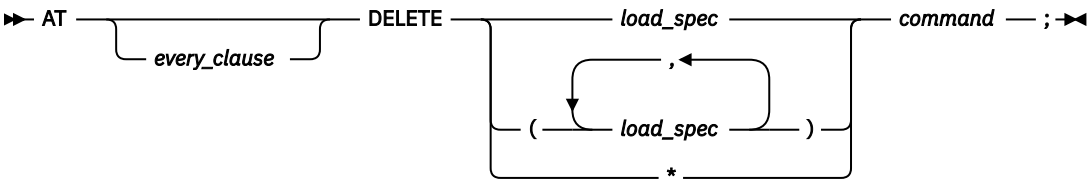
```
AT DATE * PERFORM
   QUERY LOCATION;
   LIST DATE-FIELD
   GO;
END-PERFORM;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“every_clause syntax” on page 40
“block_spec” on page 12
“PLAYBACK commands” on page 187

# AT DELETE command

Gives z/OS Debugger 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.




*****
Sets a breakpoint at every DELETE of any load module.

***command***
A valid z/OS Debugger command.

**Usage notes**

- z/OS Debugger gains control for deletes that are affected by the Language Environment delete service, MVS delete service, or EXEC CICS RELEASE. If the Dynamic Debug facility is deactivated (by entering the SET DYNDEBUG OFF command) or SVC screening is disabled, z/OS Debugger is not notified of deletes affected by the MVS delete service. Refer to *IBM z/OS Debugger Customization Guide* for instructions on how to control SVC screening.
- AT DELETE cannot specify the initial load module.
- If this breakpoint is set in a parent enclave, it can be triggered and operated on with commands while the application is in a child enclave.
- For a CICS application on z/OS Debugger, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on z/OS Debugger, it is cleared at the end of a process.
- The AT DELETE command cannot be used while you replay recorded statements by using the PLAYBACK commands.

**Examples**

- Each time a load module is deleted, request input from the terminal.

```
AT DELETE *;
```

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

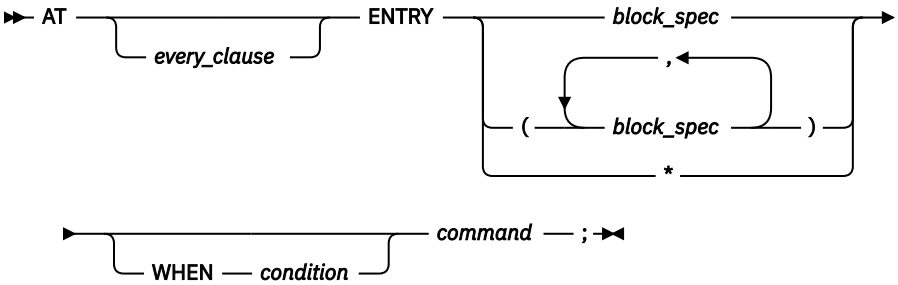
```
AT DELETE mymod CLEAR AT CHANGE (var1:>x);
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“every_clause syntax” on page 40
“load_spec” on page 15
“PLAYBACK commands” on page 187

## AT ENTRY command

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



*****
Sets a breakpoint at every ENTRY of any block.

***command***
A valid z/OS Debugger command. If you are using remote debug mode, you can specify only commands that are supported in remote debug mode.

***condition***
A valid z/OS Debugger conditional expression.

**Usage notes**

- For VS COBOL II programs, z/OS Debugger supports only the AT ENTRY * command.
- To specify an AT ENTRY breakpoint for a program that is not currently known to z/OS Debugger, 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 z/OS Debugger detects a program name that matches the one you specified, it sets the breakpoint.
- An ENTRY 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 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 z/OS Debugger, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on z/OS Debugger, it is cleared at the end of a process.
- ENTRY breakpoints for blocks in a fetched or loaded program are converted to deferred breakpoints when that program is released.
- The AT ENTRY command 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 can restrict the circumstances under which the AT ENTRY break point is raised by specifying a WHEN condition. If a WHEN condition is specified, z/OS Debugger stops at the AT ENTRY break point if the specified entry point matches the current entry point and the WHEN condition is true.
- 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 ENTRY command with a WHEN condition, every time z/OS Debugger reaches the entry, it evaluates the condition. If the condition evaluates to true, z/OS Debugger stops and runs the command associated with the breakpoint.
- When z/OS Debugger evaluates the condition and the condition is invalid, z/OS Debugger does one of the following actions:
  - If SET WARNING is set to ON, z/OS Debugger stops and displays a message that it could not evaluate the condition. You need to enter a command to indicate what action you want z/OS Debugger to take.
  - If SET WARNING is set to OFF, z/OS Debugger does not stop nor display a message that it could not evaluate the condition. z/OS Debugger continues running the program.
- A deferred AT ENTRY command creates an implicit NAMES INCLUDE for the target of the deferred AT ENTRY.
- You cannot use the AT ENTRY command to stop at the entry of a nested block in a C or C++ program. A nested block is a group of statements delimited by { and }. The compiler assigns a name to these blocks using the following pattern: %BLOCK*n*, where *n* is a sequentially-assigned number.

**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 the entry of program myprog with parameter myparm, to stop at the entry point to myprog only when myparm equals 100, enter the following command:

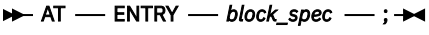
```
AT ENTRY myprog WHEN myparm=100;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“every_clause syntax” on page 40
“condition” on page 12
“block_spec” on page 12
“AT APPEARANCE command” on page 42
“PLAYBACK commands” on page 187
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

## AT ENTRY command (remote debug mode)

Defines a breakpoint at the entry point of the specified block.

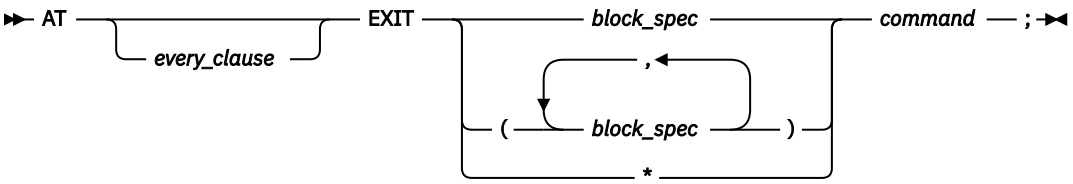



Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“block_spec” on page 12
“AT ENTRY command” on page 54
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

# AT EXIT command

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




*****
Sets a breakpoint at every EXIT of any block.

***command***
A valid z/OS Debugger command.

**Usage notes**

- For VS COBOL II programs, z/OS Debugger supports only the AT EXIT * command.
- 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.
- An 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 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 z/OS Debugger, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on z/OS Debugger, 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.
- The AT EXIT command cannot be used while you replay recorded statements by using the PLAYBACK commands.
- 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 a LangX 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
- You cannot use the AT EXIT command to stop at the exit of a nested block in a C or C++ program. A nested block is a group of statements delimited by { and }. The compiler assigns a name to these blocks using the following pattern: %BLOCK*n*, where *n* is a sequentially-assigned number.

**Example**

At exit of `main`, print a message and TRIGGER the SIGUSR1 condition. The current programming language setting is C.

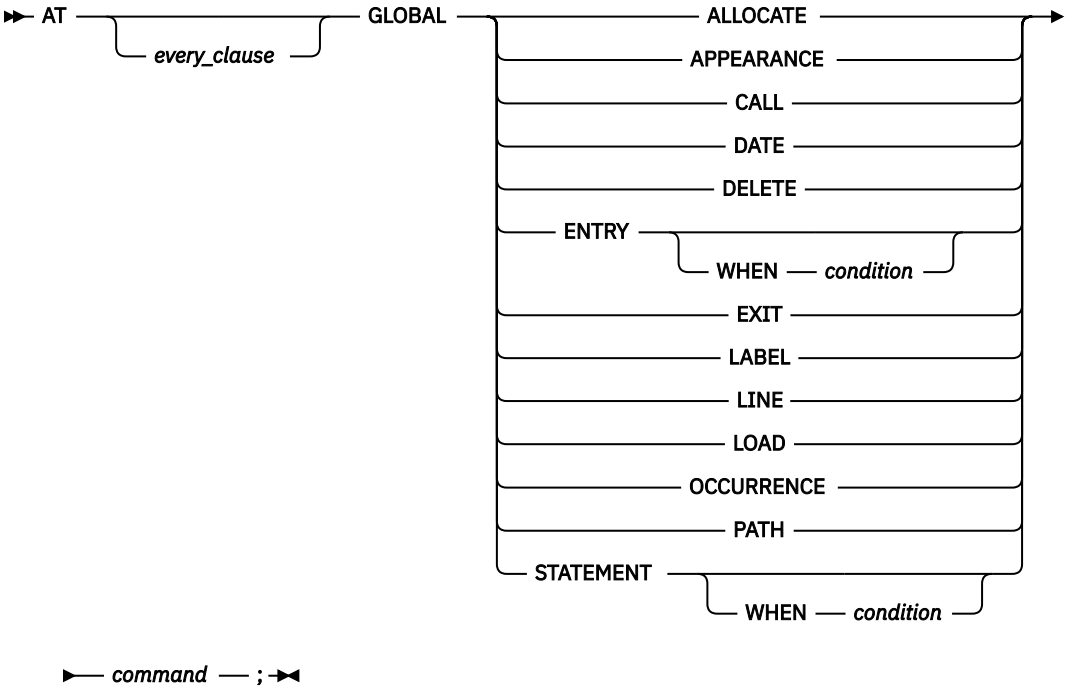
```
AT EXIT main {
  puts("At exit of the program");
  TRIGGER SIGUSR1;
  GO;
}
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“every_clause syntax” on page 40
“block_spec” on page 12
“PLAYBACK commands” on page 187

# AT GLOBAL command

Gives z/OS Debugger 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.




***command***
A valid z/OS Debugger 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_Unknwn but cannot remember the name, you can issue AT GLOBAL ENTRY and z/OS Debugger 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**

- z/OS Debugger does not support the AT CALL, AT LABEL and AT PATH commands for disassembled or VS COBOL II programs.
- z/OS Debugger does not support the AT CALL command for LangX 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.
- The AT GLOBAL OCCURRENCE breakpoint takes precedence over an AT OCCURRENCE *condition* breakpoint.
- The AT GLOBAL OCCURRENCE command takes precedence over the **test_level** setting of the TEST runtime option. For example, if your **test_level** setting is ALL, a condition is raised, and you set an AT GLOBAL OCCURRENCE breakpoint, then z/OS Debugger stops only for the breakpoint. z/OS Debugger does not stop twice (once for the AT GLOBAL OCCURRENCE and once for the test_level setting of ALL).

- You cannot use the AT GLOBAL ENTRY, AT GLOBAL EXIT, and AT GLOBAL PATH commands to stop at the entry or exit of a nested block in a C or C++ program. A nested block is a group of statements delimited by { and }. The compiler assigns a name to these blocks using the following pattern: %BLOCK*n*, where *n* is a sequentially-assigned number.

**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 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:

```
AT GLOBAL EXIT PERFORM
LIST TITLED (benefits, scale);
GO;
END-PERFORM;
```

- To set a GLOBAL DATE breakpoint, specify:

```
AT DATE *;
```

  or

```
AT GLOBAL DATE;
```

- To combine a global breakpoint with other z/OS Debugger commands, specify:

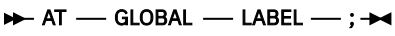
```
AT GLOBAL DATE QUERY LOCATION;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“every_clause syntax” on page 40
“PLAYBACK commands” on page 187

## AT GLOBAL LABEL command (remote debug mode)

Gives z/OS Debugger control for every instance of the specified AT-Label condition. Global breakpoints are always performed for all Compile Units that are known to z/OS Debugger.




Use AT GLOBAL LABEL breakpoints if you do not have specific information of where to set your breakpoint. For example, if you want to halt at a label that you do not know the name of, enter AT GLOBAL LABEL command to halt z/OS Debugger when it encounters a label.

**Usage note**

To set a global breakpoint, specify an asterisk (*) with the AT command, or enter the AT GLOBAL command.

**Example**

If you want to set a global AT LABEL breakpoint, specify one of the following commands:

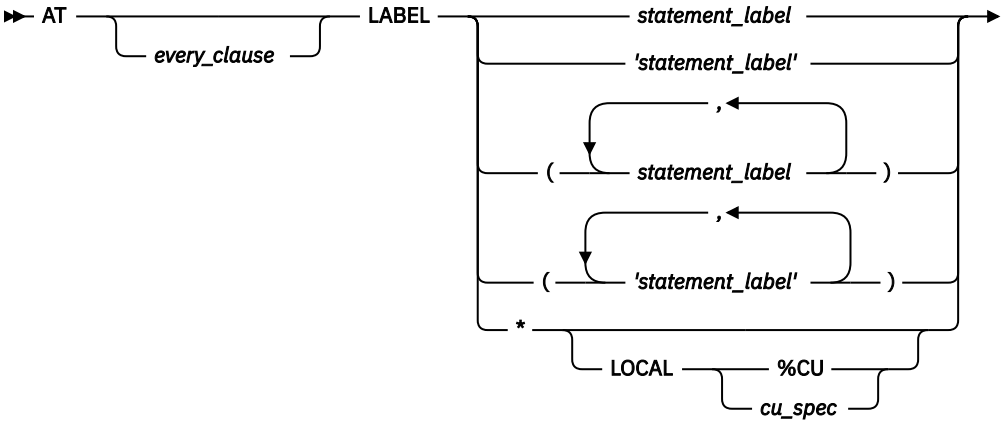
```
AT LABEL *;
```

or

```
AT GLOBAL LABEL;
```

## AT LABEL command

Gives z/OS Debugger 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 z/OS Debugger gains control at each label. For COBOL and LangX COBOL, AT LABEL lets you specify several labels, but for any group of labels that are associated with a single statement, z/OS Debugger gains control for that statement only once.




*****
Sets a breakpoint at every LABEL.

**LOCAL**
Specifies that the AT LABEL breakpoint is limited to all labels in the specified compile unit.

***cu_spec***
A valid compile unit specification.

***command***
A valid z/OS Debugger command.

**Usage notes**

- Use the syntax of *statement_label* enclosed in apostrophes (') only for LangX COBOL programs. It is not supported in any other programming language.
- z/OS Debugger 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 z/OS Debugger 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 z/OS Debugger, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on z/OS Debugger, 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.
  - 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. z/OS Debugger is entered for each specified label.
- To be able to set LABEL breakpoints in COBOL programs, you must compile your program with one of the following compilers and TEST compiler suboptions:
  - Specify the HOOK suboption with Enterprise COBOL for z/OS, Version 4
  - Specify the STMT, PATH, or ALL suboption and the SYM suboption with one of the following compilers:
    - any release of the Enterprise COBOL for z/OS and OS/390, Version 3, compiler
    - any release of the COBOL for OS/390 and VM, Version 2, compiler

  When defining specific LABEL breakpoints z/OS Debugger 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 OF or IN, but z/OS Debugger only uses OF 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 for z/OS, Version 3.4, or later, programs and you comply with the following requirements, you can use the AT LABEL command to set breakpoints (except at a label variable):
    - If you are running z/OS Version 1 Release 6, apply the Language Environment PTF for APAR PQ99039.
    - If you are compiling with Enterprise PL/I Version 3 Release 4, apply PTFs for APARs PK00118 and PK00339.
- 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.
- For Enterprise COBOL for z/OS Version 5 and later, AT LABEL * highlights the labels similar to statement breakpoints:
  - When you use AT LABEL *, a global label breakpoint is created. You can use PF6 or AT LINE to remove or recreate a hook at the label.
  - When you toggle hooks, the global label breakpoint created with AT LABEL * is saved in SAVEBPS, but the removal of any individual label hook is not saved.
  - The EVERY clause is not supported, which means highlighting and use of PF6 do not work.
  - PF6 only works in the currently running CU. Using SET QUALIFY CU LOAD::>CU shows highlighted labels if the AT LABEL * command was issued, but PF6 does not work.
  - Highlighting is disabled if PLAYBACK START is issued and enabled once again when PLAYBACK STOP is issued.

**Examples**

- Set a breakpoint at label `create` in the currently qualified block.

```
AT LABEL create;
```

- At program label `para OF sect1` display variable names x and y and their values, and continue program execution. The current programming language setting is COBOL.

```
AT LABEL para OF 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 */
```

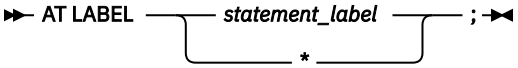
Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

“every_clause syntax” on page 40
“statement_label” on page 17
“PLAYBACK commands” on page 187

## AT LABEL command (remote debug mode)

Gives z/OS Debugger control when execution reaches the statement label that you specify. For C and PL/I programs, if multiple labels are associated with a single statement, z/OS Debugger gains control at each label that you set an AT LABEL breakpoint for. For COBOL programs, you can issue AT LABEL commands for multiple labels on the same statement, but for any group of labels that are associated with a single statement, z/OS Debugger gains control for that statement only once.




*****
Sets a breakpoint at every LABEL

**Usage notes**

- If you set a breakpoint for a specific label (for example, AT LABEL MYLABEL), and AT GLOBAL LABEL command is also set, the remote debugger stops only one time.
- z/OS Debugger does not support the AT LABEL command for VS COBOL II programs.
- A COBOL *statement_label* can have only the form of – *name*.
- A LABEL breakpoint in remote mode is limited to labels in the currently executing compile unit.
- For more information about restrictions for the AT LABEL command, see Usage notes on "AT LABEL command" on page 60

**Example**

Set a breakpoint at Label `create` in the currently qualified block.

```
AT LABEL create;
```

## AT LINE command

Gives z/OS Debugger 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.

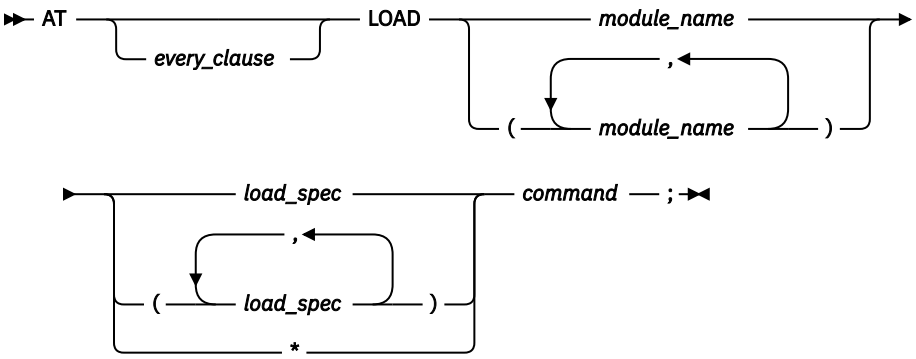
Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"AT OFFSET command (disassembly)" on page 68
"PLAYBACK commands" on page 187
"AT STATEMENT command" on page 70

## AT LOAD command

Gives z/OS Debugger control when the specified load module is brought into storage. For example, z/OS Debugger 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.




*****
Sets a breakpoint at every LOAD of any load module.

***command***
A valid z/OS Debugger command.

**Usage notes**

- z/OS Debugger 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, z/OS Debugger is not notified of any loads that are affected by the MVS LOAD service. Refer to *IBM z/OS Debugger 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 z/OS Debugger 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 z/OS Debugger, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on z/OS Debugger, it is cleared at the end of a process.
- AT LOAD on an implicitly or explicitly loaded DLL is not supported by z/OS Debugger.
- Depending on the version of the C or C++ compiler used, z/OS Debugger 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 z/OS Debugger to recognize `file2`, only `file1`. Similarly, a call to `fn2` **will not** enable z/OS Debugger to recognize `file1`.
- At the triggering of a LOAD breakpoint for C, C++, and PL/I, z/OS Debugger has enough information about the loaded module to set breakpoints and examine variables of static and extern storage classes.
- At the triggering of a LOAD breakpoint for COBOL, C, and C++ DLL's, z/OS Debugger 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 topics for more information related to the material discussed in this topic.

**Related references**
"every_clause syntax" on page 40
"load_spec" on page 15
"PLAYBACK commands" on page 187

## AT LOAD command (remote debug mode)

Gives z/OS Debugger control when the specified load module is brought into storage. For example, z/OS Debugger 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. 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.

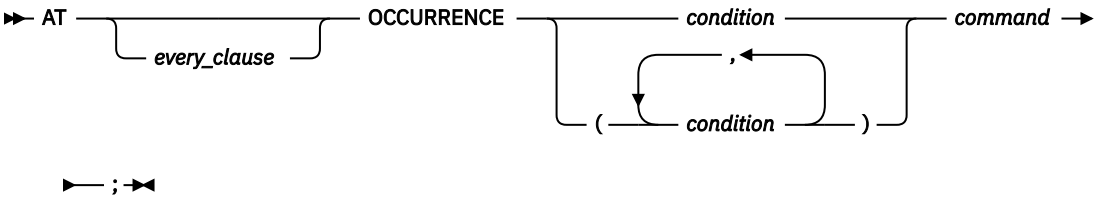
You can set LOAD breakpoints regardless of what compiler options are in effect.




**Related references**
"AT LOAD command" on page 63
Appendix A, "z/OS Debugger commands supported in Debug Tool compatibility mode," on page 517

# AT OCCURRENCE command

Gives z/OS Debugger control on a language or Language Environment condition or exception or an MVS or CICS ABEND.




***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 S*xxx* or U*xxx*, 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:

| | | |
|---|---|---|
| SIGABND | SIGILL | SIGTERM |
| SIGABRT | SIGINT | SIGUSR1 |
| SIGFPE | SIGIOERR | SIGUSR2 |
| | SIGSEGV | THROWOBJ |

When a C++ user specifies `AT CONDITION THROWOBJ`, z/OS Debugger transfers control to the user at the point of the throw in C++ code.

PL/I condition constants can be used. For conditions associated with file handling, the file reference can be a wildcard.

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.

***command***
A valid z/OS Debugger 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. z/OS Debugger is started before any C or C++ signal handler.
2. If you set an `OCCURRENCE` breakpoint for that condition, z/OS Debugger 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`, z/OS Debugger 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`), z/OS Debugger gets commands by prompting you or by reading from a commands file.
   - If the current test-level setting is `NONE`, z/OS Debugger 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 z/OS Debugger releases control, unless return is by use of `GO 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 z/OS Debugger, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on z/OS Debugger, it is cleared at the end of a process.
- For COBOL and LangX COBOL, z/OS Debugger detects Language Environment conditions. If a Language Environment condition occurs during your session, the following series of events takes place:

  1. z/OS Debugger is started before any condition handler.
  2. If you set an OCCURRENCE breakpoint for that condition, z/OS Debugger 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, z/OS Debugger 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, z/OS Debugger gets commands by prompting you or by reading from a commands file.
     - If the current test-level setting is NONE, z/OS Debugger 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, z/OS Debugger detects Language Environment and PL/I conditions. If a condition occurs, z/OS Debugger is started before any condition handler. If you have issued an ON command or set an OCCURRENCE breakpoint for the specified condition, z/OS Debugger runs the associated commands.
- If there is no AT OCCURRENCE or ON set, then:
  - If the current test-level setting is ALL, z/OS Debugger 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, z/OS Debugger gets commands by prompting you or by reading from a commands file.
  - If the current test-level setting is NONE, z/OS Debugger ignores the condition and returns control to the program.
- Once z/OS Debugger 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(OFF) 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(OFF) was specified for all abends except program checks. z/OS Debugger 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 S0C1 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).

- Stop for every file where ENDFILE condition occurs. The current programming language is PL/I.

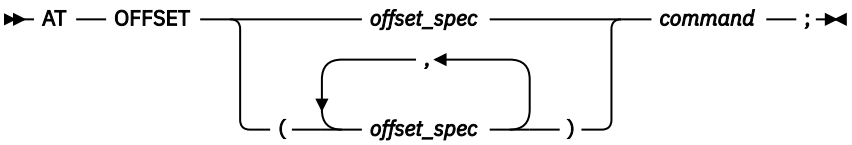
```
AT OCCURRENCE ENDFILE(*);
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“every_clause syntax” on page 40
“ON command (PL/I)” on page 181
“PLAYBACK commands” on page 187
*z/OS Language Environment Programming Guide*
*z/OS Language Environment Debugging Guide*
*PL/I for MVS and VM Language Reference*

## AT OFFSET command (disassembly)

Gives z/OS Debugger control at the specified offset in the disassembly view.




***command***
A valid z/OS Debugger command.

**Usage note**

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:

```
AT OFFSET X'2A';
```

- Set a breakpoint at offsets '2A' and '30' in the current block:

```
AT OFFSET (X'2A',X'30');
```

- Set a breakpoint in the block MYPROG at offset '3A':

```
AT OFFSET MYPROG:>X'3A';
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“PLAYBACK commands” on page 187
“offset_spec” on page 15

## AT PATH command

Gives z/OS Debugger control when the flow of control changes (at a path point). AT PATH is identical to AT GLOBAL PATH.


***command***
A valid z/OS Debugger command.

**Usage notes**

- For Enterprise COBOL for z/OS Version 5, when z/OS Debugger stops at an AFTER CALL Path point because of an AT PATH breakpoint, the location where z/OS Debugger stops is the statement after the CALL statement.
- For a CICS application on z/OS Debugger, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on z/OS Debugger, 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.
  - 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 COBOL programs compiled with the following compilers, compile your program with the NONE, PATH, or ALL suboption of the TEST compiler option to be able to set PATH breakpoints:
  - Enterprise COBOL for z/OS and OS/390, Version 3
  - COBOL for OS/390 and VM, Version 2
- For PL/I, to be able to set PATH breakpoints, you must compile your program with the PATH or ALL suboption of the TEST compiler option.
- You cannot use the AT PATH command while you replay recorded statements by using the PLAYBACK commands.
- z/OS Debugger does not support the AT PATH command while you debug a disassembled program or a VS COBOL II program.
- You cannot use the AT PATH command to stop at the entry or exit of a nested block in a C or C++ program. A nested block is a group of statements delimited by { and }. The compiler assigns a name to these blocks using the following pattern: %BLOCK*n*, where *n* is a sequentially-assigned number.

**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;
```

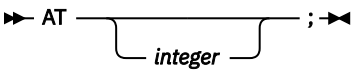
Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
*IBM z/OS Debugger User's Guide*

**Related references**
"every_clause syntax" on page 40
"%PATHCODE" on page 341
"PLAYBACK commands" on page 187

# AT Prefix command (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
```

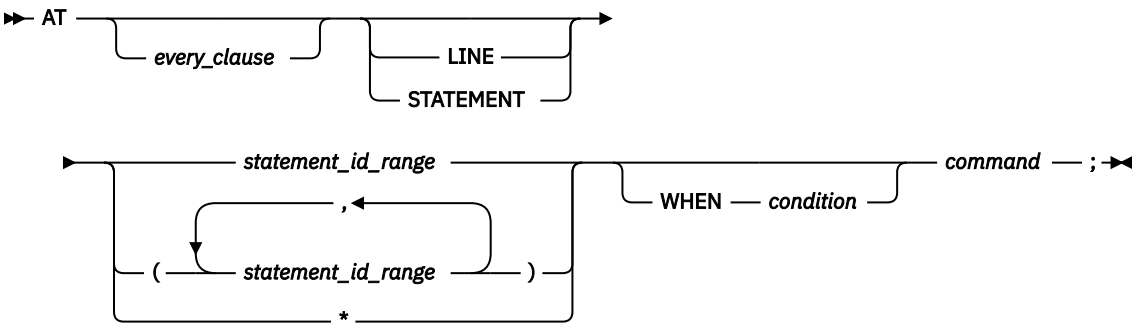
No space is needed as a delimiter between the keyword and the integer; hence, `AT 3` is equivalent to AT3.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"PLAYBACK commands" on page 187

# AT STATEMENT command

Gives z/OS Debugger control at each specified statement or line within the given set of ranges.




*****
Sets a breakpoint at every STATEMENT or LINE.

***command***
A valid z/OS Debugger command. If you are using remote debug mode, you can specify only commands that are supported in remote debug mode.

***condition***
A valid z/OS Debugger conditional expression.

**Usage notes**

- With Enterprise COBOL for z/OS Version 5, you cannot set AT STATEMENT breakpoints for statements that are inside a declarative section.
- With Enterprise COBOL for z/OS Version 5, you can set AT STATEMENT breakpoints for the WHEN and EVALUATE statements.
- 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 z/OS Debugger, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on z/OS Debugger, it is cleared at the end of a process.
- You can specify the first relative statement on each line in any one 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, z/OS Debugger 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.
- You can restrict the circumstances under which the AT STATEMENT break point is raised by specifying a WHEN condition. If a WHEN condition is specified, z/OS Debugger stops at the AT STATEMENT break point if the specified statement matches the current statement and the WHEN condition is true.

- 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 STATEMENT` command with a `WHEN` condition, every time z/OS Debugger reaches the statement, it evaluates the condition. If the condition evaluates to true, z/OS Debugger stops and runs the command associated with the breakpoint.
- z/OS Debugger evaluates references in a `WHEN` condition *before* it runs a statement.
- When z/OS Debugger evaluates the condition and the condition is invalid, z/OS Debugger does one of the following actions:
  - If `SET WARNING` is set to `ON`, z/OS Debugger stops and displays a message that it could not evaluate the condition. You need to enter a command to indicate what action you want z/OS Debugger to take.
  - If `SET WARNING` is set to `OFF`, z/OS Debugger does not stop nor display a message that it could not evaluate the condition. z/OS Debugger continues running the program.

**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);
```

- To set a breakpoint at statement or line 100 that is raised only when the value of `myvar` is equal to 100, enter the following command:

```
AT 100 WHEN myvar=100;
```

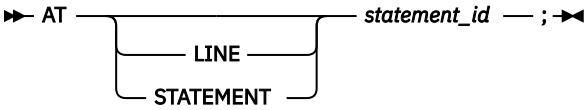
Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“every_clause syntax” on page 40
“statement_id_range and stmt_id_spec” on page 16

"AT OFFSET command (disassembly)" on page 68
"PLAYBACK commands" on page 187
Appendix A, "z/OS Debugger commands supported in Debug Tool compatibility mode," on page 517

## AT STATEMENT command (remote debug mode)

Gives z/OS Debugger control at the specified statement or line.




### Usage note

When you enter an AT STATEMENT command, the breakpoint is set relative to the location the program is stopped, which might not be the program displayed in the source view. For example, your program is stopped at program SUB1, which was called by program MAIN1, and the source view displays the source for program SUB1. Then, you click on MAIN1 in the Debug view so that the source view displays the source for MAIN1. If you enter the command AT STATEMENT 13, a breakpoint is set at statement 13 in SUB1, not statement 13 in MAIN1.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"statement_id" on page 16
"AT STATEMENT command" on page 70
Appendix A, "z/OS Debugger commands supported in Debug Tool compatibility mode," on page 517

# AT TERMINATION command

Gives z/OS Debugger control when the application program is terminated.


***command***
A valid z/OS Debugger 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 z/OS Debugger gains control, normal execution of the program is complete; however, a CALL or function invocation from z/OS Debugger can continue to perform program code. When the AT TERMINATION breakpoint gives control to z/OS Debugger:
  - 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 z/OS Debugger, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on z/OS Debugger, 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 z/OS Debugger 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 z/OS Debugger session. The current programming language setting is C.

```
AT TERMINATION {
  LIST "Program has ended";
  QUIT;
}
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

“PLAYBACK commands” on page 187

# BEGIN command

`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 z/OS Debugger command.

**Usage notes**

- The `BEGIN` command is most helpful when used in `AT` or `PROCEDURE` commands.
- The `BEGIN` command is helpful when you use it as a programming language neutral command. For example, if you create a commands file that might be used by an application created with several different programming languages, the `BEGIN` command works for all supported programming languages.
- For Enterprise PL/I, the `BEGIN` command is helpful when used in `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.

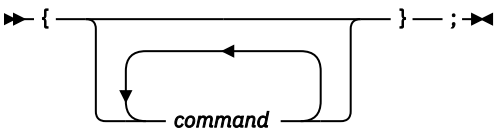
```
AT 320 BEGIN;
  LIST (x);
  a = 2;
END;
```

- 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 z/OS Debugger commands into one command. When you enclose z/OS Debugger 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.




***command***
A valid z/OS Debugger 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.

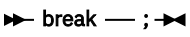
**Example**

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.




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 z/OS Debugger session. The CALL keyword cannot be abbreviated.

In C++, calls can be made to any user function provided that the function is declared with the following syntax:

```
extern "C"
```

In COBOL, the CALL command cannot be issued when z/OS Debugger is at initialization.

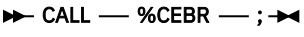
The following table summarizes the forms of the CALL command.

| Command | Description |
|---|---|
| “CALL %CEBR command” on page 77 | Starts the CICS Temporary Storage Browser Program. |
| “CALL %CECI command” on page 77 | Starts the CICS Command Level Interpreter Program. |
| “CALL %DUMP command” on page 77 | Calls a dump service to obtain a formatted dump. |
| “CALL %FA command” on page 82 | Calls IBM Fault Analyzer to provide a formatted dump of the current machine state. |
| “CALL %HOGAN command” on page 82 | Starts Computer Sciences Corporation's KORE-HOGAN application. |
| “CALL %VER command” on page 83 | Adds a line to the log describing the maintenance level of z/OS Debugger that you have installed on your system. |
| “CALL entry_name command (COBOL)” on page 83 | Calls an entry name in the application program (COBOL). |

| Command | Description |
|---|---|
| "CALL procedure command" on page 84 | Calls a procedure that has been defined with the PROCEDURE command. |

## CALL %CEBR command

Starts the CICS Temporary Storage Browser Program.




### Usage notes

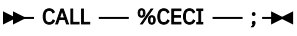
- z/OS Debugger performs an EXEC CICS LINK to the CICS browser program. When CEBR processing is complete, control is returned to z/OS Debugger through an EXEC CICS return.
- You can use this command only when you debug CICS programs in single-terminal mode in full-screen mode.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
*CICS Supplied Transactions*
*CICS Application Programming Guide*

## CALL %CECI command

Starts the CICS Command Level Interpreter Program.




### Usage notes

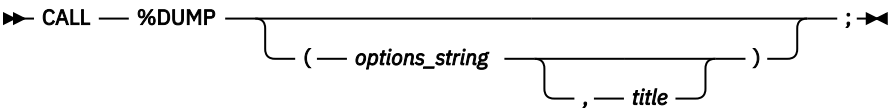
- z/OS Debugger performs an EXEC CICS LINK to the CICS command level interpreter program. When CECI processing is complete, control is returned to z/OS Debugger through an EXEC CICS return.
- You can use this command only when you debug CICS programs in single-terminal mode in full-screen mode.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
*CICS Supplied Transactions*
*CICS Application Programming Guide*

## CALL %DUMP command

Calls a dump service to obtain a formatted dump.




***title***
Specifies the identification printed at the top of each page of the dump. It must be a fixed-length character string. It must conform to the syntax rules for a character string constant enclosed in quotation marks (") or apostrophes (') for the current programming language. The string length cannot exceed 80 bytes.

***options_string***
A fixed-length character string that specifies the type, format, and destination of dump information. The string must conform to the syntax rules for a character string constant enclosed in quotation

marks (") or apostrophes (') for the current programming language. 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 z/OS Debugger 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:

| Language | Compiler option |
|---|---|
| C | `TEST(SYM)` |
| C++ | `TEST` |
| COBOL | `TEST` or `TEST(h,SYM)` |
| PL/I | `TEST(,SYM)` |

The variables, arguments, and registers are dumped starting with z/OS Debugger. 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 z/OS Debugger. 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 z/OS Debugger 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 z/OS Debugger. 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 z/OS Debugger, 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.

`CONDITION` can be abbreviated as `COND`.

**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 z/OS Debugger 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 z/OS Debugger issues an `EXEC CICS DUMP TRANSACTION`.

**NOENTRY**
Suppresses the description of the z/OS Debugger 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 z/OS Debugger 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: z/OS Debugger 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 z/OS Debugger 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 DD names must be allocated for you to receive a formatted dump:
  - CEEDUMP (default)
  - SYSPRINT.

  Control might not be returned to z/OS Debugger 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 z/OS Debugger 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 5. %DUMP options mapping to SNAP options*

| %DUMP option | SNAP option |
|---|---|
| THREAD | ignored |
| TRACEBACK | SDATA=(PCDATA),PDATA=(SA,SAH) |
| FILES | SDATA=(DM,IO) |
| VARIABLES | SDATA=(CB) |
| BLOCKS | SDATA=(SQA,LSAQ,SWA) |
| STORAGE | PDATA=(LPA,JPA,SPLS) |
| STACKFRAME | ignored |
| PAGESIZE | ignored |
| FNAME | ddname for dump |
| CONDITION | SDATA=(Q,TRT,ERR) |
| ENTRY | PDATA=(SUBTASKS) |

- 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.

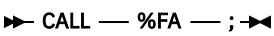
```
CALL %DUMP ("TRACEBACK FILES");
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"PLAYBACK commands" on page 187
*z/OS Language Environment Programming Guide*
*z/OS Language Environment Debugging Guide*

## CALL %FA command

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




**Usage notes**

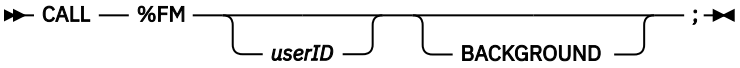
- If you are replaying recorded statements by using the PLAYBACK commands, CALL %FA provides a formatted dump of the machine state when you entered PLAYBACK START.
- You can use this command in remote debug mode.
- This command does not support 64-bit programs.

Refer to the following topics for more information related to the material discussed in this topic.

- Appendix A, "z/OS Debugger commands supported in Debug Tool compatibility mode," on page 517

## CALL %FM command

Starts IBM File Manager for z/OS.




***userID***
The ID of an MVS user. If you do not specify a *userID*, then File Manager takes one of the following options:

- If you sign on using CESN and File Manager has been installed with either *DEFAULT=SIGNON or *PASSWORD=REMEMBER, then *userID* is assigned the user ID used to sign on.
- If you have not signed on, then File Manager prompts you for a user ID before it displays the logon panel.

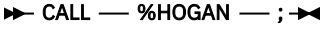
**BACKGROUND**
Specifies that all non-terminal processing be routed to a background task.

**Usage notes**

- You can use this command only when you debug CICS programs.
- You need to have IBM File Manager for z/OS V9R1 installed in the CICS region.

## CALL %HOGAN command

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

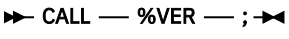


**Usage notes**

- 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 command

Adds a line to the log describing the maintenance level of z/OS Debugger that you have installed on your system.




**Usage note**

You can use this command in remote debug mode.

**Example**

You have z/OS Debugger, Version 14.2, installed on your system. Enter the `CALL %VER` command to display the following information in the Log window:

```
IBM z/OS Debugger 15.0.n
07/22/2020 09:33:00 AM Level: 15.0.n PHnnnnn
5724-T07: Copyright IBM Corp. 1992, 2020
```

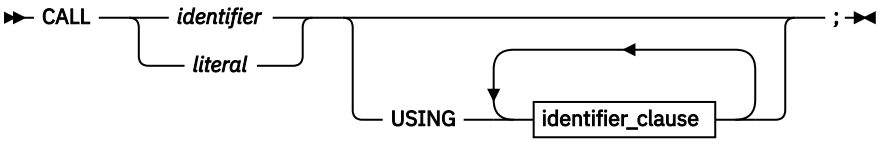
The time stamp that is shown is the product build date and time.

Refer to the following topics for more information related to the material discussed in this topic.

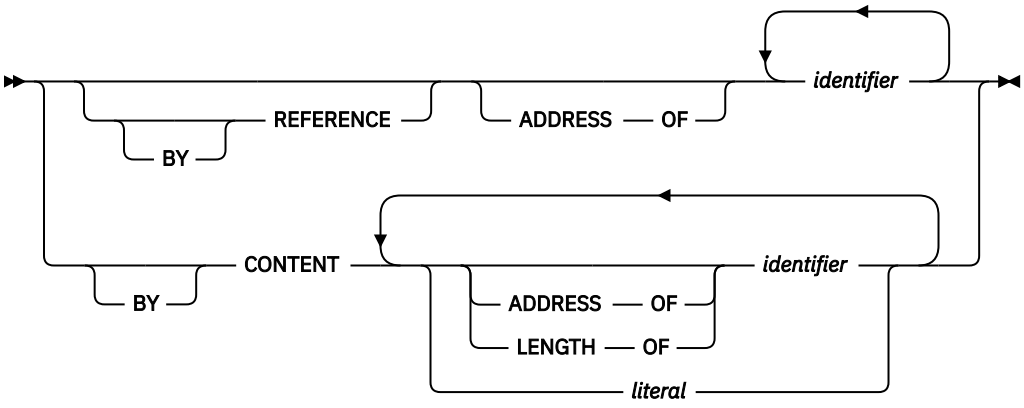
- Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

## CALL entry_name command (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).




**identifier_clause**




***identifier***
A valid z/OS Debugger COBOL identifier.

***literal***
A valid COBOL literal.

**Usage notes**

- If you have a COBOL entry point name that is the same as a z/OS Debugger procedure name, the procedure name takes precedence when using the CALL command. If you want the entry name to take precedence over the z/OS Debugger 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 z/OS Debugger 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 topics for more information related to the material discussed in this topic.

**Related references**
“PLAYBACK commands” on page 187
*Enterprise COBOL for z/OS Language Reference*

## CALL procedure command

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


***procedure_name***
The name given to a sequence of z/OS Debugger commands delimited by a PROCEDURE command and a corresponding END command.

**Usage notes**

- Because the z/OS Debugger 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.

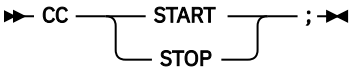
**Example**

Create and call the procedure named `proc1`.

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

# CC command

Controls whether code coverage data is collected.




**Usage notes**

- The `CC START` command collects data for the following compile unit or programs:
  - The currently qualified z/OS Debugger compile unit from the point in the program where the command is entered.
  - Programs that are run after the `CC START` command is issued and that are selected by a user-specified action. This action can be stepping into a compile unit, setting a breakpoint in a compile unit, or defining a compile unit in the DTCN or CADP profile.
- `CC STOP` deletes all code coverage data.
- To view the code coverage information generated by `CC START`, issue `LIST CC` before entering `CC STOP`.
- The collection of code coverage data can add a substantial amount of overhead. Therefore, it is a good practice to issue the `CC START` command only when you want to gather this data. Do not routinely issue the `CC START` command in debug sessions in which you do not want to gather this data.

**Examples**

- Specify that code coverage data be collected.

```
CC START;
```

- List the code coverage data.

```
LIST CC;
```

- Specify that code coverage stop and the data be deleted.

```
CC STOP;
```

**Related references**

"LIST CC command" on page 145

# CHKSTGV command

Checks whether the CICS storage check zone of a user-storage element has been overlaid.


**Usage notes**

- This command applies only to CICS applications.
- You can use this command in remote debug mode.
- Do not use this command to replace the practices described in *CICS Problem Determination Guide* in the section *Dealing with storage violations*.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

*CICS Problem Determination Guide*

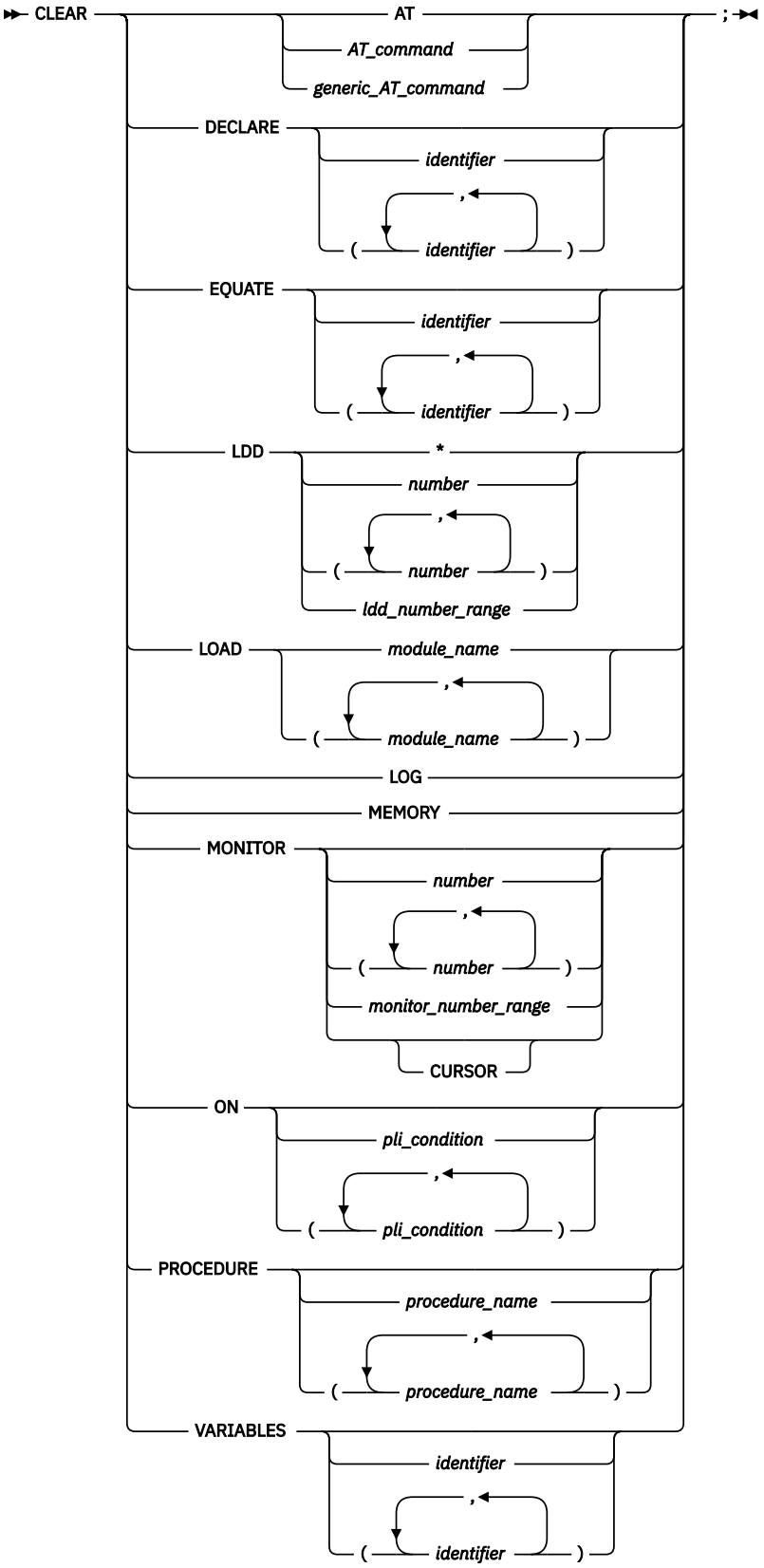
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

**Related tasks**

"Detecting CICS storage violations early" in the IBM z/OS Debugger User's Guide

# CLEAR command

The CLEAR command removes the actions of previously entered z/OS Debugger commands. Some breakpoints are removed automatically when z/OS Debugger 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.




**AT**

Removes all breakpoints, including GLOBAL breakpoints, set by previously entered AT commands, except for AT TERMINATION and suspended 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 z/OS Debugger 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 z/OS Debugger session using SET EQUATE. This operand must follow the rules for the current programming language.

**LDD**
Removes one or more LOADDEBUGDATA (LDD) commands known to z/OS Debugger. The LDD command's sub-parameter must be one of those listed in the output of the LIST LDD command. It is recommended that you enter the LIST LDD command before each CLEAR LDD command because the LDD entry numbers are affected by previous CLEAR LDD commands. This command has the following sub-parameters:

*****
Removes all LDD commands known to z/OS Debugger across all enclaves.

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

***ldd_number_range***
Identifies the first and last number as seen in the LIST LDD command's output, separated by a hyphen (-), that you want to clear. When the current programming language setting is COBOL, blanks are required around the hyphen (-). Blanks are optional for other programming languages. However, in remote debug mode, blanks are required around the hyphen (-) for all programming languages.

**Usage note**

You can use the CLEAR LDD command in remote debug mode.

**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 z/OS Debugger 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.

**MEMORY**
Clears the Memory window including the memory currently being displayed, the base address, and the history area.

**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.

***monitor_number_range***
Identifies the first and last monitor number in a range of monitors, separated by a hyphen (-), that you want to delete. When the current programming language setting is COBOL, blanks are required around the hyphen (-). Blanks are optional for other programming languages.

**CURSOR**
Indicates that you want to delete the variable identified by the cursor's current location. The cursor can be placed only in the Monitor window.

**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 z/OS Debugger procedures. If no *procedure_name* follows PROCEDURE, all inactive procedures are cleared.

***procedure_name***
The name given to a sequence of z/OS Debugger 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 z/OS Debugger 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.
- 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 enter CC prefix commands to clear a selected block of lines from 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 z/OS Debugger 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 LDD, CLEAR ON, and CLEAR VARIABLES commands cannot be used while you replay recorded statements by using the PLAYBACK commands.

- To use the cursor to indicate which variable in the Monitor window to remove, do one of the following methods:
  - Assign the CLEAR MONITOR CURSOR to a PF key. Move the cursor to a variable in the Monitor window and press the PF key. This method is more convenient.
  - Type the CLEAR MONITOR command on the command line, then move the cursor to a variable in the Monitor window. Press Enter.
- Based on the application flow and structure, the CLEAR LDD command might not take effect until the next z/OS Debugger session is started.
- The CLEAR LDD * command removes all LDD commands known to z/OS Debugger across all enclaves.
- Because the SAVEBPS data set is updated during each enclave exit, if at any time the CLEAR LDD command is issued afterwards, the LDD commands will have already been saved in the SAVEBPS data set and thus will be restored during the next debug session.
- The SET EXPLICITDEBUG ON command takes precedence over the CLEAR LDD command. As a result, even though the CLEAR LDD command is processed, it will not undo the already processed LDD command.

**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 *;
```

- The following examples show how to display the LDD commands known to z/OS Debugger and how to use the CLEAR LDD command:

– To display the LDD commands known to z/OS Debugger, specify:

```
LIST LDD;
```

Suppose that you get the following output:

```
1. LDD TBND003::>TBND003A;
2. LDD MYPROG;
3. LDD MYPROG3;
4. LDD PROG4::>PROG5;
```

To remove all the LDD commands, specify:

```
CLEAR LDD *;
```

If you then enter the following command:

```
LIST LDD;
```

You will get the following result:

```
There are no LDD commands established.
```

– To display the LDD commands known to z/OS Debugger, specify:

```
LIST LDD;
```

Suppose that you get the following output:

```
1. LDD 1A::>1AB;
2. LDD PGM1C;
```

To remove the LDD 1A::>1AB command, specify:

```
CLEAR LDD 1;
```

– To display the LDD commands known to z/OS Debugger, specify:

```
LIST LDD;
```

Suppose that you get the following output:

```
1. LDD TBND005::>TBND005A;
2. LDD MYPROG;
3. LDD MYPROG5;
4. LDD PROG5::>PROG5Y;
```

If you then enter the CLEAR LDD 5 command, you will get the following output:

```
No LDD command was established for LDD 5.
```

– To display the LDD commands known to z/OS Debugger, specify:

```
LIST LDD;
```

Suppose that you get the following output:

```
1.    LDD TBND003::>TBND003A;
2.    LDD MYPROG;
3.    LDD MYPROG3;
4.    LDD PROG3::>PROG3C;
```

If you then enter the CLEAR LDD (1,4) command, you will get the following output:

```
Removes LDD TBND003::>TBND003A and LDD PROG3::>PROG3C
```

– To display the LDD commands known to z/OS Debugger, specify:

```
LIST LDD;
```

Suppose that you get the following output:

```
1.    LDD TBND003::>TBND003A;
2.    LDD MYPROG;
3.    LDD MYPROG3;
4.    LDD PROG6::>PROG6F;
```

If you then enter the CLEAR LDD 4 – 5 command (for COBOL or all languages in remote debug mode), you will get the following output:

```
No LDD command was established for LDD 5.
```

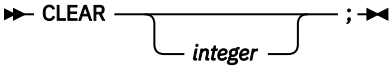
However, z/OS Debugger removes the LDD PROG6::>PROG6F command.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“CLEAR prefix (full-screen mode)” on page 92
“AT command” on page 37
“LIST command” on page 140
“PLAYBACK commands” on page 187
“Prefix commands (full-screen mode)” on page 192
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

# 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.




***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.
- Use CC in the Monitor window prefix area to clear a selected block of lines from the Monitor window.

**Examples**

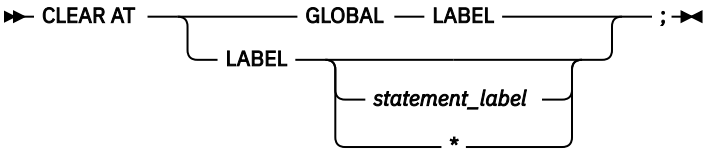
- In the Source window, clear a breakpoint at the third statement or verb in the line (typed in the prefix area of the line where the statement is found).

  ```
  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.

# CLEAR AT command (remote debug mode)

You can use the CLEAR AT command to remove actions that were completed by using the AT GLOBAL LABEL or the AT LABEL commands.

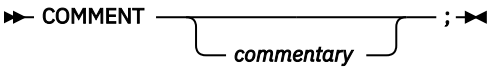



### Usage note

To clear a global breakpoint, specify an asterisk (*) with the CLEAR AT LABEL command, or specify a CLEAR AT GLOBAL LABEL command.

# COMMENT command

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




***commentary***
Commentary text not including a semicolon. An embedded semicolon is not allowed; text after a semicolon is treated as another z/OS Debugger command. DBCS characters can be used within the commentary.

### Usage notes

- You can use the COMMENT command in remote debug mode by entering it in the Debug Console or the **Action** field, which is in the **Optional Parameters** section of the **Add a Breakpoint** task.

### 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 z/OS Debugger commands.

```
COMMENT Entering subroutine testrun; LIST (x); GO;
```

The COMMENT command can be used as an executable command, but it is treated as a Null command and no output is produced. For example, there will be no output of the COMMENT command in the following cases:

- When it is specified as a command to be executed as an action of another command. For example:

```
AT 10 COMMENT xxx;
```

- When it is used inside of any command that allows one to specify a sequence of commands such as DO/END, BEGIN/END, or PERFORM/END-PERFORM.
- When it is used inside of a PROCEDURE command.

To get output in these cases, use the LIST command instead of the COMMENT command. For example:

```
AT 10 LIST 'xxx';
```

# COMPUTE command (COBOL)

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




***reference***
A valid z/OS Debugger COBOL numeric reference.

***expression***
A valid z/OS Debugger 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 z/OS Debugger 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.
- 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 z/OS Debugger 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);
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

"MOVE command (COBOL)" on page 175
"PLAYBACK commands" on page 187
"SET WARNING command (C, C++, COBOL, and PL/I)" on page 263

# CURSOR command (full-screen mode)

The CURSOR command moves the cursor between the last saved position on the z/OS Debugger session panel (excluding the header fields) and the command line.


### 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.

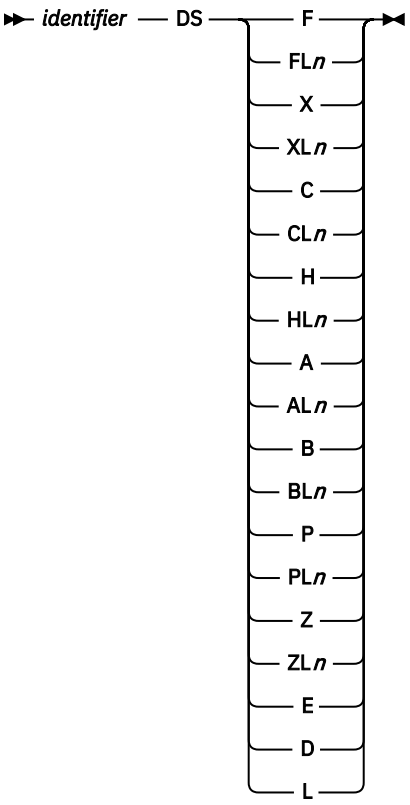
### Example

Move the cursor between the last saved position on the z/OS Debugger session panel and the command line.

```
CURSOR;
```

# Declarations (assembler, disassembly, and LangX COBOL)

Use declarations to declare session variables that are effective during a z/OS Debugger 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 z/OS Debugger commands as if they were declared to the compiler. Declared variables are removed when your z/OS Debugger session ends or when the CLEAR command is used to remove them.




***identifier***
A valid assembler identifier.

**F, FL*n*, X, XL*n*, C, CL*n*, H, HL*n*, A, AL*n*, B, BL*n*, P, PL*n*, Z, ZL*n*, 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**

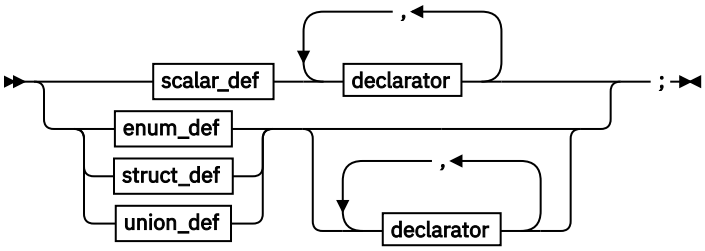
The range of valid *n* values depends on the type specifier as follows:

- C and X: 1 to 65525
- F, H, and A: 1 to 4
- B: 1 to 256
- P and Z: 1 to 16

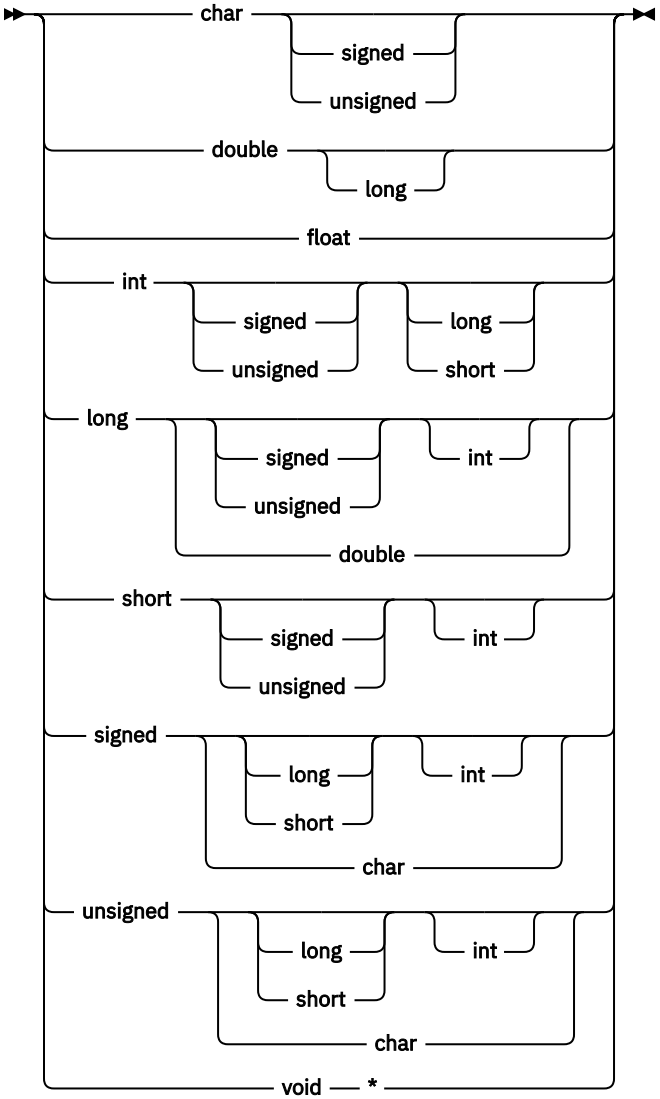
# Declarations (C and C++)

Use `declarations` to declare session variables and tags that are effective during a z/OS Debugger 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 z/OS Debugger commands as if they were declared to the compiler. Declared variables and tags are removed when your z/OS Debugger session ends or when the CLEAR command is used to remove them. The keywords must be the correct case and cannot be abbreviated.

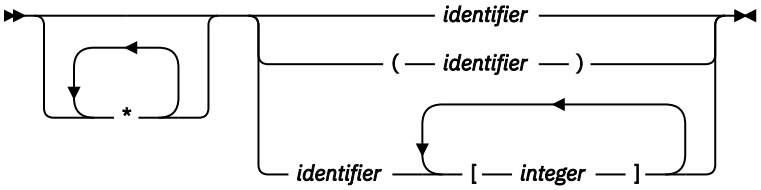
You can also declare `enum`, `struct`, and `union` data types. The syntax is identical to C except that `enum` members can only be initialized to an optionally signed integer constant.



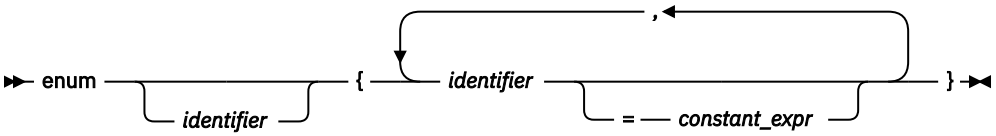

**scalar_def**



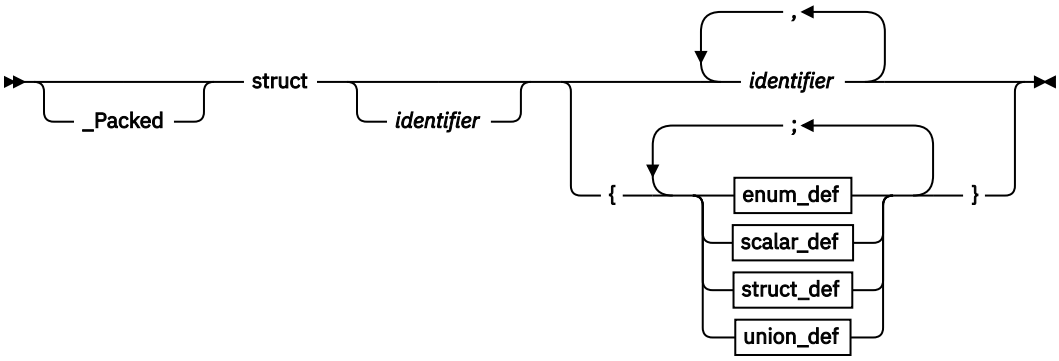

**declarator**



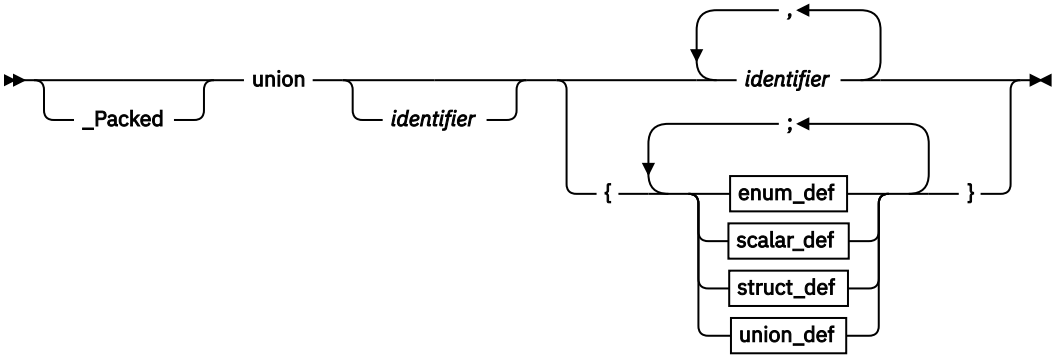

**enum_def**




**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 var1;
} avar;
```

- Interactively declare variables using structure tags.

```
struct tagone {int a; int b;} c;
```

  then specify:

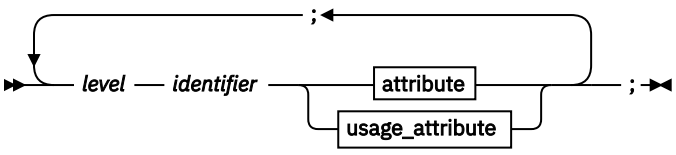
```
struct tagone d;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
*IBM z/OS Debugger User's Guide*

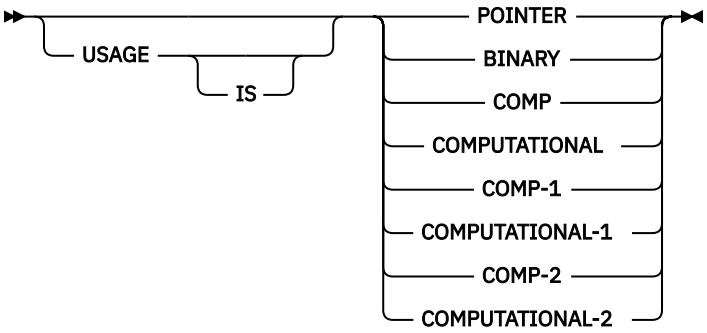
# Declarations (COBOL)

Use declarations to declare session variables that are effective during a z/OS Debugger 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 z/OS Debugger commands as if they were declared to the compiler. Declared variables are removed when your z/OS Debugger session ends or when the CLEAR command is used to remove them. The keywords cannot be abbreviated.




**attribute**


**usage_attribute**




***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**

- For Enterprise COBOL for z/OS Version 5, if you declare a session variable by using the attribute `UNSIGNED BINARY`, it can be used only when the current qualification is an Enterprise COBOL for z/OS Version 5 program.
- For Enterprise COBOL for z/OS Version 5, it enforces COBOL rules for variable names when session variables are declared. Version 4 allows some invalid names to be used. Some examples are as follows:
  - For Version 5, it does not allow the name "4-44"; however, the name is allowed in Version 4. The name is invalid because COBOL requires at least one alphabetical character in a variable name.
  - For Version 5, it does not allow the name "SV12#"; however, the name is allowed in Version 4. The name is invalid because '#' is not allowed. Only '-', '_', and alphanumerical characters are allowed in a COBOL variable name.
  - For Version 5, it does not allow the name "_SV12"; however, the name is allowed in Version 4. The name is invalid because '_' cannot be used as the first character in a variable name.
- For Enterprise COBOL for z/OS Version 5, `COMP-4` and `COMPUTATIONAL-4` are also accepted.
- 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.

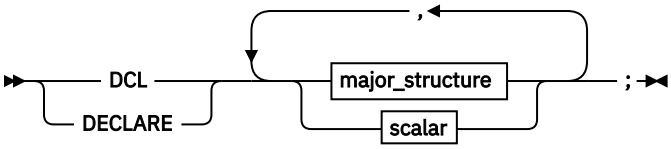
```
77 temp PIC S9(9) USAGE COMP;
```

Refer to the following topics for more information related to the material discussed in this topic.

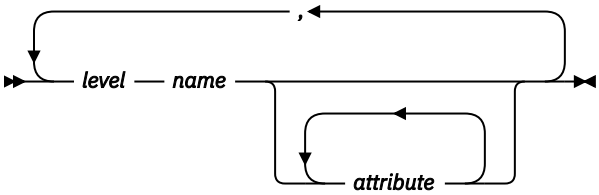
**Related tasks**
*IBM z/OS Debugger User's Guide*
**Related references**
*Enterprise COBOL for z/OS Language Reference*

# DECLARE command (PL/I)

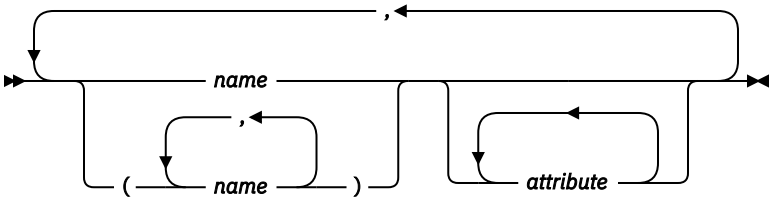
The DECLARE command declares session variables that are effective during a z/OS Debugger session. Variables declared this way can be used in other z/OS Debugger commands as if they were declared to the compiler. They are removed with the CLEAR command or when your z/OS Debugger session ends. The keywords cannot be abbreviated.




**major_structure**




**scalar**




***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, z/OS Debugger 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 z/OS Debugger 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        CPLX       FIXED      LABEL      PTR
BIT           DECIMAL    FLOAT      OFFSET     REAL
CHARACTERS    EVENT      GRAPHIC    POINTER    VARYING
COMPLEX
```

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 cannot 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 topics for more information related to the material discussed in this topic.

**Related tasks**

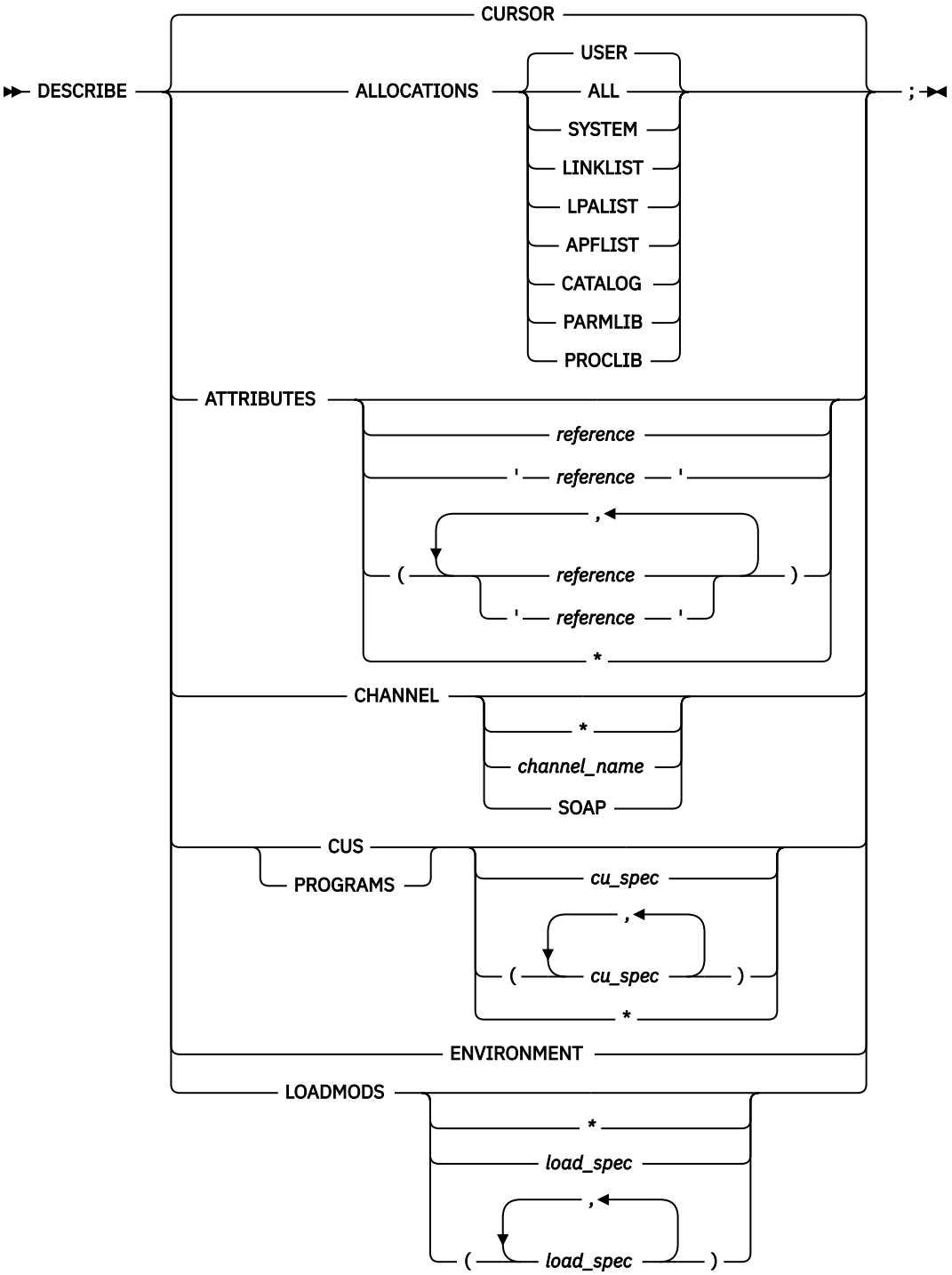
*IBM z/OS Debugger User's Guide*

Refer to the following topics for more information related to the material discussed in this topic.

**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, the run-time environment, and CICS channels and containers.



**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 z/OS Debugger 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 substringed 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 z/OS Debugger LangX COBOL reference. This form must be used for LangX 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 LangX COBOL programs.

**CHANNEL**
Describes CICS channels and containers, including containers that hold Web services state data. You can specify one of the following suboptions:

***channel_name***
Describe all containers in the channel *channel_name*.

*****
Describe all the containers in all the channels in the current scope.

**SOAP**
Describe all SOAP containers. SOAP is a synonym for DFHNODE.

If you do not specify a suboption, z/OS Debugger lists all of the containers in the current channel.

**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.

***cu_spec***
The name of the compile unit whose attributes you want to list.

*****
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, LangX COBOL, assembler, and disassembly do not provide any information for `DESCRIBE ENVIRONMENT`.

**LOADMODS**
This command displays information about load modules known to z/OS Debugger 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 z/OS Debugger along with the address, length, entry point, and the dataset from which the module was loaded.

***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**

- For Enterprise COBOL for z/OS Version 5, the output of `DESCRIBE ATTRUBUTES` for `RENAMES` data items shows `PIC X` instead of `AN-GR`.
- For Enterprise COBOL for z/OS Version 5, If two or more level 01 or 77 data items have the same name, `DESCRIBE ATTRIBUTES` with no operand displays an error message when you attempt to show the attributes of those data items.
- For Enterprise COBOL for z/OS Version 5, the output of `DESCRIBE ATTRIBUTES` for a level 88 variable does not show an address.
- If you use the `DESCRIBE ATTRIBUTES` command without specifying any data item, it shows the attributes of all data items defined in the currently qualified block. The output of this command is changed for Enterprise COBOL for z/OS Version 5 in the following ways:
  - For records, the output shows only the high-level attributes of the record, such as length and address. The output does not show the attributes of each subordinate group or data item defined within the record. This reduces the amount of the output produced. For Enterprise COBOL for z/OS Version 4, it also shows the attributes of all subordinate data items within each record or group, that is, the entire

data hierarchy. To see this level of detail in Enterprise COBOL for z/OS Version 5, you can specify a particular data item on the DESCRIBE ATTRIBUTES command. If the data item is a record or group, it shows the attributes of all subordinate data items within that record or group.

  - For data items that are not records, which are scalar data items, the type of the data item is no longer displayed on a separate line in the output as it was in Enterprise COBOL for z/OS Version 4, but instead it is shown after the data item name on the line that includes "ATTRIBUTES for". This further reduces the number of lines of the output produced, and makes the output for scalar data items more consistent with the output for records.

- You can use the DESCRIBE CUS, DESCRIBE CHANNEL, and DESCRIBE LOADMODS commands 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 6, Version 1 Release 7, and Version 1 Release 8.
- For Enterprise COBOL for z/OS Version 5, the PIC definition and other attributes of a variable are displayed as declared in the program.
- For Enterprise COBOL for z/OS Version 5, the result of issuing DESCRIBE ATTRIBUTES for a z/OS Debugger variable that represents a register does not include an address. For example, DESCRIBE ATTRIBUTES %GPR15.
- For Enterprise COBOL for z/OS Version 5, the output of DESCRIBE ATTRIBUTES for a record or a group variable is displayed with the levels as declared in the program.
- LangX COBOL PIC attributes might not match the original PIC specification in the following situations:
  - A COMP-3 variable always has an odd number of digits in its PIC value.
  - All non-numerical strings 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.
- The DESCRIBE LOADMODS command does not display information about load modules or compile units provided by operating system, subsystem, or runtime software (for example: MVS, CICS, Db2, IMS, and Language Environment) because z/OS Debugger ignores these modules.
- The DESCRIBE LOADMODS command cannot display the DSNAME of load modules loaded by LPA, LLA, AOS loader, or an unknown provider because the DSNAME for these providers is not available.
- 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)
  - A CU name shown as a load module name followed by ">" indicates the entry point CU for a load module that is the target of an AT LOAD command.
- You can use the DESCRIBE CHANNEL command only if your application program runs on CICS Transaction Server Version 3.1 or later.
- For PL/I, COBOL, LangX COBOL, assembler, and disassembly, if a channel name is mixed case, you must enclose it in quotation marks (") or apostrophes ('). If you do not enclose it in quotation marks or apostrophes, z/OS Debugger converts it to all upper case.
- For C and C++, all channels names are case sensitive. The following table compares how the same command must be typed differently, depending on the programming language you are debugging:

*Table 6. Comparison of the same command used in different programming languages*

| If the container name is... | If the programming language is PL/I, COBOL, LangX COBOL, assembler or disassembly, type in... | If the programming language is C or C++, type in... |
|---|---|---|
| chname | DESCRIBE CHANNEL 'chname' | DESCRIBE CHANNEL chname |
| conNAME | DESCRIBE CHANNEL 'conNAME' | DESCRIBE CHANNEL conNAME |

**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`.

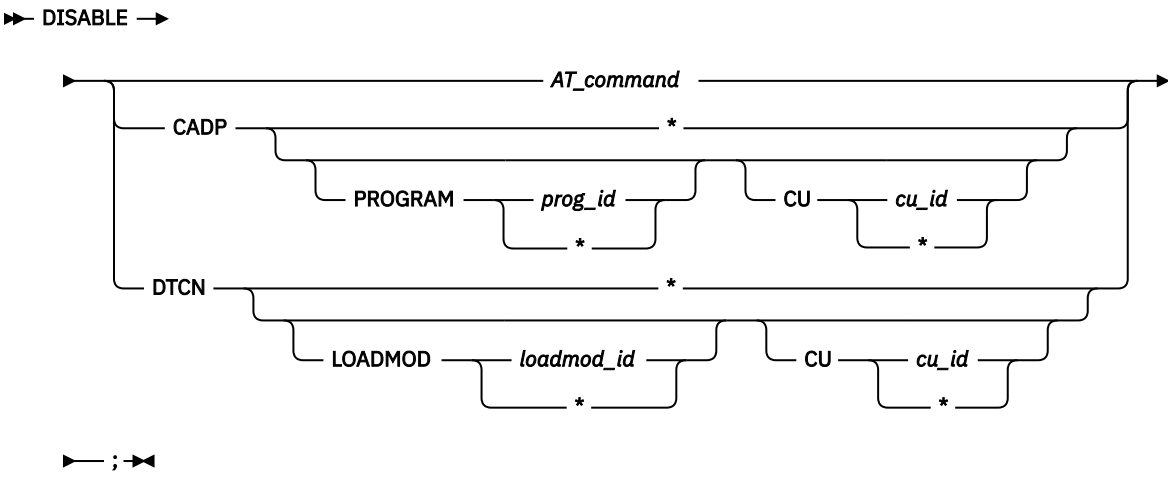
```
DESCRIBE PROGRAMS myprog;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“references” on page 15
“cu_spec” on page 13
“LIST CONTAINER command” on page 147
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

# DISABLE command

The DISABLE command makes an AT or pattern-match breakpoint inoperative. However, the breakpoint is not cleared. Later, you can make the breakpoint operative by using the ENABLE 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`.

**`DTCN LOADMOD`, `DTCN CU`, `CADP PROGRAM`, or `CADP CU`**
Prevents z/OS Debugger from being started by a program, load module, or compile unit specified in *prog_id*, *loadmod_id*, or *cu_id* that matches a program or compile unit specified in a DTCN or CADP profile. The following comparisons are made:

- For DTCN, z/OS Debugger compares *loadmod_id* with the value in the LoadMod field and *cu_id* with the value in the CU field.
- For CADP, *prog_id* is compared to what is specified in the Program field and *cu_id* is compared to what is specified in the Compile Unit field.

You can specify a specific name (for example, PROG1) or a partial name with the wild card character (for example, EMPL*).

**Usage notes**

- You can use the `DISABLE CADP` and `DISABLE DTCN` commands in remote debug mode.
- 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.
- For pseudo-conversational applications running under CICS, the `DISABLE CADP` or `DISABLE DTCN` commands apply only to the current CICS pseudo-conversational task.
- For PL/I, COBOL, LangX COBOL, assembler and disassembly, if the *cu_id* is mixed case or case sensitive, you must enclose the name in quotation marks (") or apostrophes (').
- For C and C++, z/OS Debugger always treats the *cu_id* as case sensitive, even if it is not enclosed in quotation marks (").

**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;
```

- z/OS Debugger starts every time PROGA runs because you have a DTCN profile that specifies an asterisk (*) in the LoadMod field and PROGA in the CU field. field. If you do not want z/OS Debugger to start every time PROGA runs, enter one of the following commands:
  - `DISABLE DTCN LOADMOD * CU PROGA;`
  - `DISABLE DTCN CU PROGA;`
- You have a CADP profile that specifies PROG1 in the Program field and CU1 in the Compile Unit field. If you do not want z/OS Debugger to start every time this program and compile unit are run, enter the following command:

```
DISABLE CADP PROGRAM PROG1 CU CU1;
```

- You have a CADP profile that specifies CU1 in the Compile Unit field. If you do not want z/OS Debugger to start every time the compile unit is run, enter one of the following commands:

```
DISABLE CADP PROGRAM * CU CU1;
DISABLE CADP CU CU1;
```

- You have several CADP profiles and z/OS Debugger is started every time a program matches one of these profiles. If you do not want z/OS Debugger to be started every time a program matches any of these profiles, enter the following command:

```
DISABLE CADP *;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**

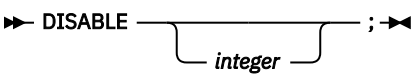
"Controlling pattern-match breakpoints with the ENABLE and DISABLE commands" in the IBM z/OS Debugger User's Guide

**Related references**

“ENABLE command” on page 113
“DISABLE prefix (full-screen mode)” on page 109
“LIST DTCN or CADP command” on page 149
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

## DISABLE prefix (full-screen mode)

Disables a statement breakpoint or offset breakpoint when you issue this command through the Source window prefix area.



***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
```

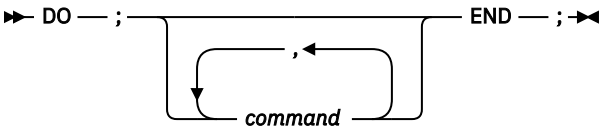
You do not need to enter a space between the keyword and the integer: `DIS 3` is equivalent to `DIS3`.

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
*IBM z/OS Debugger User's Guide*

# DO command (assembler, disassembly, LangX COBOL, and COBOL)

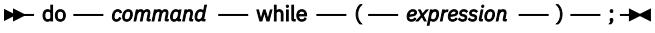
The DO command performs one or more commands that are collected into a group. The DO and END keywords delimit a group of commands called a DO group. The keywords cannot be abbreviated.




***command***
A valid z/OS Debugger 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.




***command***
A valid z/OS Debugger command.

***expression***
A valid z/OS Debugger 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.

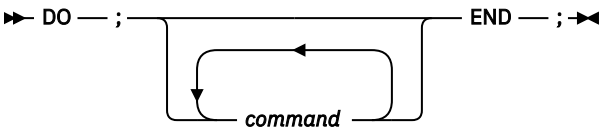
```
int reply1;
```

```
do {
  printf("Enter a 1.\n");
  scanf("%d", &reply1);
} while (reply1 != 1);
```

# DO command (PL/I)

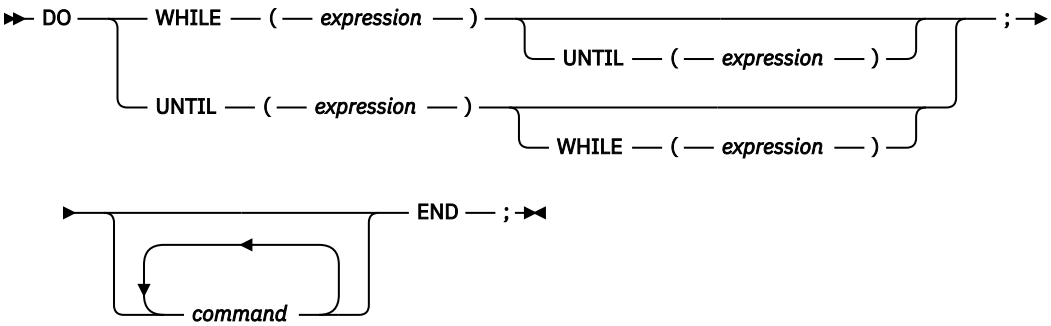
The DO command allows one or more commands to be collected into a group that can (optionally) be repeatedly executed. The DO and END keywords delimit a group of commands collectively called a DO group. The keywords cannot be abbreviated.

**Simple**




***command***
A valid z/OS Debugger command.

**Repeating**



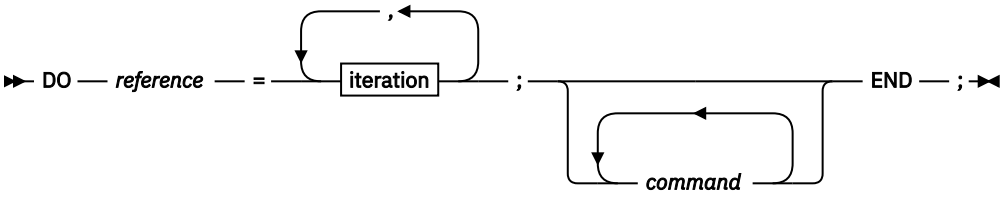

**`WHILE`**
Specifies that *expression* is evaluated before each execution of the command list. If the expression evaluates to true, the commands are executed and the DO group begins another cycle; if it evaluates to false, execution of the DO group ends.

***expression***
A valid z/OS Debugger PL/I Boolean expression.

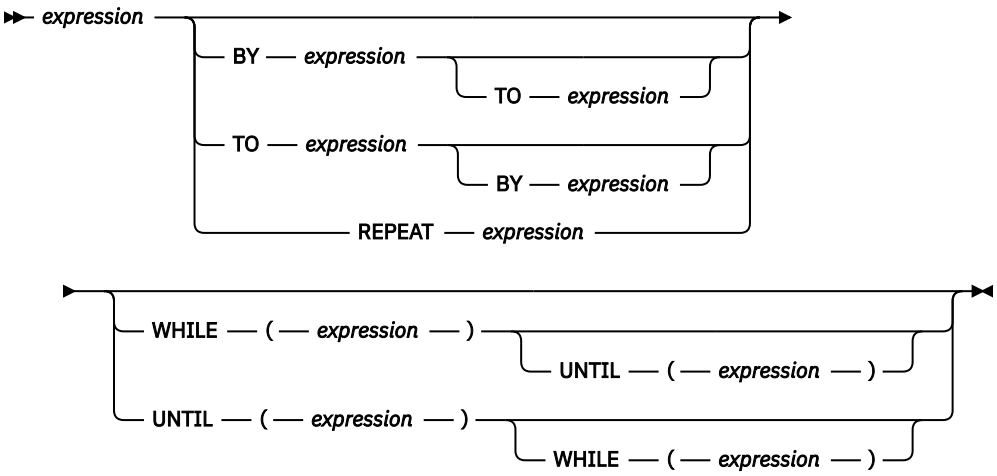
**`UNTIL`**
Specifies that *expression* is evaluated after each execution of the command list. If the expression evaluates to false, the commands are executed and the DO group begins another cycle; if it evaluates to true, execution of the DO group ends.

***command***
A valid z/OS Debugger command.

**Iterative**




**iteration**




***reference***
A valid z/OS Debugger PL/I reference.

***expression***
A valid z/OS Debugger 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 z/OS Debugger 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.

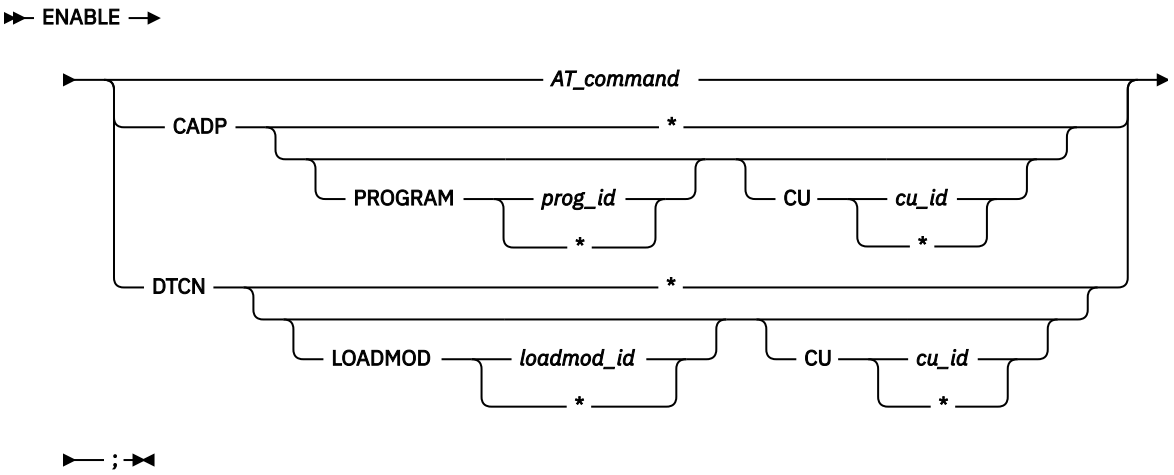
```
DO i = 1 REPEAT 2*i UNTIL (i = 256); END;
```

- 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 activates an AT or pattern-match breakpoint after it was disabled with the DISABLE 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.

**DTCN LOADMOD, DTCN CU, CADP PROGRAM, or CADP CU**
Re-enable a CADP or DTCN profile that was previously disabled by the DISABLE command. The names you specify for *loadmod_id*, *prog_id*, or *cu_id* must match the *loadmod_id*, *prog_id*, or *cu_id* you specified in the DISABLE command.

If you do not specify a *loadmod_id*, *prog_id*, or *cu_id*, z/OS Debugger enables all previously disabled DTCN or CADP profiles. If you try to specify a *loadmod_id*, *prog_id*, or *cu_id* for a profile that was not disabled, z/OS Debugger displays an error message.

### Usage notes

- You can use the ENABLE CADP and ENABLE DTCN commands in remote debug mode.
- 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.
- For pseudo-conversational applications running under CICS, the `ENABLE CADP` or `ENABLE DTCN` commands apply only to the current CICS pseudo-conversational task.
- For PL/I, COBOL, LangX COBOL, assembler and disassembly, if the *cu_id* is mixed case or case sensitive, you must enclose the name in quotation marks (") or apostrophes (').
- For C and C++, z/OS Debugger always treats the *cu_id* as case sensitive, even if it is not enclosed in quotation marks (").

**Examples**

- Reenable the previously disabled command `AT ENTRY mysub CALL proc1;`.

```
ENABLE AT ENTRY mysub;
```

- Allow DTCN to start z/OS Debugger every time PROGA runs, which was previously prevented with the command `DISABLE DTCN CU PROGA;`, by entering the following command:

```
ENABLE DTCN CU PROGA;
```

- Allow CADP to start z/OS Debugger every time a program that matches any of the CADP profiles is run. This was previously prevented with the command `DISABLE CADP *;`.

```
ENABLE CADP *;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**

"Controlling pattern-match breakpoints with the ENABLE and DISABLE commands" in the IBM z/OS Debugger User's Guide

**Related references**

“DISABLE prefix (full-screen mode)” on page 109
“ENABLE prefix (full-screen mode)” on page 114
“LIST DTCN or CADP command” on page 149
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

## 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.


***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**

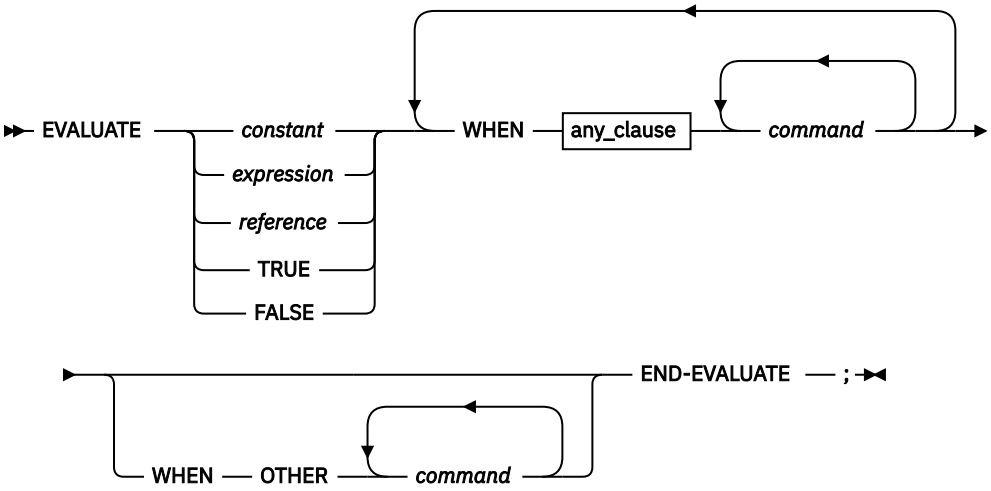
Enable the breakpoint at the third statement or verb in the line (typed in the prefix area of the line where the statement is found).

```
ENABLE 3
```

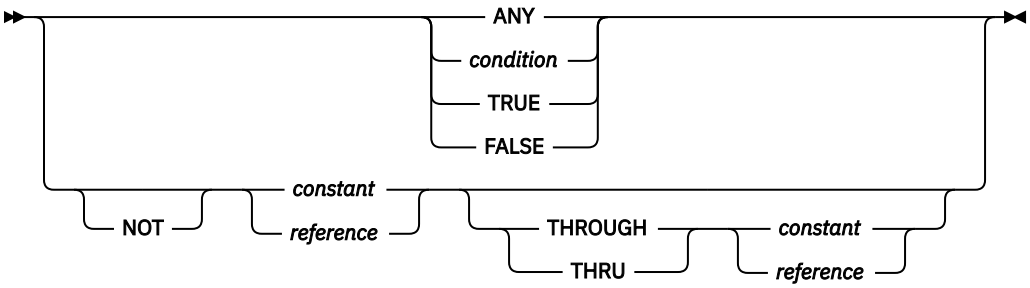
No space is needed as a delimiter between the keyword and the integer; hence, ENABLE  3 is equivalent to ENABLE3.

# EVALUATE command (COBOL)

The EVALUATE command provides a shorthand notation for a series of nested IF statements. The keywords cannot be abbreviated.




**any_clause**




***constant***
A valid z/OS Debugger COBOL constant.

***expression***
A valid z/OS Debugger COBOL arithmetic expression.

***reference***
A valid z/OS Debugger COBOL reference.

***condition***
A simple relation condition.

***command***
A valid z/OS Debugger 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

- z/OS Debugger 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;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

“Allowable comparisons for the IF command (COBOL)” on page 132

*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***
A valid z/OS Debugger 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.

```
x = 3 + 4‘5;
y = 7;
z = 8 * func(x, y);
```

- Increment y and assign the remainder of the integer division of omega by 4 to `alpha`.

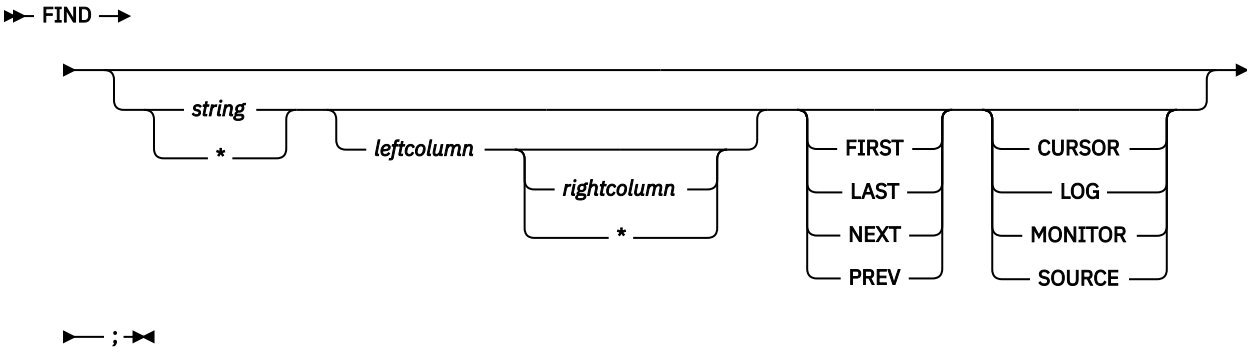
```
alpha = (y++, omega % 4);
```

- To list and assign a new value to R1 in the disassembly view:

```
LIST(R1);
R1 = 0x0001FAF0;
```

# FIND command

The `FIND` command provides full-screen and line mode search capability in the source object, and full-screen searching of the log and monitor objects.




***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:

- The length of the string cannot exceed 128 bytes.
- If the string contains spaces, or is an asterisk (*), a question mark (?) or a semicolon (;) it must be enclosed in quotation marks (") or apostrophes (') as described in the following rules:
  - For C and C++, use quotation marks (").
  - For COBOL, LangX COBOL, assembler, disassembly, or PL/I, use quotation marks (") or apostrophes (').

*Table 7. Examples of how to specify quotation marks (") and apostrophes (') for strings in a FIND command.*

| C | C++ | COBOL or LangX COBOL | Assembler or disassembly | PL/I |
|---|---|---|---|---|
| "ABC" | "IntLink::*" | "A5" or 'A5' | 'ABC' or "ABC" or C'ABC' | 'ABC' or "ABC" |

- If the string contains a quotation mark (") or apostrophe ('), you might have to specify the string with an even number of quotation marks or apostrophes (also known as *balance*). Use the following rules to determine how to balance the string:
  - For PL/I, if the string has an apostrophe, you must add an apostrophe immediately following that apostrophe. If the string contains a space, surround the entire string with apostrophes.
  - For C and C++, if the string has a quotation mark, you must add a quotation mark immediately following that quotation mark. If the string contains a space, surround the entire string with quotation marks.
  - For assembler, COBOL, LangX COBOL, or disassembly, if the string contains an apostrophe and it is delimited by apostrophes, you must add an apostrophe immediately after the apostrophe that is in the string. If the string contains a quotation mark and it is delimited by quotation marks, you must add a quotation mark immediately after the quotation mark that is in the string. If the string contains a space, you do not have to balance the quotation marks; however you must surround the entire string with a quotation marks or apostrophes.

If no operands are specified, a repeat FIND is performed. The usage notes and *IBM z/OS Debugger User's Guide* describes repeat FIND.

*****
Use the string from the previous FIND command.

***leftcolumn***
A positive integer that specifies the leftmost column for the search. This is supported only in the Source window and in line mode. It is ignored in the Log and Monitor windows. If *rightcolumn* and * are omitted, then the string must start in *leftcolumn*.

***rightcolumn***
A positive integer that specifies the rightmost column for the search. This is supported only in the Source window and in line mode. It is ignored in the Log and Monitor windows.

*****
Specifies that the length of each source record is used as the right column for the search. This is supported only in the Source window and in line mode. It is ignored in the Log and Monitor windows.

**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, z/OS Debugger displays an error message.
  - If the previous FIND command that you entered specified or implied the FIRST or NEXT parameter, z/OS Debugger uses the NEXT parameter.
  - If the previous FIND command that you entered specified the LAST or PREV parameter, z/OS Debugger uses the PREV parameter.
  - If the previous FIND command that you entered specified a *leftcolumn* parameter, z/OS Debugger uses that *leftcolumn* parameter.
  - If the previous FIND command that you entered specified a *rightcolumn* parameter, z/OS Debugger uses that *rightcolumn* parameter.
  - If a repeat FIND immediately follows an unsuccessful FIND or repeat FIND, z/OS Debugger 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, z/OS Debugger wraps from the first line to the last line.
  - If the cursor is not in a window, z/OS Debugger uses the same window that was used for the previous FIND command.
- In full-screen mode, z/OS Debugger 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, z/OS Debugger 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, z/OS Debugger 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 show 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, z/OS Debugger wraps to the end of the object and continues from the end of the last line in the object.
  - If z/OS Debugger 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. The highlighted string is protected from overtyping. If you need to overtype the string, press enter and place the cursor where you want to type and proceed with the overtype.

  - If z/OS Debugger 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 z/OS Debugger searched only part of the object, then z/OS Debugger displays the message 'Bottom of data reached' or 'Top of data reached', as appropriate. If z/OS Debugger searched through the entire object, then it displays the message 'Search target not found'.
- In line 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, z/OS Debugger 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, z/OS Debugger 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 z/OS Debugger searched only part of the source and did not find the string, then z/OS Debugger displays the message 'Bottom of data is reached' or 'Top of data is reached', as appropriate. If z/OS Debugger searched through the entire source without finding the string, then it displays the message 'Search target not found'.
  - If z/OS Debugger finds the string, the line that contains the string is displayed and marked with a vertical bar character (|) beneath the string.
- The search in the Source window and in line mode can be limited to certain columns by choosing one of the following methods:
  - If you enter a pair of column numbers indicating the first and last columns to be searched, the string is found if it is completely contained within the specified columns.
  - If a single column is specified, the string must start in the specified column.
  - If the second column specified is larger than the record size, the record size is used.
  - If the columns are not specified, the columns to be searched default to the columns defined by the SET FIND BOUNDS command. If you have not entered the SET FIND BOUNDS command, the columns default to 1 *.

  The column alignment of the source might not match the original source code. The *leftcolumn* and *rightcolumn* specifications are related to the scale shown in the Source window, not the original source.
- The full-screen FIND command is not logged; however, the FIND command is logged in line mode.
- If you are searching for strings with trigraphs in them when debugging C or C++ code, the trigraphs or their equivalents can be used as input, and z/OS Debugger matches them to trigraphs or their equivalents. An exception is that column specifications other than 1 * are not allowed in FIND or SET FIND BOUNDS if you search source code and trigraphs are found.
- If you are searching in the monitor window and SET MONITOR WRAP OFF is in effect, z/OS Debugger will search all of the scrolled data.
- You cannot use the FIND command in the Memory window.

**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 z/OS Debugger 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" ;
```

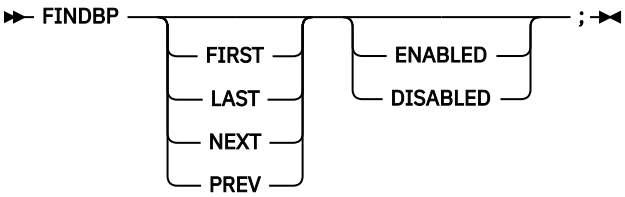
- If you entered the command `FIND xyz LAST;` or `FIND xyz PREV;` and the cursor is on the found string ("xyz"), then press the PF key assigned to the `FIND` command to repeat the search. z/OS Debugger runs the command `FIND xyz PREV;`.
- If you entered the command `FIND xyz;`, z/OS Debugger searches in the forward direction. To find the string "xyz" in the backward direction, enter the command `FIND * PREV;`.
- If you want to find a COBOL paragraph definition named `paraa` that starts in column 8 in COBOL's Area A, enter the following command:

```
FIND paraa 8 ;
```

- If you want to find a reference to a COBOL paragraph named `paraa` in COBOL's Area B, then enter one of the following commands:
  - `FIND paraa 12 72;`
  - ```
    SET FIND BOUNDS 12 72;
    FIND paraa;
    ```

## FINDBP command

The `FINDBP` command provides full-screen search capability for line, statement and offset breakpoints in the source object. The `FINDBP` keyword cannot be abbreviated.




**FIRST**
Starts at the beginning of the source object and searches forward to find the first line, statement, or offset breakpoint.
    ```

**LAST**
Starts at the end of the source object and searches backward to find the last line, statement, or offset breakpoint.

**NEXT**
Starts at the next line after the current cursor location in the Source window and searches forward to find the next line, statement, or offset breakpoint

**PREV**
Starts at the previous line before the current cursor location in the Source window and searches backward to find the previous line, statement, or offset breakpoint

**ENABLED**
Restricts the searching to enabled breakpoints. The default is to list both enabled and disabled breakpoints.

**DISABLED**
Restricts the searching to disabled breakpoints. The default is to list both enabled and disabled breakpoints.

**Usage notes**

- If no operands are specified, a repeat FINDBP is performed. A repeat FINDBP behaves in the following ways:
  - If the previous FINDBP command that you entered specified or implied the FIRST or NEXT parameter, z/OS Debugger uses the NEXT parameter.
  - If the previous FINDBP command that you entered specified or implied the LAST or PREV parameter, z/OS Debugger uses the PREV parameter.
  - If a repeat FINDBP immediately follows an unsuccessful FINDBP or repeat FINDBP, z/OS Debugger continues searching, wrapping from the last line to the first line. If the original direction of the FINDBP was backward to the beginning of the source object, z/OS Debugger wraps from the first line to the last line.
  - If the previous FINDBP command that you entered specified or implied the ENABLED or DISABLED parameter, z/OS Debugger uses the ENABLED or DISABLED parameter, respectively.
  - If you want to frequently use a repeat FINDBP, set a PF key (for example, PF17 or shift PF5) to FINDBP. For instructions on assigning a command to a PF key, see "SET PFKEY command" on page 247.
- Searches show the following behavior:
  - If you specify FIRST, the search begins at the first line of the source object.
  - If you specify LAST, the search begins at the last line of the source object.
  - If you specify NEXT or the command defaults to NEXT and the cursor is on a source line or in its prefix or suffix area, the search begins at the line after the line the cursor is on.
  - If you specify NEXT or the command defaults to NEXT and the cursor is not on a source line or in its prefix or suffix area, the search begins at the first line in the Source window.
  - If you specify PREV or the command defaults to PREV and the cursor is on a source line or in its prefix or suffix area, the search begins at the line before the line the cursor is on.
  - If you specify PREV or the command defaults to PREV and the cursor is not on a source line or in its prefix or suffix area, the search begins at the line before the first line in the Source window. If the first line of the source object is displayed, z/OS Debugger wraps to the end of the source object and continues with the last source line.
  - If z/OS Debugger finds the breakpoint, z/OS Debugger scrolls the Source window so that you can see the breakpoint. z/OS Debugger places the cursor at the beginning of the prefix area for the source line that contains the breakpoint.
  - If z/OS Debugger does not find the breakpoint, 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 z/OS Debugger searched only part of the source object, then z/OS Debugger displays the message "Bottom of data

reached" or "Top of data reached", as appropriate. If z/OS Debugger searched through the entire source object, then it displays the message "No line, statement or offset breakpoints were found".

- If multiple line or statement breakpoints exist on the same source line, the FINDBP command finds only one of them.
- The FINDBP command does not find AT STATEMENT * breakpoints.
- The FINDBP command searches only through the currently qualified compile unit, which is the compile unit visible in the Source window.
- z/OS Debugger does not log the FINDBP command.
- If you know the line number or statement number of the breakpoint you are looking for, the quickest way to find it is to use the SCROLL TO *nnnnn* or POSITION *nnnnn* command, which scrolls the Source window so that the line containing *nnnnn* in the prefix area is the first line in the Source window.

**Examples**

- Search for the next line in the Source window that contains a line, statement, or offset breakpoint.

```
FINDBP
```

- Search for the first line in the source object that contains a line, statement, or offset breakpoint. Then search for the next two breakpoints.

```
FINDBP FIRST
FINDBP
FINDBP
```

**Related references**

**Related references**

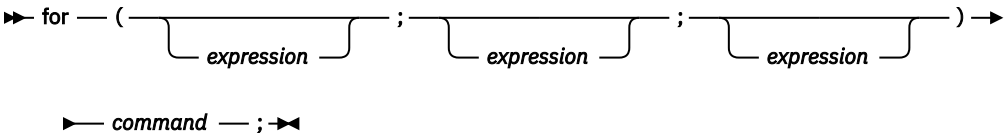
"AT LINE command" on page 63
"AT OFFSET command (disassembly)" on page 68
"AT STATEMENT command" on page 70
"LIST AT command" on page 141, with the LINE, OFFSET, or STATEMENT options
"POSITION command" on page 191
"SCROLL command (full-screen mode)" on page 204, with the TO option
"SET PFKEY command" on page 247

# 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.




***expression***
A valid z/OS Debugger C and C++ expression.

***command***
A valid z/OS Debugger command.

z/OS Debugger 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.

z/OS Debugger 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.

z/OS Debugger 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.

```
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.

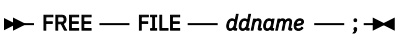
```
count = 1;
while (count <= 20) {
  printf("count = %d\n", count);
  count++;
}
```

- The following `for` command does not contain an initialization expression.

```
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.




***ddname***
Name of the file to free.

# GO command

The G0 command causes z/OS Debugger to start or resume running your program.


**BYPASS**

Bypasses the user or system action for the condition that caused the breakpoint. It is valid only when z/OS Debugger is entered for an:

AT CALL Breakpoint
HLL or Language Environment condition
Condition that is raised by an MVS or CICS ABEND when running without the Language Environment run time

**Usage notes**

- For CICS only: The ABEND is reported whether BYPASS is or is not specified. When there is a HANDLE ABEND, control is passed to the abend handler, and the GO BYPASS command is ignored.
- 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.
- You can use the GO command in remote debug mode by entering it in the Debug Console or the **Action** field, which is in the **Optional Parameters** section of the **Add a Breakpoint** task.
- When a COBOL IGZ condition of severity 2 or higher occurs, GO BYPASS will bypass the condition. When the IGZ condition is raised by a COBOL program (for example the subscript out of range message IGZ0006S), GO BYPASS will bypass the condition and resume control back into the COBOL program. However, be aware that control might not return to the next statement of the program that raised the condition, since the compiler might have rearranged the statements.

**Examples**

- Resume execution.

```
GO;
```

- Resume execution and bypass user and system actions for the condition that caused the breakpoint.

```
GO BYPASS;
```

- Your application has abended 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 topics for more information related to the material discussed in this topic.

**Related references**

"AT command" on page 37

## GOTO command

The GOTO command causes z/OS Debugger to resume program execution at the specified statement id. The GOTO keyword cannot be abbreviated. If you want z/OS Debugger to return control to you at a target location, make sure there is a breakpoint at that location.


**Usage notes**

- You can use the GOTO command if the SET WARNING is set to OFF and the runtime level allows GOTO without compiler enablement for the following programs:
  - A COBOL program compiled without hooks being inserted by the compiler and with optimization, if you compiled with the NOEJPD suboptions of the TEST compiler option
  - A program compiled with Enterprise COBOL for z/OS Version 5 or later and optimized by Automatic Binary Optimizer for z/OS

  The use of GOTO in this case might cause unpredictable behaviors, including abends, when the GOTO command is executed or followed. You can get the best behavior of GOTO command in programs that are compiled with OPT and TEST(NOEJPD) options in either of the following situations:
  - When the target of the GOTO or JUMPTO command is a paragraph name or a section name (label).
  - When the target of the GOTO or JUMPTO command is the first statement in the paragraph or section.

  You can get the best behavior especially if the statements are targets of COBOL statements PERFORM or GOTO in the COBOL program. See "SET WARNING command (C, C++, COBOL, and PL/I)" on page 263.
- 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 PLITEST 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 in the following situations:
  - A COBOL program compiled with hooks inserted by the compiler. If you are using Enterprise COBOL for z/OS, Version 4, compile your program with the HOOK suboption of the TEST compiler option. If you are using any of the following compilers, compile your program with either PATH or ALL suboption and the SYM suboption of the TEST compiler option:
    - Enterprise COBOL for z/OS and OS/390, Version 3
    - COBOL for OS/390 & VM, Version 2
  - A COBOL program compiled without hooks inserted by the compiler and without optimization. If you are using Enterprise COBOL for z/OS, Version 4, compile your program with the NOHOOK suboption of the TEST compiler option. If you are using any of the following compilers, compile your program with the NONE suboption of the TEST compiler option:
    - Enterprise COBOL for z/OS and OS/390, Version 3 Release 2 or later
    - Enterprise COBOL for z/OS and OS/390, Version 3 Release 1, with APAR PQ63235 installed
    - COBOL for OS/390 & VM, Version 2 Release 2
    - COBOL for OS/390 & VM, Version 2 Release 1, with APAR PQ63234 installed
  - A COBOL program compiled without hooks inserted by the compiler and with optimization. You must compile your program with Enterprise COBOL for z/OS, Version 4, and specify the EJPD and NOHOOK

suboption of the TEST compiler option. Specifying the EJPD suboption might cause some loss of optimization.

  - For Enterprise COBOL for z/OS Version 5, programs are always compiled without hooks inserted by the compiler. If you are using the TEST compiler option in combination with any level of the OPT compiler option, it is recommended to use the EJPD suboption of the TEST compiler option.

- This command cannot be used if you are stopped at an AT APPEARANCE breakpoint, an AT LOAD breakpoint, or an AT DELETE breakpoint.

**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, z/OS Debugger 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
```

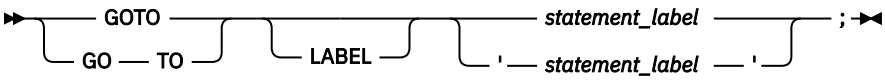
Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
*IBM z/OS Debugger User's Guide*

**Related references**
“statement_id” on page 16

## GOTO LABEL command

The GOTO LABEL command causes z/OS Debugger to resume program execution at the specified statement label. The specified label must be in the same block. If you want z/OS Debugger to return control to you at the target location, make sure there is a breakpoint at that location.




***statement_label***
A valid statement label within the currently executing program or, in PL/I, a label variable.

**Usage notes**

- For COBOL, if a GOTO LABEL command is issued and the specified label contains an EXIT statement, the results might be unpredictable such as an ABEND because the EXIT statement might not be specified with a return location.
- You can use the GOTO command if the SET WARNING is set to OFF and the runtime level allows GOTO without compiler enablement for the following programs:
  - A COBOL program compiled without hooks being inserted by the compiler and with optimization, if you compiled with the NOEJPD suboptions of the TEST compiler option
  - A program compiled with Enterprise COBOL for z/OS Version 5 or later and optimized by Automatic Binary Optimizer for z/OS

  The use of GOTO in this case might cause unpredictable behaviors, including abends, when the GOTO command is executed or followed. You can get the best behavior of GOTO command in programs that are compiled with OPT and TEST(NOEJPD) options in either of the following situations:
  - When the target of the GOTO or JUMPTO command is a paragraph name or a section name (label).
  - When the target of the GOTO or JUMPTO command is the first statement in the paragraph or section.

You can get the best behavior especially if the statements are targets of COBOL statements PERFORM or GOTO in the COBOL program. See "SET WARNING command (C, C++, COBOL, and PL/I)" on page 263.

- Use the syntax of *statement_label* enclosed in apostrophes (') only for LangX 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 z/OS Debugger 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.
  - 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 programs, you can use GOTO LABEL command if you compile your program with the following suboptions and compilers:
  - The HOOK suboption of the TEST compiler option with Enterprise COBOL for z/OS, Version 4
  - The PATH or ALL suboption and the SYM suboption of the TEST compiler option with the following compilers:
    - Enterprise COBOL for z/OS and OS/390, Version 3
    - COBOL for OS/390 & VM, Version 2
  - For Enterprise COBOL for z/OS Version 5, programs are always compiled without hooks inserted by the compiler. If you are using the TEST compiler option in combination with any level of the OPT compiler option, it is recommended to use the EJPD suboption of the TEST compiler option.

  The label can take one of the following 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 z/OS Debugger always displays OF in the log.
- 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.
- This command cannot be used if you are stopped at an AT APPEARANCE breakpoint, an AT LOAD breakpoint, or an AT DELETE breakpoint.

**Examples**

- Go to the label constant laba in block suba in program prog1.

```
GOTO prog1:>suba:>laba;
```

- Go to the label constant para OF sect1. The current programming language setting is COBOL.

```
GOTO LABEL para OF sect1;
```

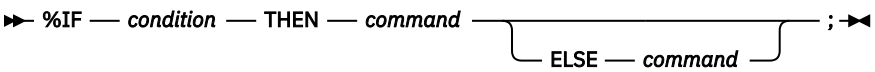
Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
*IBM z/OS Debugger User's Guide*

**Related references**
"statement_label" on page 17

# %IF command (programming language neutral)

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 keywords cannot be abbreviated.




***condition***
A simple relation condition valid for all supported programming languages.

***command***
A valid z/OS Debugger command or a BEGIN-END group containing one or more valid z/OS Debugger commands. The z/OS Debugger commands must be valid for all supported programming languages.

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**

- The IF commands that are specific to a programming language might contain restrictions or usage notes. Those restrictions and usage notes also apply to the %IF command.
- The variable names used in *condition* must be syntactically valid for all supported programming languages.
- If you want to nest %IF commands, you cannot mix them with programming language-specific IF commands.

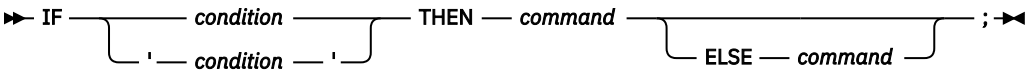
Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"BEGIN command" on page 74
"IF command (assembler, disassembly, and LangX COBOL)" on page 130
"if command (C and C++)" on page 130
"IF command (COBOL)" on page 131

"IF command (PL/I)" on page 134

# IF command (assembler, disassembly, and LangX 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.




***condition***
An assembler conditional expression.

**'*condition*'**
A LangX COBOL conditional expression enclosed in apostrophes (').

***command***
A valid z/OS Debugger command or a D0 group containing one or more valid z/OS Debugger Commands.

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 note**

You cannot use the IF command while you replay recorded statements by using the PLAYBACK command.

**Examples**

- If the value of register 1 is 0, then assign 0 to variable XYZ by using the following command:

```
IF %R1 = 0 THEN STORAGE(XYZ)=0;
```

- If the value of variable XYZ is equal to 22, set a breakpoint at statement 52 by using the following command:

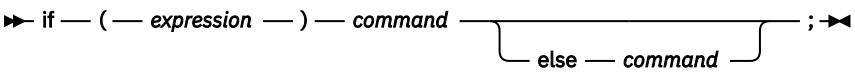
```
IF XYZ=22 THEN  AT 52;
```

- If the value of the LangX COBOL variable XYZ is 2, assign 0 to variable XYZ by using the following command:

```
IF 'XYZ = 2' THEN 'XYZ' = '0';
```

# 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.




**`expression`**
A valid z/OS Debugger C and C++ expression.

**`command`**
A valid z/OS Debugger 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 z/OS Debugger to get more input (for example, an `if` containing USE or other commands that cause z/OS Debugger 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.

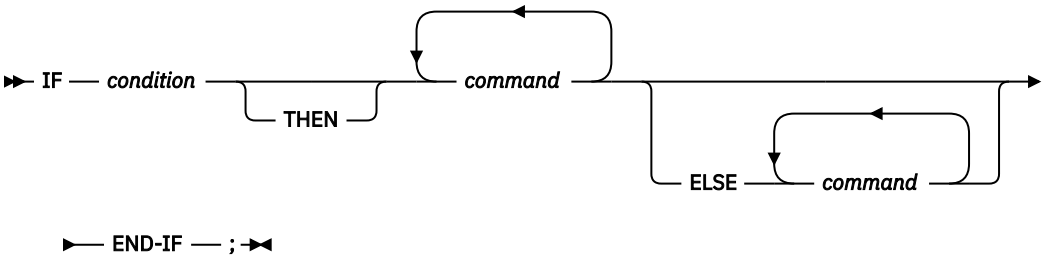
```
if (score >= 90)
   grade = "A";
```

- 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 with the following form: *Item-1* `operator` *Item-2*. *Item-1* and *Item-2* can be a data-item or a literal. The operator can be one of the following operations:

- `>`
- `<`
- `=`
- `NOT =`
- `>=`
- `<=`
- `NOT <`
- `NOT >`

**command**
A valid z/OS Debugger 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, z/OS Debugger 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 z/OS Debugger to get more input (for example, an IF containing USE or other commands that cause z/OS Debugger 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 topics for more information related to the material discussed in this topic.

**Related references**

"Allowable comparisons for the IF command (COBOL)" on page 132

## Allowable comparisons for the IF command (COBOL)

The following table shows the allowable comparisons for the z/OS Debugger IF command. A description of the codes follows the table.

For Enterprise COBOL for z/OS Version 5, z/OS Debugger supports all the same comparisons that are supported in the COBOL language, so the following table does not apply. See the *Enterprise COBOL for z/OS Language Reference* for more information.

| OPERAND | GR | AL | AN | ED | BI | NE | ANE | NDI | NN DI | ID | IN | IDI | PTR | @ | IF | EF | D1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Group (GR) | NN | NN | NN | NN | NN | NN | NN | NN[10] | | NN | | NN | | | NN | NN | |
| Alphabetic (AL) | NN | NN | | | | | | NN | | | | | | | | | |
| Alpha numeric (AN)[8] | NN | | NN | | | | | NN | | | | | | | | | |
| External Decimal (ED)[8] | NN | | | NU | | | | | | | | | | | | | |
| Binary | NN | | | | NU | | | | | | NU[4] | | | | | | |
| Numeric Edited (NE) | NN | | | | | NN | | | | | | | | | | | |
| Alphanumeric Edited (ANE) | NN | | | | | | NN | NN | | | | | | | | | |
| FIGCON ZERO[7] | NN | | | NU | NU | | | NN | | NU | | | | | NU | NU | |
| FIGCON[1,7] | NN | NN | NN | | | | NN | NN[9] | NU | | | | | | | | |
| National Data Item (NDI) | NN[10] | NN | NN | | | | NN | NN | | | | | | | | | |
| National Numeric Data Item (NNDI) | | | | | | | | | NN | | | | | | | | |
| Numeric Literal[7] | NN | | | NU | NU | | | | NN | NU | NU[4] | | | | NU | NU | |

| OPERAND | GR | AL | AN | ED | BI | NE | ANE | NDI | NN DI | ID | IN | IDI | PTR | @ | IF | EF | D1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Alphanumeric Literal[2,7] | NN | NN[3] | NN | | | NN | NN | NN | | | | | | | | | |
| Alphanumeric hex literal[11] | NN | NN | NN | | | NN | NN | | | | | | | | | | |
| Internal Decimal (ID)[8] | NN | | | | | | | | | NU | | | | | | | |
| Index Name (IN) | NN | | | | NU[4] | | | | | | IO[4] | NU | | | | | |
| Index Data Item (IDI) | NN | | | | | | | | | | NU | IV | | | | | |
| Pointer Data Item (PTR) | | | | | | | | | | | | | NU[5] | NU[5] | | | |
| Address of (@) | | | | | | | | | | | | | NU[5] | NU[5] | | | |
| Floating Point Literal[7] | X | | | | | | | | | | | | | | NU | NU | |
| Internal Floating Point (IF) | NN | | | | | | | | | | | | | | NU | NU | |
| External Floating Point (EF) | NN | | | | | | | | | | | | | | NU | NU | |
| DBCS data item (D1) | | | | | | | | | | | | | | | | | NN |
| DBCS Literal[7] | | | | | | | | | | | | | | | | | NN |
| Address hex Literal[6] | | | | | | | | | | | | | NU[5] | NU[5] | | | |
| National Literal | NN[10] | | | | | | | NN | | | | | | | | | |
| National Hex Literal[12] | NN[10] | | | | | | | NN | | | | | | | | | |

**Notes:**

1. FIGCON includes all figurative constants except ZERO and ALL.
2. A alphanumeric literal must be enclosed in quotation marks (") or apostrophes ('). A quotation mark or apostrophe embedded in the string must be followed by another quotation mark or apostrophe when it is used as the opening delimiter.
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 quotation marks (") or apostrophes (') 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 quotation marks (") or apostrophes (') and preceded by X.
12. Must be hexadecimal characters only, delimited by either quotation marks (") or apostrophes (') 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

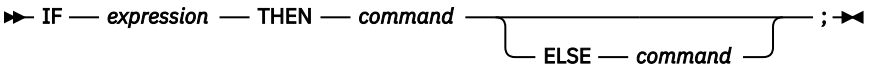
**X**
High potential for user error

Refer to the following topics for more information related to the material discussed in this topic.

**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.




***expression***
A valid z/OS Debugger PL/I expression.

If necessary, the expression is converted to a BIT string.

***command***
A valid z/OS Debugger 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 z/OS Debugger to get more input (for example, an IF containing USE or other commands that cause z/OS Debugger 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.

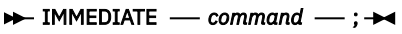
```
AT 23 IF j = 10 THEN LIST TITLED (rmdr, totodd, terms(j)); ELSE GO;
```

# 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.

Prefix the PF key definitions for the FIND, FINDBP, RETRIEVE, SCROLL, and WINDOW commands with the IMMEDIATE command so that these commands work when you enter a group of commands.




***command***
One of the following z/OS Debugger commands:

- FIND
- FINDBP
- 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.

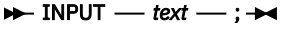
```
IMMEDIATE WINDOW OPEN LOG;
```

- 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.




***text***
Specifies text input to a pending read.

**Usage notes**

- The *text* consists of everything between the INPUT keyword and the semicolon (or end-of-line). Any leading or trailing blanks are removed by z/OS Debugger.
- If a semicolon (;) is included as part of the *text*, the *text* must be surrounded in quotation marks (") or apostrophes (') and conform to the syntax rules for a character string constant enclosed in quotation marks or apostrophes for the current programming language.
- If the *text* contains a quotation mark (") or apostrophe ('), the quotation mark or apostrophe must be followed by a matching quotation mark or apostrophe.
- 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 z/OS Debugger 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, you must enter your input as a valid character string for your programming language. If you want to include a quotation mark (") or apostrophe (') in your input, you must follow each quotation mark or apostrophe with a matching quotation mark or apostrophe and enter the 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;
```

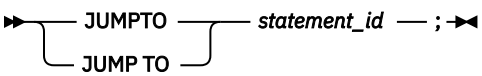
Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

“SET INTERCEPT command (C and C++)” on page 232
“SET INTERCEPT command (COBOL, full-screen mode, line mode, batch mode)” on page 233

# 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.




**Usage notes**

- You can use the JUMPTO command if the SET WARNING is set to OFF and the runtime level allows JUMPTO without compiler enablement for the following programs:
  - A COBOL program compiled without hooks being inserted by the compiler and with optimization, if you compiled with the NOEJPD suboptions of the TEST compiler option
  - A program compiled with Enterprise COBOL for z/OS Version 5 or later and optimized by Automatic Binary Optimizer for z/OS

The use of JUMPTO in this case might cause unpredictable behaviors, including abends, when the JUMPTO command is executed or followed. You can get the best behavior of JUMPTO in programs that are compiled with OPT and TEST(NOEJPD) options in either of the following situations:

- When the target of the GOTO or JUMPTO command is a paragraph name or a section name (label).
- When the target of the GOTO or JUMPTO command is the first statement in the paragraph or section.

You can get the best behavior especially if these statements are targets of COBOL statements PERFORM or GOTO in the COBOL program. See "SET WARNING command (C, C++, COBOL, and PL/I)" on page 263.

- 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 JUMPTO command follows the COBOL language rules that apply to the GOTO statement. You can use the JUMPTO command in the following situations:
  - A COBOL program compiled with hooks inserted by the compiler. If you are using Enterprise COBOL for z/OS, Version 4, compile your program with the HOOK suboption of the TEST compiler option. If you are using any of the following compilers, compile your program with either PATH or ALL suboption and the SYM suboption of the TEST compiler option:
    - Enterprise COBOL for z/OS and OS/390, Version 3
    - COBOL for OS/390 & VM, Version 2
  - A COBOL program compiled without hooks inserted by the compiler and without optimization. If you are using Enterprise COBOL for z/OS, Version 4, compile your program with the NOHOOK suboption of the TEST compiler option. If you are using any of the following compilers, compile your program with the NONE suboption of the TEST compiler option:
    - Enterprise COBOL for z/OS and OS/390, Version 3 Release 2 or later
    - Enterprise COBOL for z/OS and OS/390, Version 3 Release 1, with APAR PQ63235 installed
    - COBOL for OS/390 & VM, Version 2 Release 2
    - COBOL for OS/390 & VM, Version 2 Release 1, with APAR PQ63234 installed
  - A COBOL program compiled without hooks inserted by the compiler and with optimization. You must compile your program with Enterprise COBOL for z/OS, Version 4, and specify the EJPD and NOHOOK suboption of the TEST compiler option. Specifying the EJPD suboption might cause some loss of optimization.
  - For Enterprise COBOL for z/OS Version 5, programs are always compiled without hooks inserted by the compiler. If you are using the TEST compiler option in combination with any level of the OPT compile option, it is recommended to use the EJPD suboption of the TEST compile option.
- You can use the JUMPTO command in remote debug mode by entering it in the Debug Console or the **Action** field, which is in the **Optional Parameters** section of the **Add a Breakpoint** task.
- This command cannot be used if you are stopped at an AT APPEARANCE breakpoint, an AT LOAD breakpoint, or an AT DELETE breakpoint.

**Example**

You want to jump to statement 24 and then stop there. Enter the following command:

```
JUMPTO 24;
```

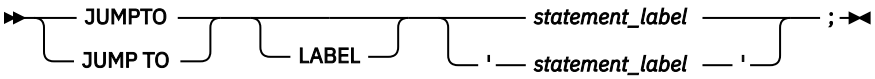
Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
*IBM z/OS Debugger 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.




***statement_label***
A valid statement label within the currently executing program or, in PL/I, a label variable.

**Usage notes**

- For COBOL, if a JUMPTO LABEL command is issued and the specified label contains an EXIT statement, the results might be unpredictable such as an ABEND because the EXIT statement might not be specified with a return location.
- You can use the JUMPTO command if the SET WARNING is set to OFF and the runtime level allows JUMPTO without compiler enablement for the following programs:
  - A COBOL program compiled without hooks being inserted by the compiler and with optimization, if you compiled with the NOEJPD suboptions of the TEST compiler option
  - A program compiled with Enterprise COBOL for z/OS Version 5 or later and optimized by Automatic Binary Optimizer for z/OS

  The use of JUMPTO in this case might cause unpredictable behaviors, including abends, when the JUMPTO command is executed or followed. You can get the best behavior of JUMPTO in programs that are compiled with OPT and TEST(NOEJPD) options in either of the following situations:
  - When the target of the GOTO or JUMPTO command is a paragraph name or a section name (label).
  - When the target of the GOTO or JUMPTO command is the first statement in the paragraph or section.

  You can get the best behavior especially if these statements are targets of COBOL statements PERFORM or GOTO in the COBOL program. See “SET WARNING command (C, C++, COBOL, and PL/I)” on page 263.
- Use the syntax of *statement_label* enclosed in apostrophes (') only for LangX 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 z/OS Debugger 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.
  - 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 programs, you can use `JUMPTO LABEL` command if you compile your program with the following suboptions and compilers:
  - The `HOOK` suboption of the `TEST` compiler option with Enterprise COBOL for z/OS, Version 4
  - The `PATH` or `ALL` suboption and the `SYM` suboption of the `TEST` compiler option with the following compilers:
    - Enterprise COBOL for z/OS and OS/390, Version 3
    - COBOL for OS/390 & VM, Version 2
  - For Enterprise COBOL for z/OS Version 5, programs are always compiled without hooks inserted by the compiler. If you are using the `TEST` compiler option in combination with any level of the `OPT` compiler option, it is recommended to use the `EJPD` suboption of the `TEST` compiler option.

  The label can take one of the following 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 z/OS Debugger always displays `OF` in the log.
- 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:>laba;
```

- Jump to the label constant `para OF sect1`. The current programming language setting is COBOL.

```
JUMPTO LABEL para OF sect1;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
*IBM z/OS Debugger 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.

| Command | Description |
|---|---|
| “LIST (blank) command” on page 141 | Displays Source Identification panel |
| “LIST AT command” on page 141 | Lists the currently defined breakpoints. |
| “LIST CALLS command” on page 144 | Displays the dynamic chain of active blocks. |
| “LIST CC command” on page 145 | Lists code coverage data. |
| “LIST CONTAINER command” on page 147 | Displays the contents of a container. |
| “LIST CURSOR command (full-screen mode)” on page 148 | Displays the variable pointed to by the cursor. |
| “LIST DTCN or CADP command” on page 149 | List the load modules, programs, and compile units that were disabled by the DISABLE command. |
| “LIST expression command” on page 149 | Displays values of expressions. |
| “LIST FREQUENCY command” on page 155 | Lists statement execution counts. |
| “LIST LAST command” on page 155 | Displays a list of recent entries in the history table. |
| “LIST LDD command” on page 156 | Displays a list of LOADDEBUGDATA (LDD) commands known to z/OS Debugger. |
| “LIST LINE NUMBERS command” on page 157 | Lists all line numbers that are valid locations for an AT LINE breakpoint. |
| “LIST LINES command” on page 157 | Lists one or more lines from the current listing or source file displayed in the Source window. |
| “LIST MONITOR command” on page 157 | Lists the current set of MONITOR commands. |
| “LIST NAMES command” on page 158 | Lists the names of variables, programs, or z/OS Debugger procedures. |
| “LIST ON (PL/I) command” on page 160 | Lists the action (if any) currently defined for the specified PL/I conditions. |

| Command | Description |
|---|---|
| “LIST PROCEDURES command” on page 160 | Lists the commands contained in the specified z/OS Debugger procedure. |
| “LIST REGISTERS command” on page 161 | Displays the current register contents. |
| “LIST STATEMENT NUMBERS command” on page 162. | Lists all statement numbers that are valid locations for an AT STATEMENT breakpoint. |
| “LIST STATEMENTS command” on page 162 | Lists one or more statements from the current listing or source file displayed in the Source window. |
| “LIST STORAGE command” on page 163 | Provides a dump-format display of storage. |
| “LIST TRACE LOAD command” on page 165 | Lists the load modules or DLLs in the TRACE LOAD table. |

## LIST (blank) command

Displays the Source Identification panel, where you associate compile units with the names of their respective listing, source, or separate debug file. This association controls what z/OS Debugger displays in the Source window. LIST is equivalent to PANEL LISTINGS and PANEL SOURCES.

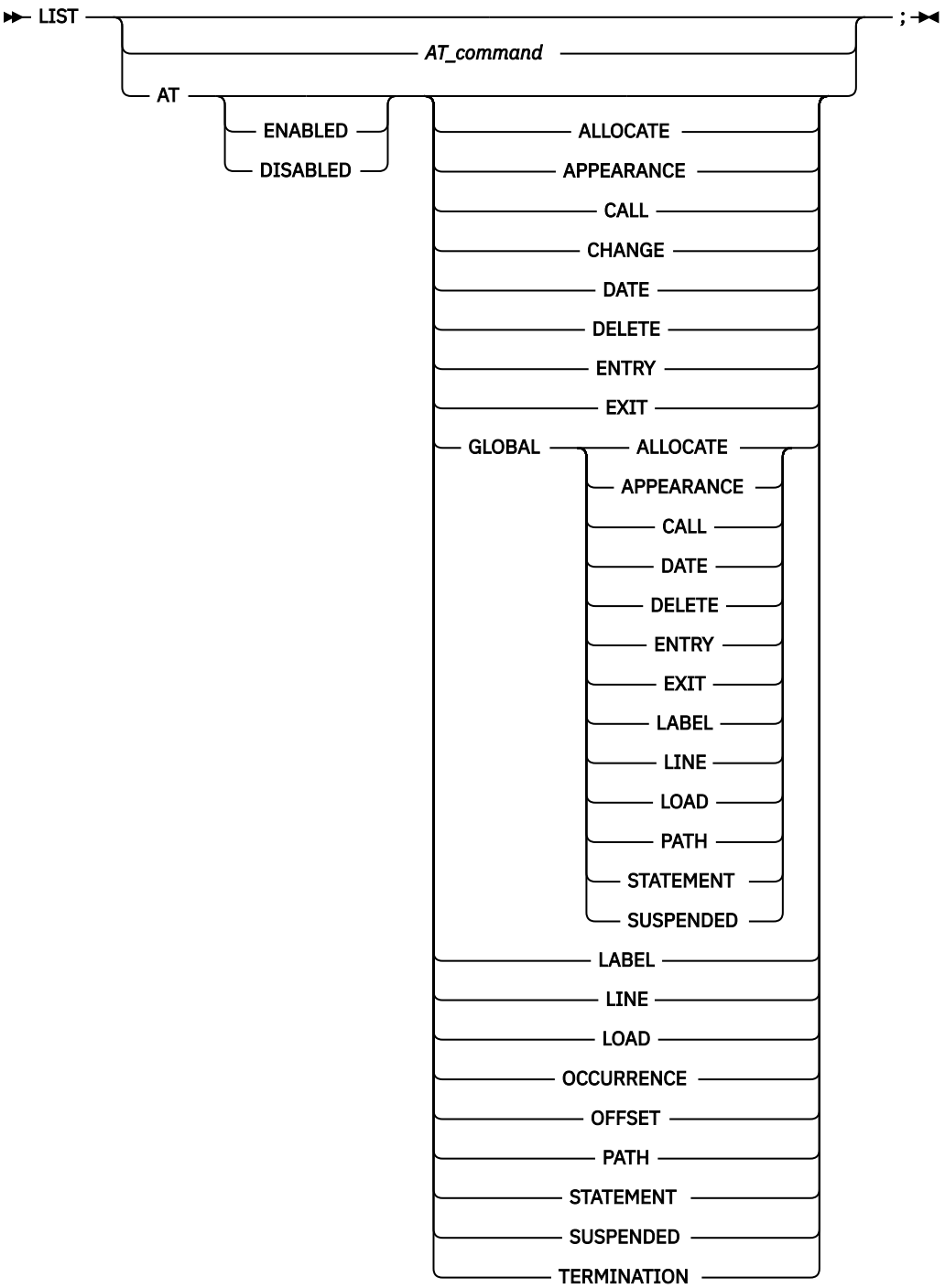
Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

“PANEL command (full-screen mode)” on page 183

## LIST AT command

Lists the currently defined breakpoints, including the action taken when the specified breakpoint is activated. If no action is defined, z/OS Debugger 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

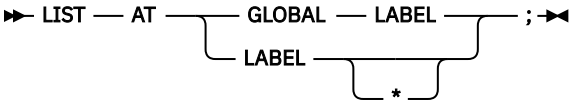
```
LIST AT CHANGE structure.un.m;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"AT command" on page 37

## LIST AT command (remote debug mode)

Lists the currently defined AT GLOBAL LABEL or AT LABEL breakpoints.




**GLOBAL**
Lists whether the AT GLOBAL LABEL breakpoint is defined

**LABEL**
Lists the AT LABEL breakpoints that are defined.

**Usage notes**

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

  ```
  LIST AT LABEL *;
  ```

  or

  ```
  LIST AT GLOBAL LABEL;
  ```

- If you have only a global breakpoint set and you specify LIST AT LABEL command without either an asterisk (*) or GLOBAL keyword, you get a message that says such breakpoints do not exist.

## LIST CALLS command

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 the current active enclave in the process.

**Usage notes**

- For Enterprise COBOL for z/OS Version 5, when a program contains nested programs and your current execution point is inside one of these nested programs, the output of `LIST CALLS` command shows the main program and the nested programs.

  Example:

  – At ENTRY in COBOL program

  ```
  NEST3TS ::> MYMAIN :> EST1A :> NT2A :> ESA
  ```

  – From LINE 62.1 in COBOL program

  ```
  NEST3TS ::> MYMAIN :> MAI :> NEST1A :> NEST2A
  ```

  – From LINE 42.1 in COBOL program

  ```
  NEST3TS::> MYMAIN :> MYMN :> NT1A
  ```

  – From LINE 19.1 in COBOL program

  ```
  NEST3TS::> MYMAIN :> MYMAIN
  ```

  NEST1A, NEST2A, and NEST3A are all nested programs.
- For Enterprise COBOL for z/OS Version 5, if the actual execution of your program is in one of the declarative sections, the output of the `LIST CALLS` command shows an extra entry for the declarative.
- 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.

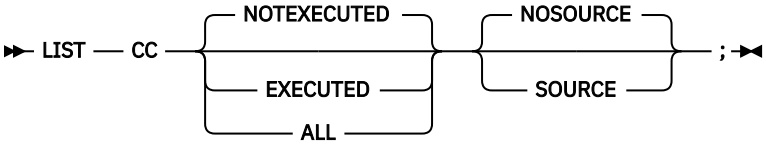
**Example**

Display the current dynamic chain of active blocks.

```
LIST CALLS;
```

# LIST CC command

Lists code coverage data.




**NOTEXECUTED**
Lists all unexecuted statements in all of the CUs in the scope of the CC START command.

**EXECUTED**
Lists all executed statements in all of the CUs in the scope of the CC START command.

**ALL**
Lists all statements in all of the CUs in the scope of the CC START command, indicating which have been executed and which have not.

**SOURCE**
Displays the source statement.

**NOSOURCE**
Specifies that the source statement will not be displayed.

**Usage notes**

- LIST CC lists all captured code coverage data.
- A lowercase 'x' following the statement number indicates that the statement was executed. If the 'x' is not present, the statement was not executed.
- CC START must be in effect for code coverage data to exist.
- CC STOP causes all code coverage data to be deleted.

**Examples**

The following examples show the output displayed in the z/OS Debugger Log.

**LIST CC;**

In this example, the user enters CC START in IBTH013.

```
0059 Code Coverage: not executed in IBTH013
0060 76.1
0063 79.1
0069 Total Statements=50 Total Statements Executed=25  Percent
Executed=50
```

**LIST CC Executed;**

In this example, the user enters CC START in IBTH013.

```
0059 Code Coverage: executed in IBTH013
0061 77.1 x
0062 78.1 x
0063 80.1 x
0064 Total Statements=24 Total Statements Executed=10  Percent
Executed=42
```

**LIST CC SOURCE;**

In this example, the user enters CC START in IBTH013 and has entry break points set in IBTH013A and IBTH013B.

```
0059 Code Coverage: not executed in IBTH013
0060 76.1
0061       76        DISPLAY "IBTH013 - COBOL MAIN BEGINNING".
0062 79.1
0063       79        MOVE SPACES TO WK-TEXT-128-G
0064 Total Statements=24  Total Statements Executed=10  Percent Executed=42
0065 Code Coverage: not executed in IBTH013A
0066 45.1
0067       45        MOVE 0 TO WK-TIME.
0065 Total Statements=48  Total Statements Executed=20  Percent Executed=42
0066 Code Coverage: not executed in IBTH013B
0067 1103.1
0068       1103      MOVE SPACES TO WK-TEXT-128-G.
0069 Total Statements=72  Total Statements Executed=30  Percent
Executed=42
```

**LIST CC ALL SOURCE;**

In this example, the user enters CC START in IBTH013.

```
0059 Code Coverage: All in IBTH013
0060 76.1
0061       76        DISPLAY "IBTH013 - COBOL MAIN BEGINNING".
0062 77.1 x
```

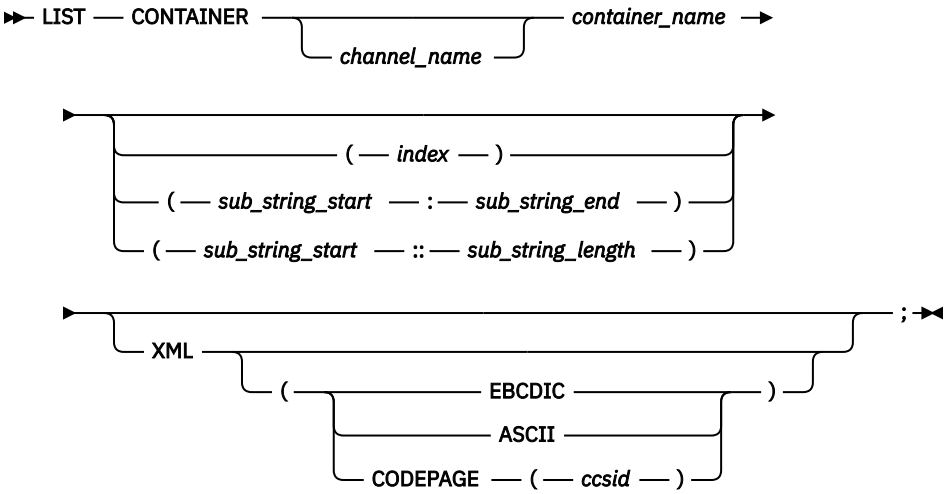
```
0063       77        MOVE SPACES TO PROG.
0064 78.1 x
0065       78        MOVE SPACES TO ZED.
0066 79.1
0067       79        MOVE SPACES TO WK-TEXT-128-G.
0068 80.1 x
0069       80        END.
0069 Total Statements=50  Total Statements Executed=25  Percent
Executed=50
```

**Related references**

"CC command" on page 85

## LIST CONTAINER command

Displays the contents of a container.




***channel_name***
The name of the channel that z/OS Debugger searches through to find a container. If you do not provide a channel name, z/OS Debugger searches through the current channel.

***container_name***
The name of the container.

***index***
A decimal or hexadecimal value indicating the location of a single byte in the container to display.

***sub_string_start***
A decimal or hexadecimal value indicating the starting location of a series of bytes to display.

***sub_string_end***
A decimal or hexadecimal value indicating the ending location of a series of bytes to display.

***sub_string_length***
A decimal or hexadecimal value indicating the number of bytes to display.

**XML**
Indicates that the specified area contains a complete XML 1.0 or 1.1 document. The specified area is passed to the z/OS XML parser for processing. If the parser detects any syntax errors, the error data is shown in the z/OS Debugger log. Otherwise, z/OS Debugger displays a formatted version of the XML document in the z/OS Debugger log.

**EBCDIC**
Indicates that the specified area contains EBCDIC characters.

**ASCII**
Indicates that the specified area contains ASCII characters.

**CODEPAGE**
Indicates that the specified area contains characters in the specified code page.

***ccsid***
Specifies the Coded Character Set Identifiers used to encode the XML. z/OS Debugger uses the z/OS Unicode Services to convert the characters in the XML from this code page to the code page specified by the EQAOPTS CODEPAGE command before the characters are displayed on the 3270 terminal. The *ccsid* can be a decimal number in the range 1 to 65535.

**Usage notes**

- You can use the LIST CONTAINER command in remote debug mode, except for the XML option.
- For PL/I, COBOL, LangX COBOL, assembler, and disassembly, if the name is mixed case or case sensitive, you must enclose the name in quotation marks (") or apostrophes (').
- For C and C++, the name is always treated as case sensitive, even if it is not enclosed in quotation marks (").
- XML is supported only when you run on z/OS Version 1.8 or later.
- If you specify XML but not EBCDIC, ASCII, nor CODEPAGE, z/OS Debugger attempts to detect if the encoding of the XML document is EBCDIC or ASCII.
- Some information in the XML document (for example, most of the DTD specification and some white space) might not be listed because the z/OS XML parser does not return it to z/OS Debugger.

**Examples**

- For PL/I, COBOL, LangX COBOL, assembler, or disassembly, enter the following command to display two bytes, starting at the first byte, of container CONNAME, which is in channel CHNAME:

```
LIST CONTAINER CHNAME CONNAME ( 1 :: 2);
```

- For PL/I, COBOL, LangX COBOL, assembler, or disassembly, enter the following command to display two bytes, starting at the first byte, of container CONNAME, which is in channel chname:

```
LIST CONTAINER 'chname' CONNAME ( 1 :: 2);
```

- For C/C++, enter the following command to display the contents of container conName, which is in the current channel:

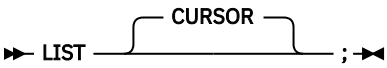
```
LIST CONTAINER conName;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
"Displaying containers and channels" in the IBM z/OS Debugger User's Guide
**Related references**
“DESCRIBE command” on page 103
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

## LIST CURSOR command (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.

```
LIST CURSOR
```

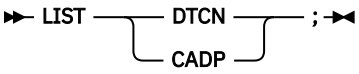
- A COBOL program has a statement of the form:

```
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 DTCN or CADP command

List the programs and compile units that were disabled by the DISABLE CADP or DISABLE DTCN command.




**DTCN**
List the load modules and compile units that were disabled by the DISABLE DTCN command.

**CADP**
List the programs and compile units that were disabled by the DISABLE CADP command.

**Usage note**

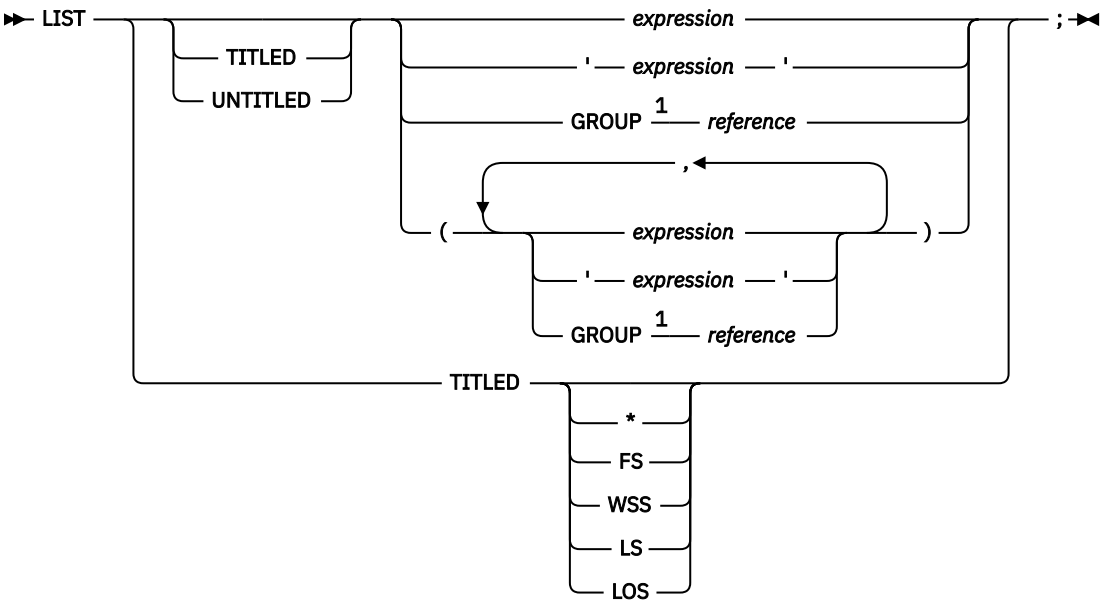
You can use the LIST DTCN or LIST CADP command in remote debug mode.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"ENABLE command" on page 113
"DISABLE command" on page 107
Appendix A, "z/OS Debugger commands supported in Debug Tool compatibility mode," on page 517

## LIST expression command

Displays values of expressions.




Notes:

[1] Only for COBOL.

**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 *lvalues*. For COBOL, this is the default *except* for expressions consisting of only a single constant. For assembler, disassembly, and LangX COBOL, this is the default for expressions that are valid as receivers of a z/OS Debugger 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 lvalues*. For COBOL, this is the default for expressions consisting of only a single constant. For assembler, disassembly, and LangX COBOL, this is the default for expressions that are not valid as receivers of a z/OS Debugger assembler assignment statement. For the LIST command, an expression also includes character strings enclosed in either quotation marks (") or apostrophes ('), 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);`.

**GROUP (COBOL)**
Displays a reference as an EBCDIC character string. If you specify GROUP on an elementary item, it has no effect. The operand following the GROUP keyword must be a reference; for example, `LIST TITLED GROUP y;`. You cannot specify expressions.

***expression***
An expression valid in the current programming language other than LangX COBOL.

**'*expression*'**
A valid LangX COBOL expression enclosed in apostrophes (').

**Usage notes**

- For Enterprise COBOL for z/OS Version 5, this command has the following usage notes:
  - When you use the LIST command to display a record or group, the levels of the record or group are shown as they are declared in the program.
  - When you use the LIST command on an array (table), the output shows the value of each array element, one per line. The output changes in the following ways:
    - The subscripts for each array element are shown in parentheses after the name of the array. This matches how array subscripts are specified in COBOL. Previously, each line of the output had the word "SUB" at the beginning, and the subscripts were shown in parentheses after this word.
    - If the array has subordinate data items, which means the elements are groups but not scalars, the value of each subordinate data item is displayed for an array element before the values of the next array element are displayed. In other words, array element n is displayed before array element n+1. The output better reflects how the array is organized in memory.
  - When you use the LIST command to list a single member of an array, the output is the same as a variable of the given type.

    Examples:

    Given these arrays:

    ```
    5 ARR1 OCCURS 2 TIMES INDEXED BY IX1.
      10 X PIC 99 USAGE BINARY.
      10 Y PIC 99 USAGE BINARY.
    5 ARR2 PIC 9 USAGE BINARY OCCURS 2 TIMES INDEXED BY IX2.
    ```

    The output is as follows:

    ```
    LIST ARR1 ( 1 ) ;
    10 X of 05 ARR1 = 00001
    10 Y of 05 ARR1 = 00002

    LIST ARR2 ( 1 ) ;
    ARR2 ( 1 ) = 00000
    ```
  - The output of `LIST %EPA` when inside a declarative section shows the internal entry point of the declarative but not the entry point of the program.
  - When you use the `LIST TITLED *` command, only those variables for active blocks are shown.
  - When you use the `LIST expression` command to display the value of an element of an array and you do not specify an array subscript or index, the value of the first element is displayed. That is, it defaults to `Index = 1`.
  - You cannot use index data items as an index name when you reference an array element. For example, if you have the following declaration in your program, `77 IXDI1 USAGE IS INDEX`, you cannot use `IXDI1` as index in `LIST ARR(IXDI1)`.
  - The number of digits that are displayed after you use the LIST command for a numerical value is consistent with what is specified in the *Enterprise COBOL for z/OS Version 5 Programming Guide*.
  - The result of displaying an expression with the LIST command shows the sign if either operand is signed. In the following example, `LIST SIGNED_ONE + TWO = +0003`, the first operand is signed.

  - When you use the LIST command to display a variable of National type, the output shows the N prefix. Example: `NAT= N'abcde'`.
  - You cannot display the values of variables in a Nested Program (block) if they are declared in the Local or Linkage Section and the Nested Program (block) is not active.
  - When you use the `LIST TITLED` command, the output that displays the value of a variable that has unprintable character is shown as dots. It shows HEX values only if you use %HEX.
  - For an object-oriented program, if a variable name is a part of both the Object name and the Method name, and you use the LIST command with this variable, you can get an error message to indicate that it is ambiguous.
  - The DBCS string is prefixed with a G, for example, `V_-DBCS = G'DBCS string'`.
- 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 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.
- z/OS Debugger 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).
- If a character variable contains character data that cannot be displayed in its declared data type, z/OS Debugger displays the data with a special character. The topic "How z/OS Debugger handles characters that can't be displayed in their declared data type" in the IBM z/OS Debugger User's Guide describes what z/OS Debugger does in this situation. If you display the data in hexadecimal, it will require twice as many bytes. The maximum number of bytes that can be displayed is 65,535.
- 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.
- You can enter the L prefix command by using the Source window prefix area to display the value of the variables on that line. For the list of supported compile units, see “L prefix command (full-screen mode)” on page 153.
- For Enterprise PL/I programs, to change the display format so that z/OS Debugger displays arrays and elements as they are stored in memory, enter the `SET LIST BY SUBSCRIPT ON` command.
- If the Log window is not visible, z/OS Debugger displays the result of the `LIST` *`expression`* command in the List pop-up window.

**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 z/OS Debugger variable %ADDRESS.

```
LIST %ADDRESS;
```

- In the disassembly view, display the value of register 1 (R1), which is the value of z/OS Debugger variable %R1.

```
LIST R1 ;
```

- In COBOL, display the names and values of variables defined in the File Section.

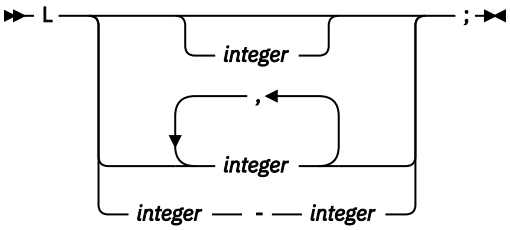
```
LIST TITLED FS;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“expression” on page 14
“SET LIST TABULAR command” on page 239
“L prefix command (full-screen mode)” on page 153

## L prefix command (full-screen mode)

The L prefix command, which you enter through the prefix area of the Source window, displays the value of an operand or operands on that line in the Log window.




***integer***
Identifies specific operands to be listed. If you do not specify an *integer*, z/OS Debugger lists all operands. If you use a single number or the form *1,2,3*, z/OS Debugger lists the specified operand or operands. If you use the form 1-4, z/OS Debugger lists operands 1 through 4.

For programs other than assembler and disassembly, *integer* identifies the position of a variable on a line, beginning from the left. The first variable on the line is position 1, the second variable on the line is position 2, and this pattern repeats until there are no more variables. If a variable is on the line more than once, only the first instance of the variable is assigned a position number.

For assembler and disassembly programs, *integer* identifies operands of the machine instruction. z/OS Debugger numbers them from left to right with the first operand numbered operand 1, the second operand numbered operand 2, and repeating the pattern until there are no more operands. If you do not specify an *integer*, z/OS Debugger lists all operands referenced explicitly or implicitly by the instruction. If you specify any form of *integer*, z/OS Debugger lists only the operands explicitly referenced by the specified operand or operands.

**Usage notes**

- For C/C++, integer values cannot be specified.
- The L prefix command can be entered only on lines that have valid executable statements.
- You can enter the L prefix command on multiple lines.
- The L prefix command works only for the following compile units:

  - Assembler or disassembly compile units
  - Enterprise COBOL compile units
  - Enterprise PL/I compile units compiled with Enterprise PL/I for z/OS, Version 3.6 or 3.7 with the PTF for APAR PK70606 applied, or later
  - C/C++ compile units compiled with the z/OS 2.1 XL C/C++ compiler or later, with DEBUG(FORMAT(DWARF)) option.
- You cannot use the L prefix command on a line that is in a block that is not currently active.
- The following notes apply when you use the L prefix command in an assembler or disassembly program:
  - When you specify *integer*, it applies to an entire machine instruction operand, not to a single symbol. For example, in the following instruction, operand 1 is the storage referenced by "SYM1-SYM2(LEN,R8)" and operand 2 is the storage referenced by SOURCE:

    ```
    MVC  SYM1-SYM2(LEN,R8),SOURCE
    ```

  - z/OS Debugger uses the current values in a register to evaluate any registers referenced by an instruction. When you reference an instruction that is not the instruction where the program is suspended, the current values in a register might differ from what the values would be if z/OS Debugger stopped the program at the instruction you referenced.
  - The L prefix command cannot access mask fields, immediate data fields, and any other constants imbedded in the machine instructions. However, z/OS Debugger does number these fields when it numbers the operands.
  - For instructions that might be coded using extended mnemonics (BC, BCR, and BRC), z/OS Debugger cannot determine whether the base form or the extended mnemonic was used. Therefore, you can use both 1 and 2 to refer to the operand representing the branch target.

**Examples**

The following set of examples use the following lines of code:

```
...
             293     move 0 to c; move 0 to b; move 0 to IND; move  b  to a;
...
             319     if a + b < b + c
             320        then move ind to c;
             321     end-if;
...
```

- To display the value of IND on line 293, enter the L3 command in the prefix area of line 293.
- To display the value of c on line 319, enter the L3 command in the prefix area of line 319. The position of c is not 4 because b is counted only once, the first time it is encountered, which is to the left of the < operator. The second b, which is to the right of the < operator, is not assigned a position number.
- To display the value of all variables on line 293, enter the L command in the prefix area of 293.

The next set of examples use the following lines of assembler source code:

```
...
200          L    R6,=X'31BA4038'
201          STM  R1,R4,0(R6)
202          TM   X'01',FLAGS
203
...
```

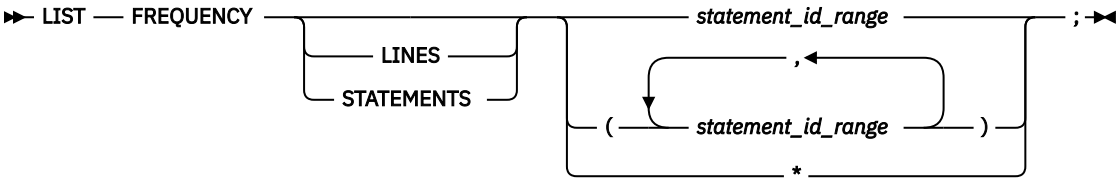
- Enter L on line 201. z/OS Debugger lists the following registers and memory locations: R1, R2, R3, R4, R6, and the sixteen bytes of storage at location X'31BA4038'.
- Enter L1-2 on line 201. z/OS Debugger lists R1 and R4.
- Enter L1 on line 202. z/OS Debugger displays an error message because the L prefix command cannot access mask and immediate fields.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“LIST expression command” on page 149

# LIST FREQUENCY command

Lists statement execution counts.




*****
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.

```
LIST FREQUENCY 1 - 20;
```

- List frequency and statement for statements 18 - 19:

```
LIST FREQUENCY LINES 18-19;
```

- List frequency for all statements in the currently qualified compile unit.

```
LIST FREQUENCY *;
```

- List frequency for all statements in all compile units.

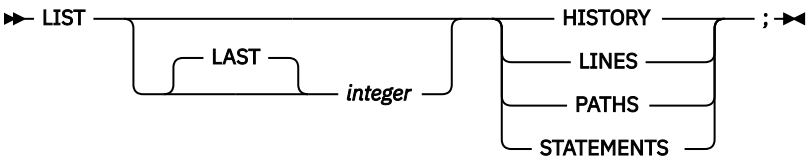
```
AT TERMINATION LIST FREQUENCY *;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“statement_id_range and stmt_id_spec” on page 16
“SET FREQUENCY command” on page 230

# LIST LAST command

Displays a list of recent entries in the history table.




***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 z/OS Debugger gained control.

```
LIST LAST 5 HISTORY;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"SET HISTORY command" on page 231

## LIST LDD command

Displays the list of LDD commands known to z/OS Debugger.


**Usage notes**

- Use the LIST LDD command if you want to see a list of LDD commands that z/OS Debugger knows about.
- The output of the LIST LDD command is sorted alphabetically by compile unit name.
- When debugging C/C++ applications in EXPLICITDEBUG mode, the LIST LDD command might display just one LDD entry that matches one of the several LDD commands containing main as the compile unit name.
- You can use the LIST LDD command in remote debug mode.

**Examples**

- To display the LDD commands known to z/OS Debugger, specify:

```
LIST LDD;
```

You might get results similar to the following output:

```
1. LDD TBND003::>TBND003A;
2. LDD MYPROG;
```

- To display the LDD commands known to z/OS Debugger, specify:

```
LIST LDD;
```

You might get results similar to the following output:

```
There are no LDD commands.
```

**Related references**
"CLEAR command" on page 86

## LIST LINE NUMBERS command

Equivalent to `LIST STATEMENT NUMBERS`.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"LIST STATEMENT NUMBERS command" on page 162

## LIST LINES command

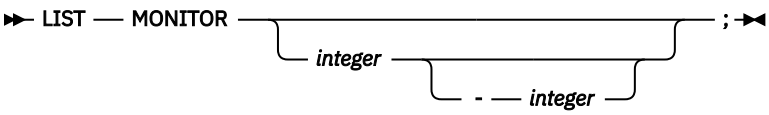
Equivalent to `LIST STATEMENTS`.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"LIST STATEMENTS command" on page 162

## LIST MONITOR command

Lists all or selected members of the current set of `MONITOR` commands and any suspended `MONITOR LOCAL` commands.




***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.
- If *integer* is not specified, both the active monitors and any suspended local monitors are listed.

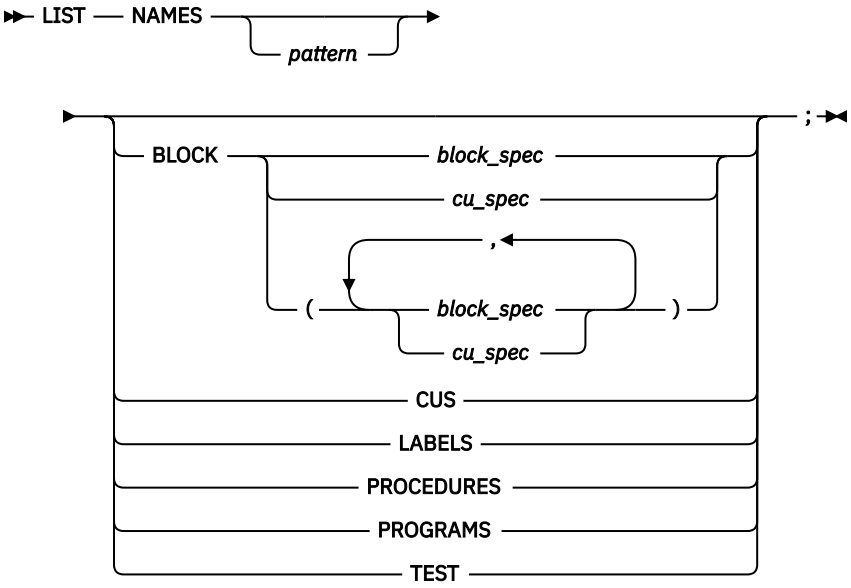
**Example**

List the fifth through the seventh commands currently being monitored.

```
LIST MONITOR 5 - 7;
```

# LIST NAMES command

Lists the names of variables, programs, or z/OS Debugger 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.




***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 quotation marks (") or apostrophes (').

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:

| C | Assembler, COBOL, and LangX COBOL | PL/I |
|---|---|---|
| "ABC" | "A5" | 'MY' |
| | 'A5' | |

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.

**LABELS**
Displays the names of all section and paragraph names in a COBOL or LangX COBOL program, and the names of all instruction labels in an assembler program. Supported only for COBOL and assembler.

**PROCEDURES**
Displays the z/OS Debugger procedure names.

**PROGRAMS**
Is equivalent to CUS.

**TEST**
Displays the z/OS Debugger session variable names.

**Usage notes**

- For Enterprise COBOL for z/OS Version 5, the output of the command LIST NAMES shows only level 0 and 77 data items and index names. Subordinate data items within records are not displayed.
- For Enterprise COBOL for z/OS Version 5, when you issue LIST NAMES CUS and a load module is linked with more than one Enterprise COBOL for z/OS Version 5 compilation unit, the output includes all Enterprise COBOL for z/OS Version 5 compilation units in the load module.
- For Enterprise COBOL for z/OS Version 5, when you issue the LIST NAMES command for a program with nested programs, the output for each block or nested program includes only the variables declared in the block.
- For Enterprise COBOL for z/OS Version 5, when you issue the LIST NAMES LABEL command for a program with nested blocks or programs, only the labels in the current block are displayed.
- 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.
- For LangX COBOL, if you specify the EQALANGX file as the source of debug information, when you enter the LIST NAMES LABELS command, z/OS Debugger might not display all of the labels because EQALANGX did not identify them with the LABEL attribute.

**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 z/OS Debugger 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);
```

- Display all section and paragraph names that begin with the letters "LAB". The currently qualified program is COBOL.

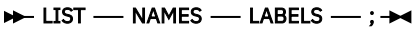
```
LIST NAMES "LAB*" LABELS;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“block_spec” on page 12
“cu_spec” on page 13

# LIST NAMES LABELS command (remote debug mode)

Displays the names of all sections and paragraphs in a COBOL program, and the names of all instruction labels in an assembler program.




**Usage Notes®**

- Use the Debug Console Command line to enter this command.
- This command displays the labels in the current executing program.
- For Enterprise COBOL for z/OS Version 5, if you specify the `LIST NAMES LABEL` command for a program with nested blocks or multiple programs, only the labels in the current block are displayed.

**Examples**

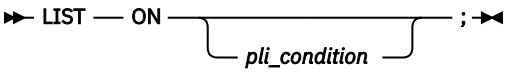
If you want to display the names of all sections and paragraphs in the current program in COBOL, specify the following commands:

```
LIST NAMES LABELS;
```

```
EQA4837I The following LABELS are known in IBCD590
PARA-IBCD590
L100
L101
```

# LIST ON (PL/I) command

Lists the action (if any) currently defined for the specified PL/I conditions.




**`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.

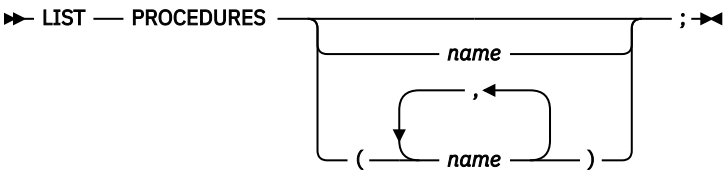
```
LIST ON ZERODIVIDE;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“ON command (PL/I)” on page 181

# LIST PROCEDURES command

Lists the commands contained in the specified z/OS Debugger `PROCEDURE` definitions.




***name***
A valid z/OS Debugger 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 z/OS Debugger procedure p2.

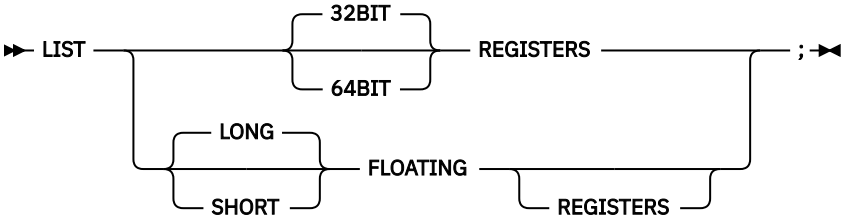
```
LIST PROC p2;
```

- List the procedures abc and proc7.

```
LIST PROCEDURES (abc, proc7);
```

## LIST REGISTERS command

Displays the current register contents.




**REGISTERS**
Displays the General Purpose Registers. 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.

**32BIT**
Displays the 32-bit General Purpose Registers (%GPR*n* and %GPRH*n*).

**64BIT**
Displays the 64-bit General Purpose Registers (%GPRG*n*).

**LONG**
Displays the value of the long-precision floating-point registers.

**SHORT**
Displays the value of the short-precision floating-point registers.

**FLOATING**
Displays the floating-point registers.

**Usage note**

If your program is running on hardware that does not support 64-bit instructions or your program is suspended at a point where the 64-bit general-purpose registers are not available, z/OS Debugger displays only the sixteen, base 32-bit general-purpose registers.

**Examples**

- Display the General Purpose Registers at the point of a program interruption:

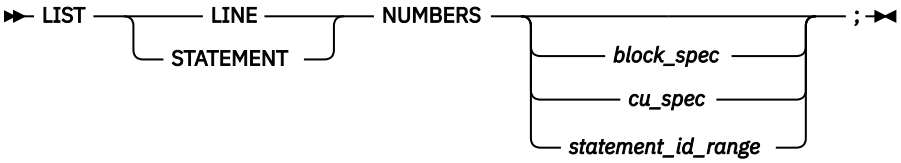
```
LIST REGISTERS;
```

- Display the floating-point registers.

```
LIST FLOATING REGISTERS;
```

## LIST STATEMENT NUMBERS command

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`.

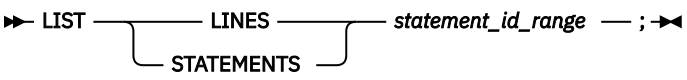
```
LIST STATEMENT NUMBERS earnings;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“block_spec” on page 12
“cu_spec” on page 13
“statement_id_range and stmt_id_spec” on page 16

## LIST STATEMENTS command

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.



**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.
- `LIST LINES` or `LIST STATEMENTS` without a `statement_id_range` are not valid to list one or more lines, or statements, from a file. However, they are accepted commands, because they are valid in the context of the `LIST LAST` command.

**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.

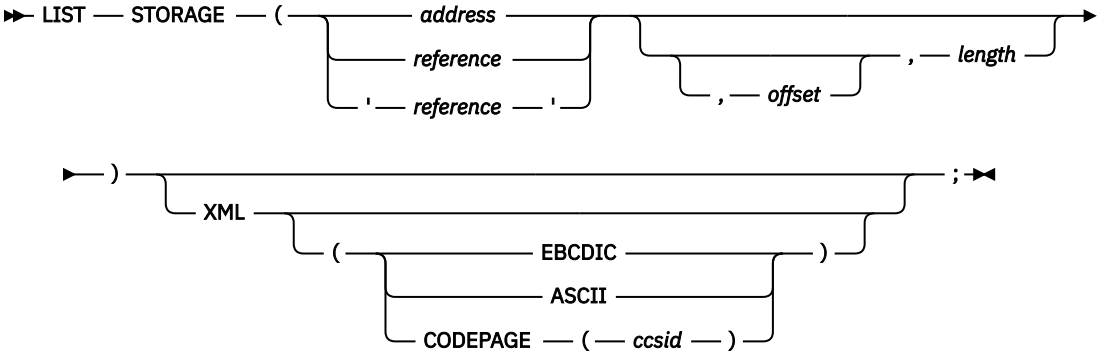
```
LIST STATEMENT 100;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“statement_id_range and stmt_id_spec” on page 16
“LIST LAST command” on page 155

# LIST STORAGE command

Displays the contents of storage at a particular address in hexadecimal or XML format.




***address***
The starting address of storage to be listed.

***reference***
A variable whose storage location is to be listed.

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*'**
A LangX COBOL variable whose storage location is to be listed. A LangX COBOL reference must be enclosed in apostrophes (').

***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, you must follow the same syntax rules for *address*. 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.

**XML**
Indicates that the specified area contains a complete XML 1.0 or 1.1 document. The specified area is passed to the z/OS XML parser for processing. If the parser detects any syntax errors, the error data is shown in the z/OS Debugger log. Otherwise, z/OS Debugger displays a formatted version of the XML document in the z/OS Debugger log file.

**EBCDIC**
Indicates that the specified area contains EBCDIC characters.

**ASCII**
Indicates that the specified area contains ASCII characters.

**CODEPAGE**
Indicates that the specified area contains characters in the specified code page.

***ccsid***
Specifies the Coded Character Set Identifiers used to encode the XML. z/OS Debugger uses the z/OS Unicode Services to convert the characters in the XML from this code page to the code page specified by the EQAOPTS CODEPAGE command before the characters are displayed on the 3270 terminal. The *ccsid* can be a decimal number in the range 1 to 65535.

**Usage notes**

- For C and C++, if *reference* is a pointer, z/OS Debugger displays the contents at the address given by that pointer.
- Using z/OS Debugger, 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 z/OS Debugger 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, z/OS Debugger displays the storage at the address contained in the referenced GPR.
  - For assembler and disassembly, you must use the indirection notation (%GPR1->) to instruct z/OS Debugger 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.
- XML is supported only when you run on z/OS Version 1.8 or later.
- If you specify XML but not EBCDIC, ASCII, nor CODEPAGE, z/OS Debugger attempts to detect if the encoding of the XML document is EBCDIC or ASCII.
- Some information in the XML document (for example, most of the DTD specification and some white space) might not be listed because the z/OS XML parser does not return it to z/OS Debugger.

- If you specify *address* with more than 8 significant digits or if *reference* references 64-bit addressable storage, z/OS Debugger assumes that the storage location is 64-bit addressable storage. Otherwise, z/OS Debugger assumes that the storage location is 31-bit addressable storage.

**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 topics for more information related to the material discussed in this topic.

**Related references**
“address” on page 11
“references” on page 15

## LIST TRACE LOAD command

Displays the entries in the TRACE LOAD table that were created since the `TRACE LOAD START` command was issued.

The syntax of this command is as follows:


**Usage notes**

- Use the `LIST TRACE LOAD` command if you want to see a list of the load modules or DLLs loaded since the `TRACE LOAD START` command was issued.

**Examples**

To display the entries in the TRACE LOAD table, enter the following command:

```
LIST TRACE LOAD;
```

You might get results similar to the following output:

```
The following were loaded:
IBCD010 loaded from TSFANAY.TEST.LOAD
IBCD010A loaded from TSFANAY.TEST.LOAD
```

**Related references**

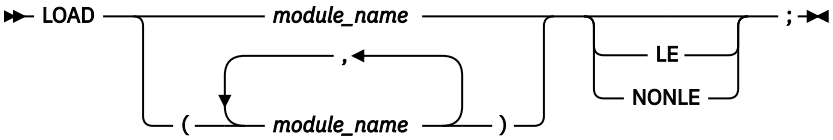
"TRACE command" on page 276.

# 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.




***module_name***
The name of one or more load modules to be loaded by z/OS Debugger.

**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.
- You can enter the SET 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 z/OS Debugger was started, you can use the LOAD command to make the module known to Language Environment.
- You cannot use this command to load a DLL.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

Appendix A, "z/OS Debugger commands supported in Debug Tool compatibility mode," on page 517

# LOADDEBUGDATA command

z/OS Debugger automatically loads the debug information for a compile unit (CU) when all of the following conditions apply:

- The compile unit was written in a Language Environment-enabled, high-level language.
- The compile unit was compiled with the TEST or DEBUG compiler option.
- Explicit debug mode is not active.

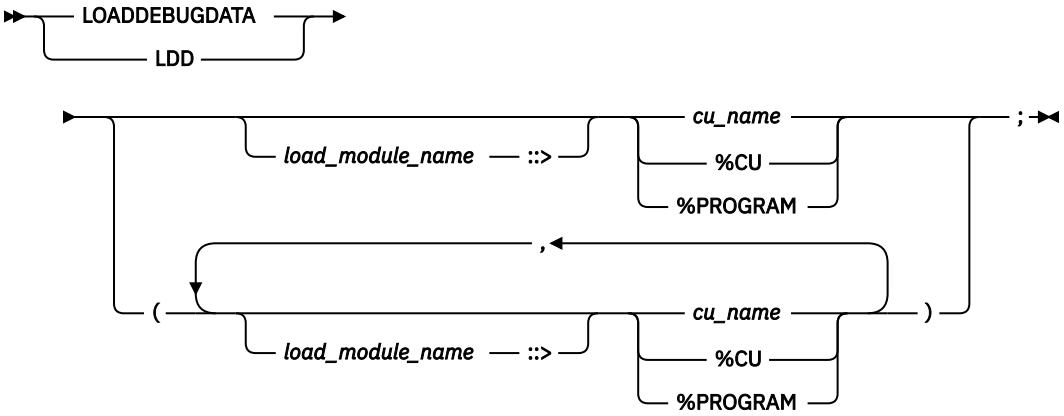
In the following situations, you must use the LOADDEBUGDATA (LDD) command to request that z/OS Debugger load the debug data:

- The compile unit was written in assembler or LangX COBOL.
- Explicit debug mode is active and the compile unit was written in a Language Environment-enabled, high-level language and compiled with the TEST or DEBUG compiler option.

## Using LDD for assembler or LangX COBOL compile units

When you use the LDD command for assembler or LangX COBOL compile units, the LDD command indicates that the compile unit (CU) is an assembler or LangX COBOL CU and z/OS Debugger loads debug data from the default data set name, *userid*.EQALANGX(*cu_name*). If the debug data is stored in a different data set, specify that data set name by using the SET SOURCE command, SET DEFAULT LISTINGS command, or the EQADEBUG DD statement. In remote debug mode, specify the data set name by using the SET DEFAULT LISTINGS command, the EQADEBUG DD statement, or providing the data set name when the remote debugger prompts you for it.

Generate the required debug information by using the EQALANGX program or, if you are debugging an assembler program, by assembling your program through IBM z/OS Debugger Utilities, as described in IBM z/OS Debugger User's Guide.




***load_module_name***
The name of the load module containing the specified compile unit (*cu_name*). If the corresponding load module is known to z/OS Debugger, the specified compile unit must be a disassembly compile unit within the specified load module. If the load module is not known to z/OS Debugger, z/OS Debugger defers the LOADDEBUGDATA command until a load module by the specified name and containing the specified compile unit is loaded. *load_module_name* is folded to upper case, unless it is enclosed in double-quotation marks or the current environment is UNIX System Services.

If you do not specify *load_module_name*, z/OS Debugger applies the LOADDEBUGDATA command to all compile units by the specified name found in any load module.

***cu_name***
The name of the assembler or LangX COBOL compile unit for which the debug data is to be loaded. If the compile unit is not currently known to z/OS Debugger, z/OS Debugger defers the LOADDEBUGDATA command until a compile unit by the specified name becomes known to z/OS Debugger.

**Usage notes** (assembler and LangX COBOL form of LDD)

- 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, z/OS Debugger 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 compile units in remote debug mode.
- After z/OS Debugger 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.
- A deferred LDD creates an implicit NAMES INCLUDE for the target of the deferred LDD.

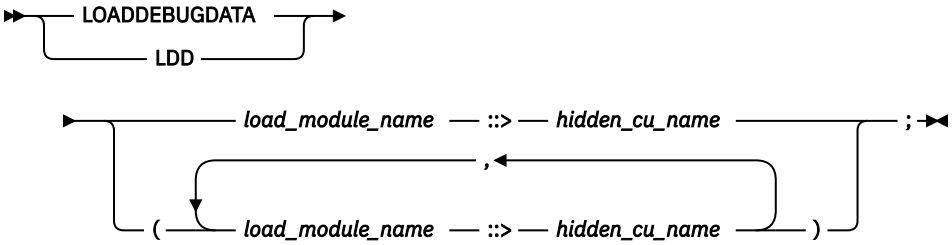
Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"CLEAR command" on page 86
"LIST LDD command" on page 156
"SET LDD command" on page 235
Appendix A, "z/OS Debugger commands supported in Debug Tool compatibility mode," on page 517

## Using LDD for high-level language compile units in explicit debug mode

When explicit debug mode is active, z/OS Debugger loads debug data only for compile units that you want to debug. For an assembler compile unit, use the LDD command as described in "Using LDD for assembler or LangX COBOL compile units" on page 167. The remainder of this topic describes how to use the LDD command to load debug data for a compile unit written in a high-level language.

You can indicate that you want to debug a compile unit in this mode by using the LDD command. You must write the compile unit in a Language Environment-enabled high-level language and compile it with the TEST or DEBUG compiler option.




***load_module_name***
The name of the load module containing the specified compile unit (*hidden_cu_name*). If z/OS Debugger does not know the load module, z/OS Debugger defers the LOADDEBUGDATA command until a load module by the specified name and containing the specified compile unit is loaded. *load_module_name* is folded to upper case, unless it is enclosed in double-quotation marks or the current environment is UNIX System Services.

***hidden_cu_name***
The name of the hidden compile unit for which the debug data is to be loaded. If this compile unit is not currently known to z/OS Debugger, z/OS Debugger defers the LOADDEBUGDATA command until a compile unit by the specified name becomes known to z/OS Debugger.

The following table describes how *hidden_cu_name* corresponds to the compile unit name, depending on the programming language:

| Programming language | Hidden CU name | Same as CU name? |
|---|---|---|
| COBOL | First program name in the source file | Yes |
| PL/I | External procedure name | Yes |
| Enterprise PL/I | External procedure name | No |
| C | First external function name in the source file | No |
| Assembler | CSECT name | Yes |

If the load module is currently loaded, enter the DESCRIBE LOAD command and review the output to determine the value for *hidden_cu_name*. If the load module is not currently loaded, enter the AT LOAD command. After z/OS Debugger gains control because of AT LOAD, you can run the DESCRIBE LOAD command and review the output to determine the value for *hidden_cu_name*.

When explicit debug mode is active, z/OS Debugger automatically loads debug data for compile units without using an LDD command in each of the following situations:

- You specified both a load module name and a compile unit name that is not a source file name enclosed in quotes in a deferred AT ENTRY command and the compile unit name was the same name that would have been used as the *hidden_CU_name* on the LDD command.
- The compile unit name is the entry point of the initial load module of an enclave, or a load module for which an AT LOAD command was entered.
- The NAMES INCLUDE command either explicitly or implicitly included both the load module and compile unit name. In the case of a compile unit name, it must be the same name that would have been used as the *hidden_CU_name* on the LDD command (not a source file name enclosed in quotes).

**Usage notes** (high-level language form of LDD)

- You can use this command in remote debug mode.
- 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, z/OS Debugger saves and restores all LDD settings.
- **For CICS only**: When a DTCN profile is active for a full screen mode debugging session, z/OS Debugger preserves all LDD settings until the DTCN profile is deleted or the terminal session is terminated.
- After z/OS Debugger successfully processes a LOADDEBUGDATA command for a compile unit, if the compile unit is deleted and then appears later, z/OS Debugger runs an implicit LDD command for the compile unit.
- You cannot enter the LDD command for the same compile unit more than once.
- A deferred LDD creates an implicit NAMES INCLUDE for the target of the deferred LDD.
- You cannot load debug data for a high-level language compile unit that is currently active. For example, if compile unit A calls compile unit B, you cannot stop in compile unit B, then run an LDD command on compile unit A.

Refer to the following topics for more information related to the material discussed in this topic.

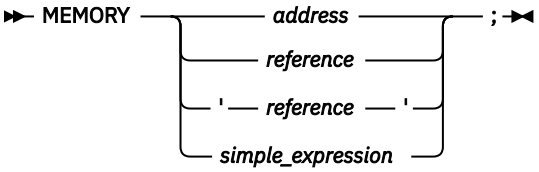
**Related references**
“CLEAR command” on page 86
“LIST LDD command” on page 156
“SET EXPLICITDEBUG command” on page 228
“SET LDD command” on page 235
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

# MEMORY command

Specifies an address to use as the starting address for the memory displayed in the Memory window.

If the address you specify is invalid, z/OS Debugger displays an error message.

The MEMORY command cannot be saved and restored.




***address***
The address to use as the starting address for the memory displayed in the Memory window.

***reference***
A variable whose location in memory is used as the starting address of the memory displayed in the Memory window.

**'*reference*'**
A LangX COBOL variable whose location in memory is used as the starting address of the memory displayed in the Memory window.

***simple_expression***
The address with a positive or negative hexadecimal or integer displacement. The resulting value is the starting address of the memory displayed in the Memory window.

**Usage notes**

- For COBOL, if you specify a variable with reference modification, then the storage location of that variable is used as a base address, not the location of the specified reference.
- If you specify *address* with more than 8 significant digits or if *reference* references 64-bit addressable storage, z/OS Debugger assumes that the storage location is 64-bit addressable storage. Otherwise, z/OS Debugger assumes that the storage location is 31-bit addressable storage.
- For C and C++, if *reference* is a pointer, z/OS Debugger displays the contents at the address given by that pointer.

**Examples**

- Display memory starting at X'2503D008' by entering the following command:

```
MEMORY X'2503D008';
```

  This address becomes the base address.
- Display memory starting at the storage location of variable *Employee_name* by entering the following command:

```
MEMORY Employee_name;
```

  The address of *Employee_name* becomes the base address.
- Display memory starting 100 hex bytes after X'0045CB00' by entering the following command:

```
MEMORY x'0045CB00' + x'100'
```

  The base address is X'0045CC00'.

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

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
"z/OS Debugger session panel" in the IBM z/OS Debugger User's Guide
"Switching between the Memory window and Log window" in the IBM z/OS Debugger User's Guide
"Displaying the Memory window" in the IBM z/OS Debugger User's Guide

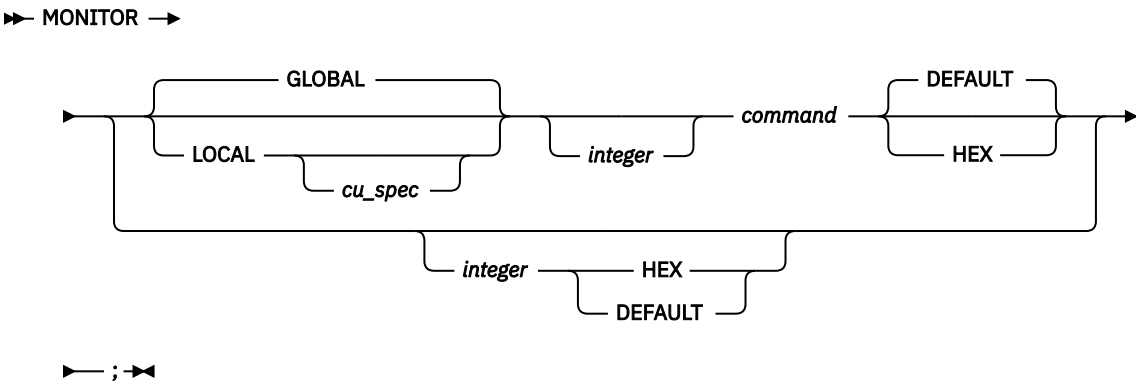
**Related references**
"address" on page 11

# 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

z/OS Debugger 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 z/OS Debugger 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 z/OS Debugger, 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 *n*th command in the MONITOR list and it must be a LIST *expression* command.

**DEFAULT**
Specifies that the value of the variable be displayed in its declared data type. You can specify the DEF parameter only with a MONITOR LIST *expression* command or the MONITOR *n* command where *n* is the *n*th command in the MONITOR list and it must be a LIST *expression* command.

**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 DEF prefix commands operate only on an individual structure element or array element when you enter them in the prefix area associated with that element.
- A monitor number identifies a global monitor command, a local monitor command, or neither.
- Using z/OS Debugger, 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.
- Replacement only occurs if the command identified by the monitor number already exists.
- When SET AUTOMONITOR ON is in effect, z/OS Debugger adds an entry that is not visible after the last active entry in the monitor list. If you specify a *number* and it is either equal to or one more than the last active entry, z/OS Debugger inserts the new MONITOR command in the last active entry and uses the next higher entry for SET AUTOMONITOR ON.

  **Note:** The SET AUTOMONITOR ON command occupies 1 (for CURRENT or PREVIOUS) or 2 (for BOTH) entries in the monitor list. These entries are not included in the list of Monitor commands from the LIST MONITOR command.
- The MONITOR LIST command does not allow the POPUP, TITLED, and UNTITLED options, except TITLED WSS. For more information about the TITLED WSS option, see "LIST expression command" on page 149. If the Working-Storage Section contains large amounts of data, monitoring it can add a substantial amount of overhead and might produce unpredictable results.
- When using the MONITOR LIST command, simple references (or C `lvalues`) 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 suspended when the enclave containing the compile unit terminates or when the load module containing the compile unit is deleted. If the associated compile unit reappears later in the same debugging session, the LOCAL monitors are restored. However, because the original monitor number might be in use at that time, they will not always be restored with the same monitor number.
- 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:
  - Suspending execution in a compile unit written in a language different from the programming language that was in effect when the original MONITOR command was entered.
  - Entering the SET PROGRAMMING LANGUAGE command.
  - Entering the SET QUALIFY command.
  - Entering the LOADDEBUGDATA command.

- You can enter the M prefix command by using the Source window prefix area to add the variables on that line to the Monitor window. For the list of supported compile units, see "M prefix (full-screen mode)" on page 173.
- If one or more variables in the Monitor Local List expression is not defined in the specified compile unit, z/OS Debugger displays an error message and does not establish a MONITOR.
- If a duplicate MONITOR command is entered, z/OS Debugger ignores the command and issues a message that a duplicate command has been entered. If the commands are the same when LIST MONITOR is entered, they are considered to be duplicates; likewise, if they are different when LIST MONITOR is entered, they are considered unique.

**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.

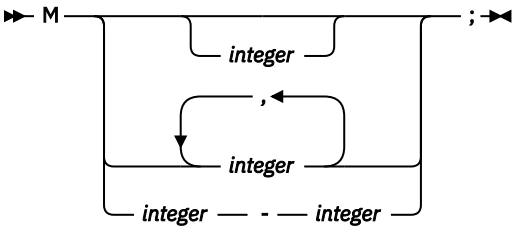
```
MONITOR LOCAL myprog LIST abc;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"cu_spec" on page 13
"CLEAR command" on page 86
"DESCRIBE command" on page 103
"LIST command" on page 140
"M prefix (full-screen mode)" on page 173
"QUERY command" on page 194
"SET MONITOR command" on page 243

# M prefix (full-screen mode)

The M prefix command, which you enter through the prefix area of the Source window, adds an operand or operands on that line to the Monitor window.




***integer***
Identifies specific operands to be monitored. If you do not specify an *integer*, z/OS Debugger monitors all operands. If you enter a single number or the form *1,2,3*, z/OS Debugger monitors the specified operand or operands. If you use the form 1-4, z/OS Debugger monitors operands 1 through 4.

For programs other than assembler and disassembly, *integer* identifies the position of a variable on a line, beginning from the left. The first variable on the line is position 1, the second variable on the line is position 2, and this pattern repeats until there are no more variables. If a variable is on the line more than once, only the first instance of the variable is assigned a position number. If no integer is specified, all the variables on the line are added to the Monitor window.

For assembler and disassembly programs, *integer* identifies operands of the machine instruction. z/OS Debugger numbers them from left to right with the first operand numbered operand 1, the second

operand numbered operand 2, and repeating the pattern until there are no more operands. If you do not specify an *integer*, z/OS Debugger adds all operands referenced explicitly or implicitly by the instruction to the Monitor window. If you specify any form of *integer*, z/OS Debugger adds only the operands explicitly referenced by the specified operand or operands to the Monitor window.

**Usage notes**

- For C/C++, integer values cannot be specified.
- The `M` prefix command can be entered only on lines that have valid executable statements.
- You can enter the `M` prefix command on multiple lines.
- The `M` prefix command works only for the following compile units:
  - Assembler or disassembly compile units
  - Enterprise COBOL compile units
  - Enterprise PL/I compile units compiled with Enterprise PL/I for z/OS, Version 3.6 or 3.7 with the PTF for APAR PK70606 applied, or later
  - C/C++ compile units, compiled with the z/OS 2.1 XL C/C++ compiler or later, with DEBUG(FORMAT(DWARF)) option.
- You cannot use the `M` prefix command on a line that is in a block that is not currently active.
- The following notes apply when you use the `M` prefix command in an assembler or disassembly program:
  - When you specify *integer*, it applies to an entire machine instruction operand, not to a single symbol. For example, in the following instruction, operand 1 is the storage referenced by "SYM1-SYM2(LEN,R8)" and operand 2 is the storage referenced by SOURCE:

    ```
    MVC  SYM1-SYM2(LEN,R8),SOURCE
    ```

  - z/OS Debugger uses the current values in a register to evaluate any registers referenced by an instruction. When you reference an instruction that is not the instruction where the program is suspended, the current values in a register might differ from what the values would be if z/OS Debugger stopped the program at the instruction you referenced.
  - When you specify an explicit base or index register in an operand, z/OS Debugger computes the effective address of the storage location when you enter the `M` prefix command. z/OS Debugger does not recompute the effective address while it monitors the operand.
  - When you specify a single symbol as a machine instruction operand, z/OS Debugger uses the current value of any base register and the currently active USING as z/OS Debugger monitors the operand.
  - The `M` prefix command cannot access mask fields, immediate data fields, and any other constants imbedded in the machine instructions. However, z/OS Debugger does number these fields when it numbers the operands.
  - For instructions that might be coded using extended mnemonics (BC, BCR, and BRC), z/OS Debugger cannot determine whether the base form or the extended mnemonic was used. Therefore, you can use both 1 and 2 to refer to the operand representing the branch target.

**Example**

The following example uses the following lines of code:

```
...
             293     move 0 to c; move 0 to b; move 0 to IND; move  b  to a;
...
             319     if a + b < b + c
             320        then move ind to c;
             321     end-if;
...
```

To add the variable c on line 293 to the Monitor window, enter the `M1` command in the prefix area of line 293.

The next set of examples use the following lines of assembler source code:

```
...
200          L    R6,=X'31BA4038'
201          STM  R1,R4,0(R6)
202          TM   X'01',FLAGS
203
...
```

- Enter `M` on line 201. z/OS Debugger adds the following registers and memory locations to the Monitor window: R1, R2, R3, R4, R6, and the sixteen bytes of storage at location X'31BA4038'.
- Enter `M1-2` on line 201. z/OS Debugger adds R1 and R4 to the Monitor window.
- Enter `M1` on line 202. z/OS Debugger displays an error message because the `M` prefix command cannot access mask and immediate fields.

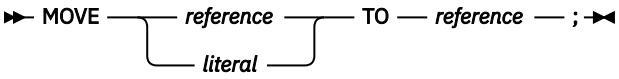
Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

"MONITOR command" on page 171

# MOVE command (COBOL)

The `MOVE` command transfers data from one area of storage to another. The keywords cannot be abbreviated.




***reference***
A valid z/OS Debugger COBOL reference.

***literal***
A valid COBOL literal.

**Usage notes**

- For Enterprise COBOL for z/OS Version 5, you can use the `MOVE` command to update the following special registers:

```
JNIENVPTR
SHIFT-IN
SHIFT-OUT
LINAGE-COUNTER of <FD>
```

- For Enterprise COBOL for z/OS Version 5, you can use the `MOVE` command to update a numerical type with a non-numerical character. For example: "`Move "-999999909" to Znumed`" where Znumed is defined as "`01 Znumed pic -9,999.909`".
- If z/OS Debugger 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 nondate 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.

  Enterprise COBOL for z/OS Version 5 compiler does not support the DATE FORMAT clause. Additional information on the DATE FORMAT clause and other clauses no longer available can be found in *Enterprise COBOL for z/OS Migration Guide* Version 5 Release 1.

- 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.
- If a COBOL variable defined as UTF-8 is used as the receiving field in a MOVE command with an alphabetic or alphanumeric operand, the operand that is not UTF-8 is converted to UTF-8 before that move is done, except for Group items.

  See *Enterprise COBOL for z/OS Language Reference* for more information about using COBOL variables with the MOVE statement.
- Literals with an N or NX prefix are always treated as National data and can be moved only to other National or UTF-8 Data Items or Group items.
- Literals with an U or UX prefix are always treated as UTF-8 data and can be moved only to other UTF-8 or 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 topics for more information related to the material discussed in this topic.

**Related tasks**
*Enterprise COBOL for z/OS Programming Guide*

**Related references**

"Allowable moves for the MOVE command (COBOL)" on page 177
"SET WARNING command (C, C++, COBOL, and PL/I)" on page 263

## Allowable moves for the MOVE command (COBOL)

The following table shows the allowable moves for the z/OS Debugger MOVE command.

| Source field | Receiving field | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **GR** | **AL** | **AN** | **ED** | **BI** | **NE** | **ANE** | **NDI** | **NNDI** | **ID** | **IF** | **EF** | **D1** | **UT** |
| GROUP (GR) | Y | Y | Y | Y[1] | Y[1] | Y[1] | Y[1] | Y[1] | | Y[1] | Y[1] | Y[1] | | Y[1] |
| ALPHABETIC (AL) | Y | Y | | | | | | Y | | | | | | Y |
| ALPHANUMERIC (AN)[4,5] | Y | | Y | | | | | Y | | | | | | Y |
| EXTERNAL DECIMAL (ED)[4,5] | Y[1] | | | Y | | | | | | | | | | |
| BINARY (BI) | Y[1] | | | | Y | | | | | | | | | |
| NUMERIC EDITED (NE) | Y | | | | | | | | | | | | | |
| ALPHANUMERIC EDITED (ANE) | Y | | | | | | Y | Y | | | | | | Y |
| FIGCON ZERO | Y | | Y | Y[2] | Y[2] | | Y | | NU | Y[2] | Y | Y | | |
| FIGCON ZERO, SPACE, or QUOTE | | | | | | | | Y | | | | | | Y |
| SPACES (AL) | Y | Y | Y | | | | Y | | | | | | | |
| HIGH-VALUE, LOW-VALUE, QUOTES | Y | | Y | | | | Y | | | | | | | |
| NATIONAL DATA ITEM (NDI) | Y[1] | | | | | | | Y | | | | | | Y |
| NATIONAL NUMERIC DATA ITEM (NNDI) | | | | | | | | | NN | | | | | |
| NUMERIC LITERAL | Y[1] | | | Y | Y | | | | NN | Y | Y | Y | | |
| ALPHANUMERIC LITERAL | Y | Y | Y | | | Y[1] | Y | Y | | | | | | Y |
| ALPHANUMERIC HEX LITERAL[6] | Y | Y | Y | Y | Y | Y | Y | | | Y | Y | Y | | |
| INTERNAL DECIMAL (ID)[4,5] | Y[1] | | | | | | | | | Y | | | | |
| FLOATING POINT LITERAL | Y[1] | | | | | | | | | | Y | Y | | |
| INTERNAL FLOATING POINT (IF) | Y[1] | | | | | | | | | | Y | Y | | |

| Source field | Receiving field | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | GR | AL | AN | ED | BI | NE | ANE | NDI | NNDI | ID | IF | EF | D1 | UT |
| EXTERNAL FLOATING POINT (EF) | Y[1] | | | | | | | | | | Y | Y[3] | | |
| DBCS DATA ITEM (D1) | | | | | | | | | | | | | Y | |
| DBCS LITERAL | | | | | | | | | | | | | Y | |
| NATIONAL LITERAL (NL) | Y | | | | | | | Y | | | | | | Y |
| NATIONAL HEX LITERAL (NHL)[7] | Y[1] | | | | | | | Y | | | | | | Y |
| UTF-8 (UT) | Y[1] | | | | | | | Y | | | | | | Y |
| UTF-8 LITERAL | Y | | | | | | | Y | | | | | | Y |
| UTF-8 HEX LITERAL[8] | Y[1] | | | | | | | Y | | | | | | Y |

**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 quotation marks (") or apostrophes (') and preceded by X.

**7**
Must be hexadecimal characters only, delimited by either quotation marks (") or apostrophes (') and preceded by NX.

**8**
Must be hexadecimal characters only, delimited by either quotation marks (") or apostrophes (') and preceded by UX.

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
*Enterprise COBOL for z/OS Programming Guide*

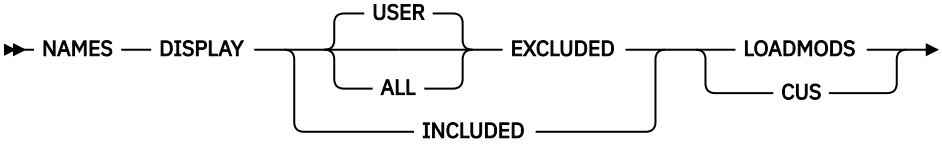
**Related references**
“MOVE command (COBOL)” on page 175

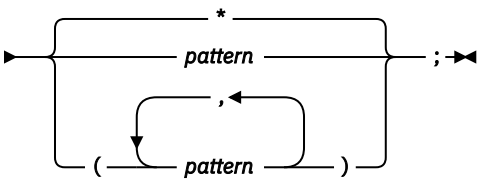
# NAMES command

Use the NAMES command only as instructed in "Debugging user programs that use system prefixed names" in the IBM z/OS Debugger User's Guide.

## NAMES DISPLAY command

Use the NAMES DISPLAY command to indicate that you want a list of all the load modules or compile units that are currently excluded or included. If you do not specify the ALL parameter, only the names excluded by user commands appear in the list that is displayed. Names that z/OS Debugger excludes by default are not included in the list that is displayed.




**USER**
Indicates that you want a list of load modules or compile units that are currently excluded at your request (by using NAMES EXCLUDE command).

**ALL**
Indicates that you want a list of all load modules or compile units that are currently excluded, including those that z/OS Debugger excludes by default.

**LOADMODS**
Indicates that you want a list of load module names.

**CUS**
Indicates that you want a list of compile unit names.

***pattern***
Specifies the name of the load module or compile unit, or a string surrounded by quotation marks (") or apostrophes (') that contains a partial load module or compile unit name followed by an asterisk to indicate that you want a list of all load modules or compile units beginning with the specified string.

**Usage note**

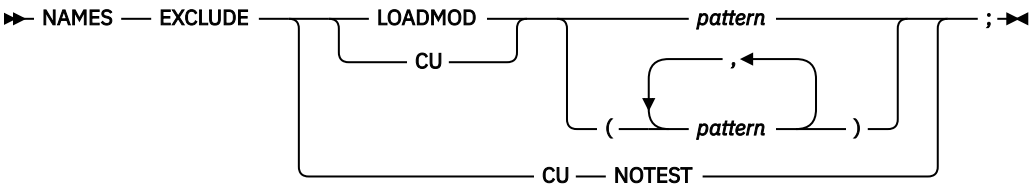
You can use this command in remote debug mode.

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
"Debugging user programs that use system prefixed names" in the IBM z/OS Debugger User's Guide
**Related references**
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

## NAMES EXCLUDE command

The NAMES EXCLUDE command enables you to indicate to z/OS Debugger the names of load modules or compile units that you do not need to debug. If these are data-only modules, z/OS Debugger does not process them. If they contain executable code, z/OS Debugger might process them in some cases. See "Optimizing the debugging of large applications" in the IBM z/OS Debugger User's Guide for more information about these situations.




**LOADMOD**
Indicates that you do not want to debug the specified load module.

**CU**
Indicates that you do not want to debug the specified compile unit.

**NOTEST**
Indicates that you do not want to debug any compile units that were not compiled with debug data.

***pattern***
Specifies the name of the load module or compile unit, or a string surrounded by quotation marks (") or apostrophes (') that contains a partial load module or compile unit name followed by an asterisk to indicate that you do not want to debug all load modules or compile units beginning with the specified string.

**Usage notes**

- You can use this command in remote debug mode.
- You cannot use the NAMES EXCLUDE command on load modules or compile units that are already known to z/OS Debugger.

  If you specify the name of a currently known load module or compile unit, it is added to the exclude list so that if the name becomes unknown, it is excluded in subsequent appearances. However, the currently known load module or compile unit remains known.
- You cannot use the NAMES EXCLUDE command to indicate to z/OS Debugger that you want to exclude the initial load module or the compile units contained in the initial load module. If you want to do this, you must specify the EQAOPTS NAMES command, as described in "Using the EQAOPTS NAMES command to include or exclude the initial load module" in the IBM z/OS Debugger User's Guide.
- For C and C++ programs, the *pattern* parameter is case sensitive. For all other languages, the pattern is not case sensitive.

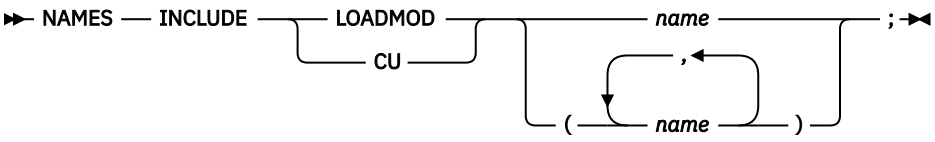
Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
"Debugging user programs that use system prefixed names" in the IBM z/OS Debugger User's Guide
"Debugging programs containing data-only modules" in the IBM z/OS Debugger User's Guide

**Related references**
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

## NAMES INCLUDE command

Use the NAMES INCLUDE command to indicate to z/OS Debugger that your program is a user load module or compile unit, not a system program. See "Debugging user programs that use system prefix names" in the IBM z/OS Debugger User's Guide for more information.




**LOADMOD**
Indicates that you want to debug the specified load module.

**CU**
Indicates that you want to debug the specified compile unit.

***name***
Specifies the name of the load module or compile unit.

**Usage notes**

- You can use this command in remote debug mode.
- You cannot use the NAMES INCLUDE command on load modules or compile units that are already known to z/OS Debugger.
- You cannot use the NAMES INCLUDE command to indicate to z/OS Debugger that you want to debug the initial load module or the compile units contained in the initial load module. If you want to do this, you must specify the EQAOPTS NAMES command, as described in "Using the EQAOPTS NAMES command to include or exclude the initial load module" in the IBM z/OS Debugger User's Guide.
- Do **not** use the NAMES INCLUDE command to debug system components (for example, z/OS Debugger, Language Environment, CICS, IMS, or compiler run-time modules). If you attempt to debug these system components, you might experience unpredictable failures. Only use this command to debug *user* programs that are named with prefixes that z/OS Debugger recognizes as system components.
- z/OS Debugger generates implicit NAMES INCLUDE commands for the following situations:
  - The target of a deferred AT ENTRY command
  - The target of an AT LOAD command
  - The target of a LOADDEBUGDATA command
  - In CICS, the programs specified in DTCN and CADP, unless they contain an asterisk (*)

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
"Debugging user programs that use system prefixed names" in the IBM z/OS Debugger User's Guide

**Related references**
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

## 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.

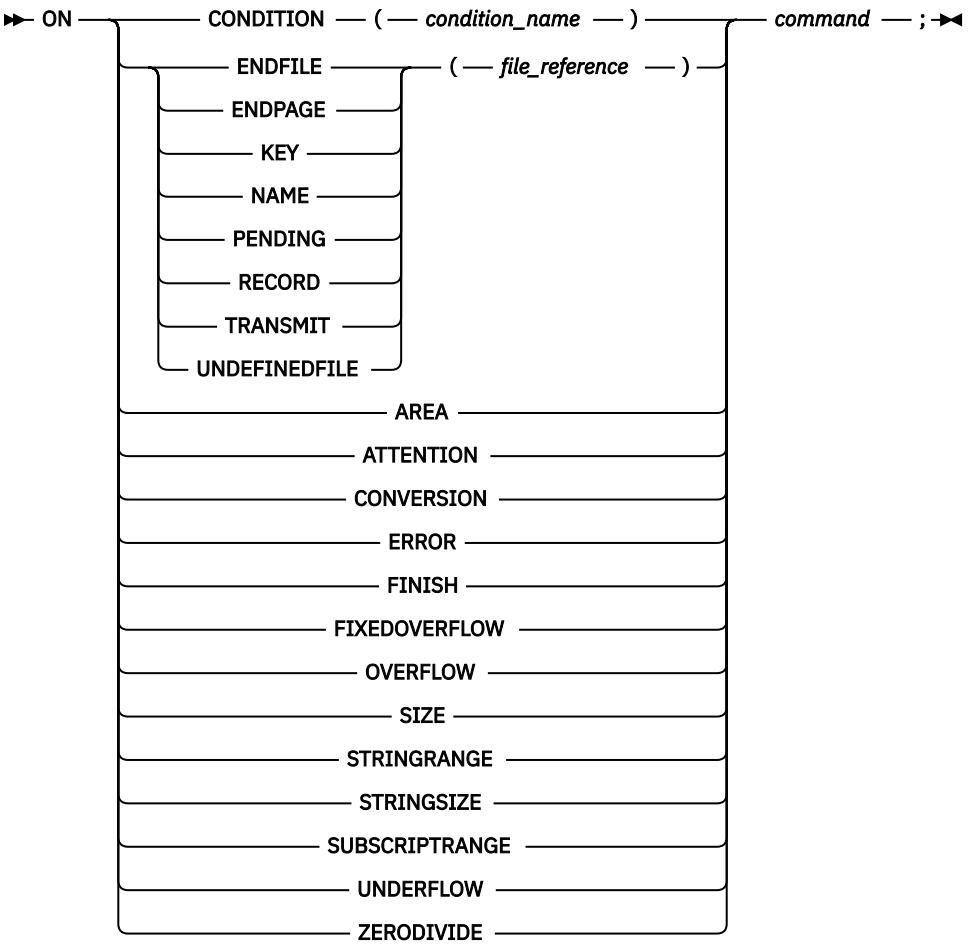
**Example**

Do nothing if array[x] > 0; otherwise, set a to 1. The current programming language setting is C.

```
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.




***condition_name***
A valid PL/I CONDITION condition name.

***file_reference***
A valid PL/I file constant, file variable (can be qualified), or an asterisk (*). If you use an asterisk (*), the breakpoint is activated for all file references associated with the condition used in the ON command.

***command***
A valid z/OS Debugger 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.
- For Enterprise PL/I, you cannot use file variables in the file_reference field.
- The ON command occurs before any existing ON-unit in your application program. The ON-unit is processed after z/OS Debugger 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.

```
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:

```
DECLARE i FIXED BINARY(31,0);
  .
  ..
  ...
i = '1s3';
```

  The following z/OS Debugger command would display and patch the error character when converting the character data to numeric:

```
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.

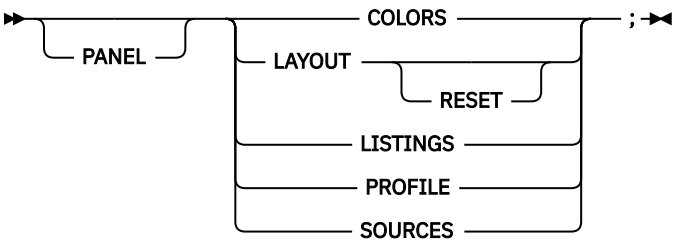
Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"AT OCCURRENCE command" on page 65
*Enterprise PL/I for z/OS Language Reference*

# PANEL command (full-screen mode)

The PANEL command displays special panels. The PANEL keyword is optional.

The PANEL command cannot be used in a command list, any conditional command, or any multiway command.




**COLORS**
Displays the Color Selection panel that allows the selection of color, highlighting, and intensity of the fields of the z/OS Debugger session panel.

**LAYOUT**
Displays the Window Layout Selection panel that controls the configuration of the windows on the z/OS Debugger session panel.

**RESET**
Restores the relative sizes of windows for the current configuration, without displaying the window layout panel. For configurations 1 and 4, 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 you associate compile units with the names of their respective listing, source, or separate debug file. `LISTINGS` is equivalent to `SOURCES`.

z/OS Debugger provides the Source Identification panel to maintain a record of compile units associated with your program, as well as their associated source, listing, or separate debug files.

You can also make source or listings available to z/OS Debugger by entering their names on the Source Identification panel.

The Source Identification panel associates compile units with the names of their respective listing, source, separate debug file 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.

```
                  Source Identification Panel
Command ===>

Compile Unit           Listings/Source File              Display
---------------------- --------------------------------- -------
DBKP515                TS64081.TEST.LISTING(IBME73)         Y
-----------            ---------------------------          -

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 z/OS Debugger. New compile units are added to the list as they become known.

**Listing/Source File**
Is the name of the listing, source, EQALANGX, or separate debug 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.dsname.C(membername)` (C and C++), `userid.dsname.Listing(membername)` (COBOL), or `userid.dsname.List(membername)` (PL/I) for partitioned data sets. For assembler and LangX COBOL the default is `userid.EQALANGX(membername)`.

**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 listing, source, or separate debug file associated with a compile unit by entering the new name over the listing, source, or separate debug 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 z/OS Debugger 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 z/OS Debugger session can be set.

**SOURCES**
Is equivalent to LISTINGS.

**Usage notes**

- For an Enterprise COBOL for z/OS Version 5 program, the contents that are displayed with PANEL SOURCE and PANEL LISTING show the location of the load module.
- All information about 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 z/OS Debugger 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;
```

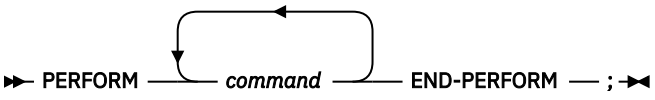
Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
“SET SCREEN command (full-screen mode)” on page 258
"Customizing your full-screen session" in the *IBM z/OS Debugger User's Guide*

# PERFORM command (COBOL)

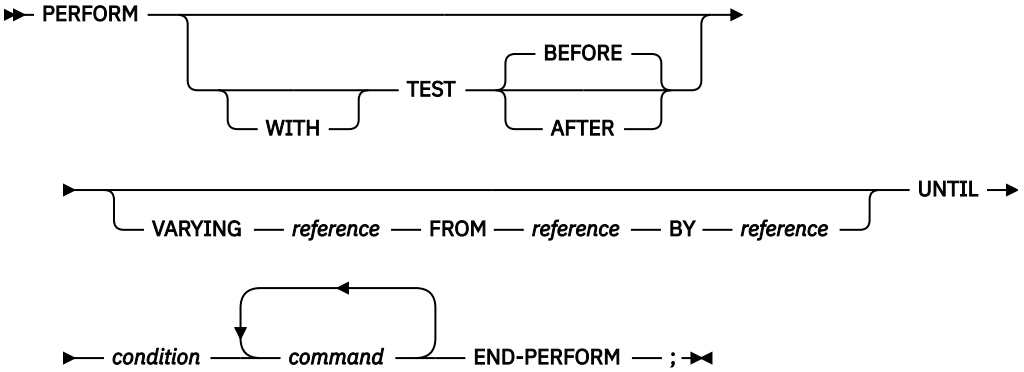
The PERFORM command transfers control explicitly to one or more statements and implicitly returns control to the next executable statement after execution of the specified statements is completed. The keywords cannot be abbreviated.

**Simple:**




***command***
A valid z/OS Debugger command.

**Repeating:**




***reference***
A valid z/OS Debugger COBOL reference.

***condition***
A simple relation condition.

***command***
A valid z/OS Debugger 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 z/OS Debugger 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:

  ```
  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 the 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 topics for more information related to the material discussed in this topic.

**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.

| Command | Description |
|---|---|
| "PLAYBACK ENABLE command" on page 188 | Informs z/OS Debugger to record all subsequent statements that you run and other information about your program. |
| "PLAYBACK START command" on page 189 | Informs z/OS Debugger to suspend normal debugging and to prepare to replay recorded statements. |
| "PLAYBACK FORWARD command" on page 190 | Informs z/OS Debugger to replay recorded statements in forward direction. |
| "PLAYBACK BACKWARD command" on page 190 | Informs z/OS Debugger to replay recorded statements in backward direction. |
| "PLAYBACK STOP command" on page 190 | Informs z/OS Debugger to stop replaying statements, resume normal debugging, and continue recording the statements that you run and other information about your program. |
| "PLAYBACK DISABLE command" on page 191 | Informs z/OS Debugger to stop recording the statements that you run and discard the information about your program that it recorded. |

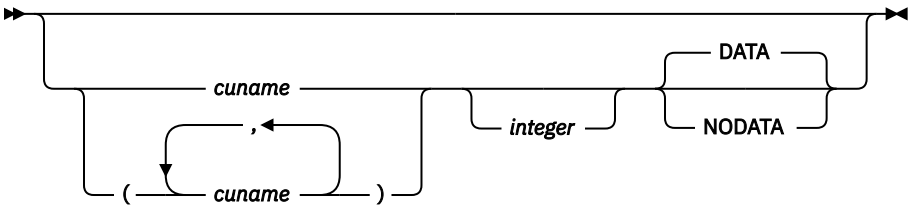
**Usage notes**

- In remote debugging, you can enable playback by using the Playback toolbar in the Debug view. Playback is recording and replaying statements. Changes made to variables are also recorded and available to replay. However, you cannot modify variables and registers during playback.

## PLAYBACK ENABLE command

The PLAYBACK ENABLE command informs z/OS Debugger to begin recording the statements that you run and information about your program. If z/OS Debugger is already recording the statements that you run, you can use the PLAYBACK ENABLE command to inform z/OS Debugger to record the statements that you run in other compile units or to change the effect of the DATA option.


**options**




***cuname***
Name of the compile unit or compile units where z/OS Debugger is to record the statements that you run. You can specify only the names of the compile units currently known.

*****
Specifies that z/OS Debugger 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 z/OS Debugger is to save information about your program, such as the value of variables and registers. z/OS Debugger 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 following compilers:

- Enterprise COBOL for z/OS, Version 6
- Enterprise COBOL for z/OS, Version 5
- Enterprise COBOL for z/OS, Version 4
- With the following compilers, you must also specify the SYM suboption of the TEST compiler option:
  - Enterprise COBOL for z/OS, Version 3.3 and Version 3.4
  - 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

DATA is the default.

**NODATA**
Specifies that z/OS Debugger does not save information about your program.

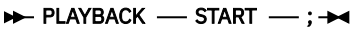
**Usage notes**

- **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 for: ADDRESS OF, LENGTH OF, and WHEN-COMPILED

## PLAYBACK START command

The PLAYBACK START command suspends normal debugging and informs z/OS Debugger 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.




### Usage notes

The following commands are available while you replay recorded statements:

| | | |
|---|---|---|
| "ALLOCATE command" on page 31 | "FIND command" on page 117 | "QQUIT command" on page 200 |
| "CALL procedure command" on page 84 | "FREE command" on page 124 | "RETRIEVE command (full-screen mode)" on page 202 |
| CLEAR EQUATE | "IMMEDIATE command (full-screen mode)" on page 135 | "RUNTO command" on page 203 |
| CLEAR LOG | null | "SCROLL command (full-screen mode)" on page 204 |
| CLEAR MONITOR | "PANEL command (full-screen mode)" on page 183 | SET (most forms) |
| CLEAR PROCEDURE | "PERFORM command (COBOL)" on page 1851 | "STEP command" on page 269 |
| "COMMENT command" on page 93 | "PLAYBACK commands" on page 187 | "SYSTEM command (z/OS)" on page 275 |
| "CURSOR command (full-screen mode)" on page 95 | "Prefix commands (full-screen mode)" on page 192 | "TSO command (z/OS)" on page 280 |
| "Declarations (COBOL)" on page 99 | "PROCEDURE command" on page 193 | "USE command" on page 280 |
| DESCRIBE CUS | "QUERY command" on page 194 | "WINDOW command (full-screen mode)" on page 282 |
| DESCRIBE PROGRAMS | "QUIT command" on page 199 | |

1Refer to "PERFORM command (COBOL)" on page 185 for restrictions.

If the DATA option is in effect and the compile unit supports the DATA option, the following commands are available:

| | |
|---|---|
| "COMPUTE command (COBOL)" on page 94 | LIST |
| DESCRIBE ATTRIBUTES | "MOVE command (COBOL)" on page 175 2 |
| DESCRIBE CURSOR | MONITOR |
| "EVALUATE command (COBOL)" on page 115 | "SET command (COBOL)" on page 266 2 |

| | |
|---|---|
| “IF command (COBOL)” on page 131 | “SET AUTOMONITOR command” on page 212 |

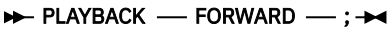
2 The target must be session variable.

The following commands are not available while you replay recorded statements:

| | | |
|---|---|---|
| “ANALYZE command (PL/I)” on page 32 | “Declarations (C and C++)” on page 96 | “if command (C and C++)” on page 130 |
| “Assignment command (assembler and disassembly)” on page 33 | “DECLARE command (PL/I)” on page 101 | “IF command (PL/I)” on page 134 |
| “Assignment command (PL/I)” on page 36 | DESCRIBE ENVIRONMENT | “INPUT command (C, C++, and COBOL)” on page 135 |
| “AT command” on page 37 | “DISABLE command” on page 107 | “ON command (PL/I)” on page 181 |
| “break command (C and C++)” on page 75 | “do/while command (C and C++)” on page 110 | “RUN command” on page 203 |
| “CALL %DUMP command” on page 77 | “DO command (PL/I)” on page 111 | “SELECT command (PL/I)” on page 207 |
| “CALL entry_name command (COBOL)” on page 83 | “ENABLE command” on page 113 | SET INTERCEPT |
| CLEAR AT | “Expression command (C and C++)” on page 116 | “switch command (C and C++)” on page 273 |
| CLEAR DECLARE | “for command (C and C++)” on page 123 | “TRIGGER command” on page 276 |
| CLEAR ON | “GO command” on page 124 | “while command (C and C++)” on page 281 |
| CLEAR VARIABLES | “GOTO command” on page 125 | |

## PLAYBACK FORWARD command

The PLAYBACK FORWARD command informs z/OS Debugger to perform STEP and RUNTO commands forward, starting from the current statement and going to the next statement.

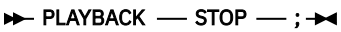



## PLAYBACK BACKWARD command

The PLAYBACK BACKWARD command informs z/OS Debugger 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.

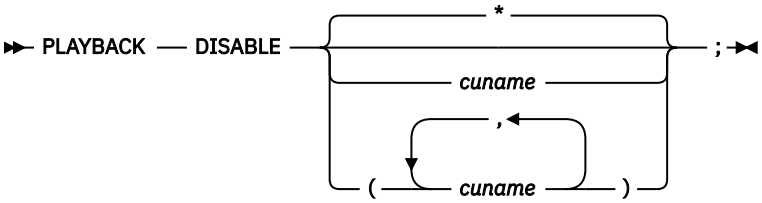

## PLAYBACK STOP command

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. z/OS Debugger continues to record the statements you run and, if you specified the DATA option, information about your program.




## PLAYBACK DISABLE command

The PLAYBACK DISABLE command informs z/OS Debugger 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 z/OS Debugger collected while recording is discarded. You can instruct z/OS Debugger 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.

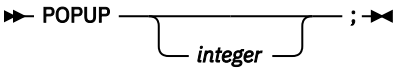



***cuname***
Indicates to z/OS Debugger to stop recording for the compile unit or compile units specified. Only the names of currently known compile units can be specified.

***
Indicates to z/OS Debugger to stop recording for all compile units. This is the default.

# POPUP command

Displays the Command pop-up window, where you can type in multiline commands.




***integer***
The number of lines for the window.

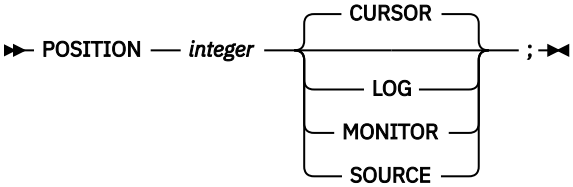
If you do not specify an integer, z/OS Debugger opens the window with the number of lines specified by the SET POPUP command.

**Related references**

"SET POPUP command" on page 248

# POSITION command

Positions the cursor to a specific line in the specified window. This command does not work in the disassembly view.




***integer***
Specifies that z/OS Debugger scroll the specified window to line number *integer*. z/OS Debugger matches *integer* to the line number in the prefix area of the specified window. z/OS Debugger can scroll either up or down. The maximum value you can specify is 999999.

# 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, including the automonitor section. 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 8. Source window prefix commands*

| Command | Description |
|---|---|
| "AT Prefix command (full-screen mode)" on page 70 | Defines a statement breakpoint through the Source window prefix area. |
| "CLEAR prefix (full-screen mode)" on page 92 | Clears a breakpoint through the Source window prefix area. |
| "DISABLE prefix (full-screen mode)" on page 109 | Disables a breakpoint through the Source window prefix area. |
| "ENABLE prefix (full-screen mode)" on page 114 | Enables a disabled breakpoint through the Source window prefix area. |
| "L prefix command (full-screen mode)" on page 153 | Displays the values of the variables on that line. |
| "M prefix (full-screen mode)" on page 173 | Adds the variables on that line to the Monitor window. |
| "QUERY prefix (full-screen mode)" on page 199 | Queries what statements have breakpoints through the Source window prefix area. |
| "RUNTO prefix command (full-screen mode)" on page 204 | Runs the program to the location that the cursor or statement identifier indicate in the Source window prefix area. |
| "SHOW prefix command (full-screen mode)" on page 269 | Specifies what relative statement or verb within the line is to have its frequency count shown in the suffix area. |

*Table 9. Monitor window prefix commands*

| Command | Description |
|---|---|
| CC...CC "CLEAR command" on page 86 | Clears selected block of multiple items of the current set from the MONITOR window. |
| CL (CLEAR MONITOR *n*) "CLEAR command" on page 86 | Clears selected member of the current set of MONITOR commands. |
| DEF (MONITOR *n* DEFAULT)"MONITOR command" on page 171 | Displays selected member of the current set of MONITOR commands in default representation. |

*Table 9. Monitor window prefix commands (continued)*

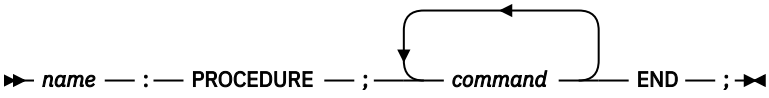
| Command | Description |
|---|---|
| HEX (MONITOR n HEX) "MONITOR command" on page 171 | Displays selected member of the current set of MONITOR commands in hexadecimal representation. |
| LIST (LIST MONITOR *n*) "LIST MONITOR command" on page 157 | Lists selected member of the current set of MONITOR commands. |

# 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 z/OS Debugger procedure name. It must be a valid identifier in the current programming language. The maximum length is 31 characters.

***command***
A valid z/OS Debugger command other than a declaration or PANEL command.

**Usage notes**

- Because the z/OS Debugger 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 z/OS Debugger, 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 z/OS Debugger 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;
```

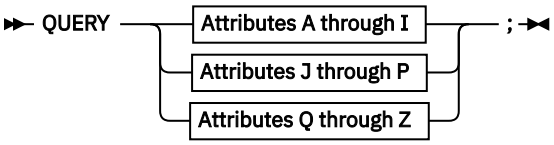
# QUALIFY RESET command

The QUALIFY RESET command is equivalent to the SET QUALIFY RESET command.

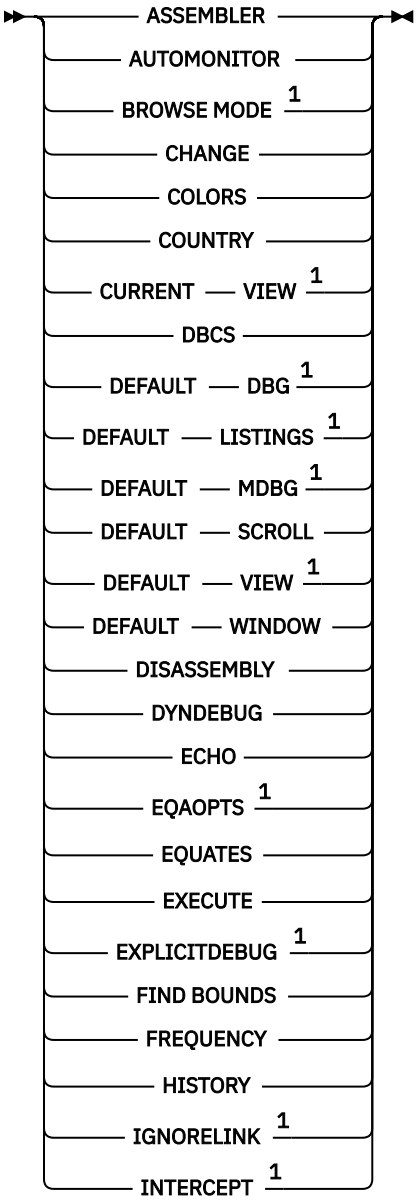
# QUERY command

The QUERY command displays the current value of the specified z/OS Debugger setting, the current setting of all the z/OS Debugger settings, or the current location in the suspended program.

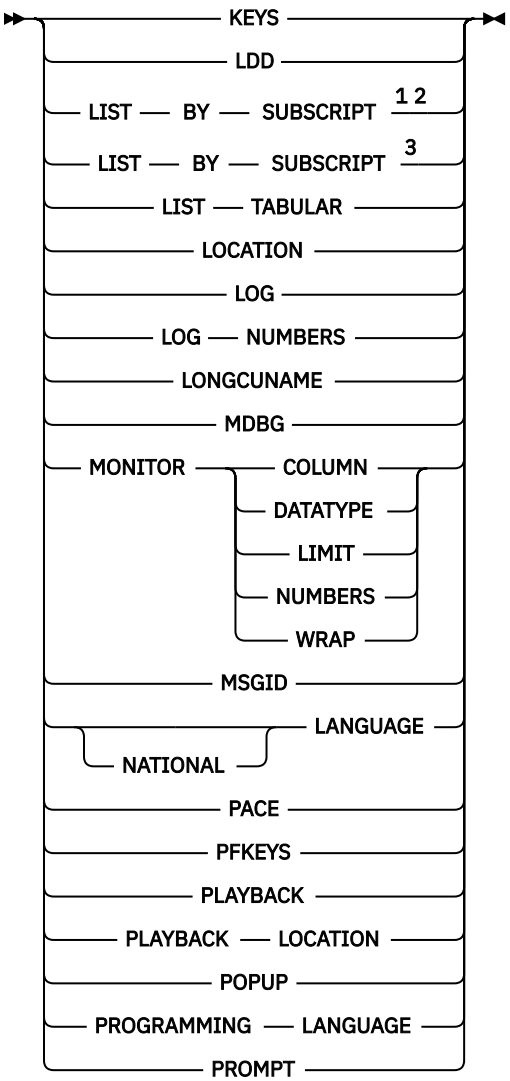
For an explanation of the z/OS Debugger settings, see the SET command.



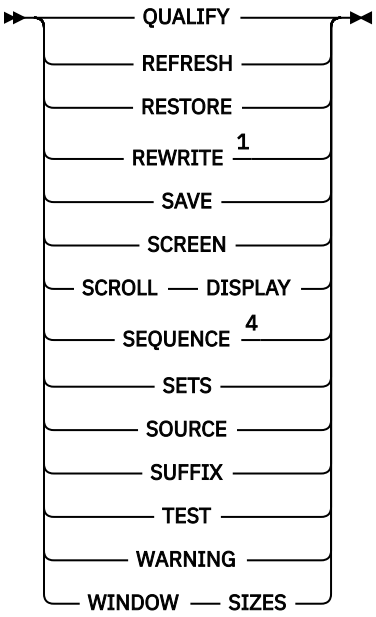

**Attributes A through I**




**Attributes J through P**




**Attributes Q through Z**




Notes:

[1] You can use this command in remote debug mode.
[2] Only for COBOL.
[3] Only for Enterprise PL/I.
[4] Only for PL/I.

**ASSEMBLER**
Displays the current ASSEMBLER setting.

**AUTOMONITOR**
Displays the current AUTOMONITOR setting.

**BROWSE MODE**
Displays the current browse mode 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 DBG**
Displays the current DEFAULT DBG setting.

**DEFAULT LISTINGS**
Displays the current DEFAULT LISTINGS setting.

**DEFAULT MDBG**
Displays the current DEFAULT MDBG 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.

**DYNDEBUG**
Displays the current DYNDEBUG setting.

**ECHO**
Displays the current ECHO setting.

**EQAOPTS**
Displays the EQAOPTS commands in effect and any errors detected while processing EQAOPTS commands.

**EQUATES**
Displays the current EQUATE definitions.

**EXECUTE**
Displays the current EXECUTE setting.

**EXPLICITDEBUG**
Displays whether explicit debug mode is active.

**FIND BOUNDS**
Displays the current FIND BOUNDS setting.

**FREQUENCY**
Displays the current FREQUENCY setting.

**HISTORY**
Displays the current HISTORY setting and size.

**IGNORELINK**
Displays the current IGNORELINK setting.

**INTERCEPT**
Displays the current INTERCEPT setting.

**KEYS (full-screen mode)**
Displays the current KEYS setting.

**LDD**
Displays the current LDD setting.

**LIST BY SUBSCRIPT**
Displays the current LIST BY SUBSCRIPT 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.

For an AT CHANGE breakpoint, if you set the breakpoint by providing a reference, z/OS Debugger displays the reference. If the reference is a Level 88 variable, z/OS Debugger displays the current setting of true or false.

For an AT CHANGE breakpoint, z/OS Debugger displays the old and new values in hexadecimal format.

**LOG**
Displays the current LOG setting.

**LOG NUMBERS (full-screen mode)**
Displays the current LOG NUMBERS setting.

**LONGCUNAME**
Displays the current LONGCUNAME setting.

**MDBG**
Displays the current MDBG 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 LIMIT (full-screen mode)**
Displays the current MONITOR LIMIT 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.

**POPUP**
Displays the current POPUP setting.

**PROGRAMMING LANGUAGE**
Displays the current PROGRAMMING LANGUAGE setting. z/OS Debugger 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.

**Usage note**

- For Enterprise COBOL for z/OS Version 5, the output for QUERY SOURCE is the location of the load module.
- You can use the QUERY ASSEMBLER, QUERY AUTOMONITOR, QUERY CURRENT VIEW, QUERY DEFAULT LISTINGS, QUERY DEFAULT VIEW, QUERY DISASSEMBLY, QUERY DYNDEBUG, QUERY EQAOPTS, QUERY EXPLICITDEBUG, QUERY IGNORELINK, QUERY INTERCEPT, QUERY LDD, QUERY LOCATION, QUERY LOG, QUERY QUALIFY, QUERY REWRITE, and QUERY WARNING commands in remote debug mode.

**Examples**

- Display the current ECHO setting.

```
QUERY ECHO;
```

- Display all current settings.

```
QUERY SETS;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"QUERY prefix (full-screen mode)" on page 199
Appendix A, "z/OS Debugger commands supported in Debug Tool compatibility mode," on page 517

## 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 "+".

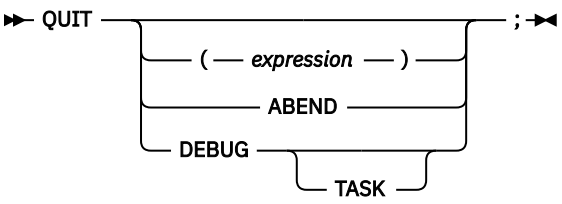
  For example, a display of "..*." indicates that four statements or verbs begin on the line and the third one has a breakpoint defined.
- The QUERY `prefix` command is not logged.

Refer to the following topics for more information related to the material discussed in this topic.

- **Related references**
- "LIST command" on page 140

# QUIT command

The QUIT command ends a z/OS Debugger session and, if an expression is specified, sets the return code. In full-screen mode, it also displays a prompt panel that asks if you really want to quit the debug session. In line, batch, and remote debug mode, the QUIT command ends the session without prompting.




***expression***
A valid z/OS Debugger 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.

You cannot use *expression* in remote debug mode.

**ABEND**
If you specify ABEND, z/OS Debugger raises a CEE2F1 exception to terminate each active enclave.

**DEBUG**
If you specify DEBUG, z/OS Debugger ends and your program keeps running. Any calls to restart z/OS Debugger 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, z/OS Debugger 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, z/OS Debugger debugging will resume.

**Usage notes**

- z/OS Debugger 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 z/OS Debugger commands file, no prompt is displayed. This behavior applies to the z/OS Debugger 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 z/OS Debugger, you must first restart your program.
- If you enter the QUIT or QQUIT command while you are debugging a non-Language Environment assembler or LangX COBOL program running under CICS, z/OS Debugger behaves the same as if you entered a QUIT ABEND command and a U4038 abend occurs.
- In remote debug mode, if any form of the QUIT command is found in a preferences or commands file, the remote debugger displays the message "Connection with debug engine was lost."

**Examples**

- End a z/OS Debugger session.

```
QUIT;
```

- End a z/OS Debugger session and use the value in variable x as the application return code.

```
QUIT (x);
```

- End a z/OS Debugger session without ending the program.

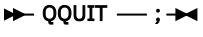
```
QUIT DEBUG;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“expression” on page 14
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

# QQUIT command

The QQUIT command ends a z/OS Debugger session without further prompting.




**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 LangX COBOL program running under CICS, z/OS Debugger behaves the same as if you had entered the QUIT ABEND command and a U4038 abend occurs.
- In remote debug mode, if any form of the QQUIT command is found in a preferences or commands file, the remote debugger displays the message "Connection with debug engine was lost."

**Example**

End a z/OS Debugger session.

```
QQUIT;
```

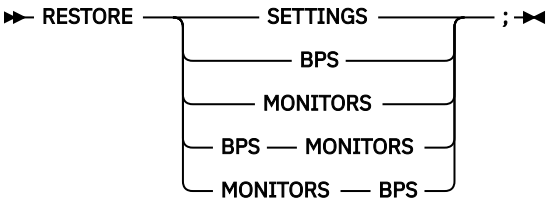
Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

"QUIT command" on page 199

# 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 z/OS Debugger 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 precede the RESTORE command with any other z/OS Debugger command except SET SAVE or another RESTORE command.

**Example**

- Restore the settings:

```
RESTORE SETTINGS;
```

- Restore the breakpoints and monitor specifications:

```
RESTORE BPS MONITORS;
```

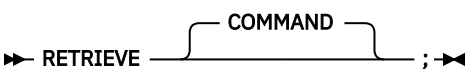
Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
*IBM z/OS Debugger User's Guide*

**Related references**
"SET RESTORE command" on page 253
"SET SAVE command" on page 255

## 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.




**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 topics for more information related to the material discussed in this topic.

**Related references**
"GO command" on page 124

## 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.


**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.
- You can use the RUNTO command in remote debug mode by entering it in the Debug Console or the **Action** field, which is in the **Optional Parameters** section of the **Add a Breakpoint** task.

### Examples

- Run to statement 67, where statement 67 is in a currently active block.

```
RUNTO 67;
```

- Run to the statement 11 in the block IPLI11A, where IPLI11A is known in the current enclave.

```
RUNTO IPLI11A :> 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 topics for more information related to the material discussed in this topic.

**Related references**
"RUN command" on page 203

## 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 RUNT067.
- 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**

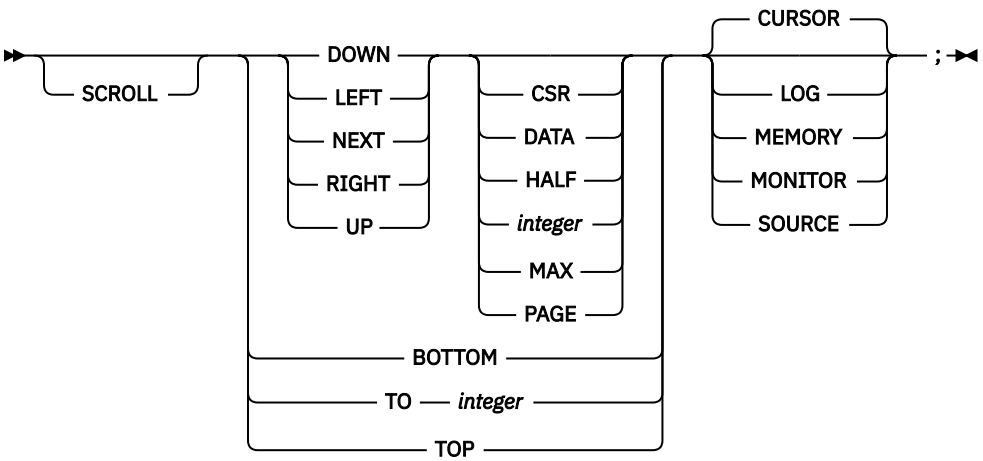
Run to the statement 67, where statement 67 is located in the Source window.

- 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, Memory, 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.

**MEMORY**
Selects the Memory window.

**MONITOR**
Selects the monitor window.

**SOURCE**
Selects the source listing window.

**Usage notes**

- You cannot use the following commands in the Memory window:
  - SCROLL TOP
  - SCROLL BOTTOM
  - SCROLL TO
  - SCROLL LEFT
  - SCROLL RIGHT
  - SCROLL MAX
- 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 z/OS Debugger 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 z/OS Debugger 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.
- When the List pop-up window displays the result of a LIST *expression* command and you enter a SCROLL DOWN or SCROLL UP command without specifying a window (LOG, MEMORY, MONITOR, or SOURCE), z/OS Debugger applies the command to the List pop-up window. The scrolling amount is always PAGE, regardless of the SET DEFAULT SCROLL setting.

**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.

- Scroll the Source window to a line breakpoint.

```
LIST AT STATEMENT;
The STATEMENT COB019 ::> COB01A9 :> 56.1 breakpoint action is:
;
SCROLL TO 56;
```

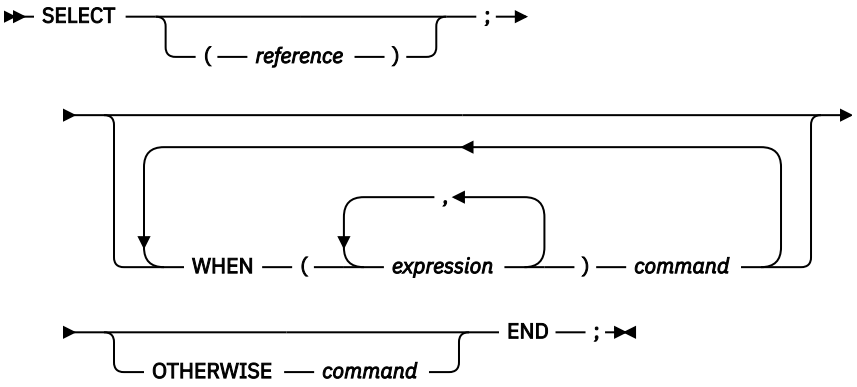
Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“SET DEFAULT SCROLL command (full-screen mode)” on page 222

# 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 z/OS Debugger message is issued.




***reference***
A valid z/OS Debugger PL/I scalar reference. An aggregate (array or structure) cannot be used as a reference.

**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 *reference* is omitted).

***expression***
A valid z/OS Debugger PL/I expression.

***command***
A valid z/OS Debugger command.

**OTHERWISE**
Specifies the command to be executed when every test of the preceding WHEN statements fails.

**Usage notes**

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

**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 z/OS Debugger error message is issued.

```
SELECT (sum);
  WHEN (c + ev) LIST ('Match on when group number 1');
  WHEN (fv, 0) LIST ('Match on when group number 2');
END;
```

# SET command

The SET command sets various switches that affect the operation of z/OS Debugger. Except where otherwise specified, settings remain in effect for the entire debug session.

The following table summarizes the forms of the SET command.

| Command | Description |
|---|---|
| "SET ASSEMBLER ON/OFF command" on page 210 | Controls the enablement of assembler debugging. |
| "SET ASSEMBLER STEPOVER command" on page 211 | Controls the behavior of the STEP OVER command while debugging assembler compile units. |
| "SET AUTOMONITOR command" on page 212 | Controls the addition of data items to the Monitor window. |
| "SET CHANGE command" on page 214 | Controls the frequency of checking the AT CHANGE breakpoints. |
| "SET COLOR command (full-screen and line mode)" on page 215 | Provides control of the color, highlighting, and intensity attributes. |
| "SET COUNTRY command" on page 218 | Changes the current national country setting. |
| "SET DBCS command" on page 218 | Controls whether DBCS shift-in and shift-out codes are recognized. |
| "SET DEFAULT DBG command" on page 219 | Defines a default partitioned data set DD name or DS name that z/OS Debugger searches through to locate the .dbg files. |
| "SET DEFAULT LISTINGS command" on page 220 | Defines a default partitioned data set (PDS) ddname or dsname searched for program source listings or source files. |
| "SET DEFAULT MDBG command" on page 221 | Defines a default partitioned data set DD name or DS name that z/OS Debugger searches through to locate the .mdbg files. |
| "SET DEFAULT SCROLL command (full-screen mode)" on page 222 | Sets the default scroll amount. |
| "SET DEFAULT VIEW command" on page 223 | Controls the default view for assembler compile units. |
| "SET DEFAULT WINDOW command (full-screen mode)" on page 224 | Sets the window that is affected by a window referencing command. |
| "SET DISASSEMBLY command" on page 224 | Controls whether the disassembly view is displayed in the Source window. |
| "SET DYNDEBUG command" on page 225 | Controls whether the Dynamic Debug facility is activated. |
| "SET ECHO command" on page 226 | Controls whether GO and STEP commands are recorded in the log window. |
| "SET EQUATE command" on page 227 | Equates a symbol to a string of characters. |
| "SET EXECUTE command" on page 228 | Controls whether commands are performed or just syntax checked. |
| "SET EXPLICITDEBUG command" on page 228 | Controls whether explicit debug mode is active. |
| "SET FIND BOUNDS command" on page 229 | Controls the columns searched in the Source window and in line mode. |

| Command | Description |
|---|---|
| "SET FREQUENCY command" on page 230 | Controls whether statement executions are counted. |
| "SET HISTORY command" on page 231 | Specifies whether entries to z/OS Debugger are recorded in the history table. |
| "SET IGNORELINK command" on page 231 | Specifies whether to ignore any new LINK level (nested enclave). |
| "SET INTERCEPT command (C and C++)" on page 232 | Intercepts input to and output from specified files. Output and prompts for input are displayed in the log. |
| "SET INTERCEPT command (COBOL, full-screen mode, line mode, batch mode)" on page 233 | Intercepts input to and output from the CONSOLE. Output and prompts for input are displayed in the log. |
| "SET INTERCEPT command (COBOL, remote debug mode)" on page 234 | Intercepts output from COBOL DISPLAY statements. Output is displayed in the Debug Console. |
| "SET KEYS command (full-screen mode)" on page 234 | Controls whether PF key definitions are displayed. |
| "SET LDD command" on page 235 | Controls how debug data is loaded for assemblies containing multiple CSECTs. |
| "SET LIST BY SUBSCRIPT command (COBOL)" on page 236 | Controls whether z/OS Debugger displays elements in an array as they are stored in memory. |
| "SET LIST BY SUBSCRIPT command (Enterprise PL/I, full-screen mode only)" on page 238 | Controls whether z/OS Debugger displays elements in an array as they are stored in memory. |
| "SET LIST TABULAR command" on page 239 | Controls the formatting of the output of the LIST command. |
| "SET LOG command" on page 239 | Controls the logging of output and assignment to the log file. |
| "SET LOG NUMBERS command (full-screen mode)" on page 241 | Controls whether line numbers are shown in the log window. |
| "SET LONGCUNAME command" on page 241 | Controls whether a long or a short CU name is shown. |
| "SET MDBG command" on page 242 | Associates a .mdbg files to one load module or DLL. |
| "SET MONITOR command" on page 243 | Controls the format and layout of variable names and values displayed in the Monitor window. |
| "SET MSGID command" on page 245 | Controls whether message identifiers are shown. |
| "SET NATIONAL LANGUAGE command" on page 245 | Switches your application to a different run-time national language. |
| "SET PACE command" on page 246 | Specifies the maximum pace of animated execution. |
| "SET PFKEY command" on page 247 | Associates a z/OS Debugger command with a PF key. |
| "SET POPUP command" on page 248 | Controls the number of lines displayed in the Command pop-up window. |

| Command | Description |
|---|---|
| "SET PROGRAMMING LANGUAGE command" on page 248 | Sets the current programming language. |
| "SET PROMPT command (full-screen mode)" on page 249 | Controls the display of the current program location. |
| "SET QUALIFY command" on page 250 | Simplifies the identification of references and statement numbers by resetting the point of view. |
| "SET REFRESH command (full-screen mode)" on page 252 | Controls screen refreshing when the SCREEN setting is ON. |
| "SET RESTORE command" on page 253 | Controls the automatic and manual restoring of settings, breakpoints, and monitor specifications. |
| "SET REWRITE command (full-screen mode)" on page 254 | Forces a periodic screen rewrite. |
| "SET SAVE command" on page 255 | Controls the automatic saving of settings, breakpoints, and monitor specifications. |
| "SET SCREEN command (full-screen mode)" on page 258 | Controls how information is displayed on the screen. |
| "SET SCROLL DISPLAY command (full-screen mode)" on page 259 | Controls whether the scroll field is displayed. |
| "SET SEQUENCE command (PL/I)" on page 259 | Controls whether z/OS Debugger interprets data after column 72 as a sequence number. |
| "SET SOURCE command" on page 259 | Associates a source listing or source file with one or more compile units. |
| "SET SUFFIX command (full-screen mode)" on page 261 | Controls the display of the Source window suffix area. |
| "SET TEST command" on page 262 | Overrides the initial TEST run-time options specified at invocation. |
| "SET WARNING command (C, C++, COBOL, and PL/I)" on page 263 | Controls display of the z/OS Debugger warning messages and whether exceptions are reflected to the application program. |

Refer to the following topics for more information related to the material discussed in this topic.

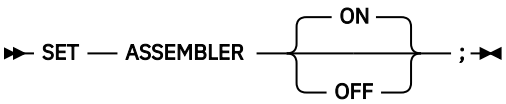
**Related references**
"SET command (COBOL)" on page 266

## SET ASSEMBLER ON/OFF command

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*
- You can display the names of disassembled CUs by using the following commands:
  - DESCRIBE CUS

- LIST
- LIST NAMES CUS
- QUERY SOURCE




**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.

**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 topics for more information related to the material discussed in this topic.

**Related references**
"SET DISASSEMBLY command" on page 224
Appendix A, "z/OS Debugger commands supported in Debug Tool compatibility mode," on page 517

## SET ASSEMBLER STEPOVER command

Specifies how z/OS Debugger processes STEP OVER commands in assembler compile units. When EXTONLY is in effect, z/OS Debugger only steps over calls to external subroutines. When EXTINT is in effect, z/OS Debugger steps over calls to external and internal subroutines. External subroutines are subroutines that are outside the current compile unit; internal subroutines are subroutines that are inside the current compile unit.

z/OS Debugger returns control to you the next time it runs any instruction in the current compile unit (CSECT) when either of the following situations occur:

- EXTONLY is in effect
- EXTINT is in effect and the assembler program calls an external subroutine

z/OS Debugger assumes that the subroutine you want to step over returns to the instruction that follows the call to that subroutine when all of the following situations occur:

- EXTINT is in effect
- The function is an internal subroutine
- The address that immediately follows the instruction where you are currently stopped contains an executable instruction (not data)

z/OS Debugger assumes that you use one of the following instructions to call internal subroutines:

- BAL
- BAS

- BRAS
- BALR
- BASR
- BASSM
- BRASL


**EXTONLY**
Specifies that z/OS Debugger steps *over* external subroutines and steps *through* internal subroutines.

**EXTINT**
Specifies that z/OS Debugger steps over external and internal subroutines.

**Usage notes**

- If EXTINT is in effect and an internal subroutine does not return to the instruction that immediately follows the call to that subroutine, one of the following situations might occur:
  - z/OS Debugger might not regain control
  - z/OS Debugger might regain control only when another breakpoint is run
  - z/OS Debugger might regain control only when an external event occurs
  - z/OS Debugger might not regain control and the program runs until it terminates
- You can use this command in remote debug mode.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"SET ASSEMBLER ON/OFF command" on page 210
"STEP command" on page 269
Appendix A, "z/OS Debugger commands supported in Debug Tool compatibility mode," on page 517

## SET AUTOMONITOR command

Controls the monitoring of data items for the statement that z/OS Debugger will run next, the most recent statement that z/OS Debugger ran, or both. The initial setting is OFF.

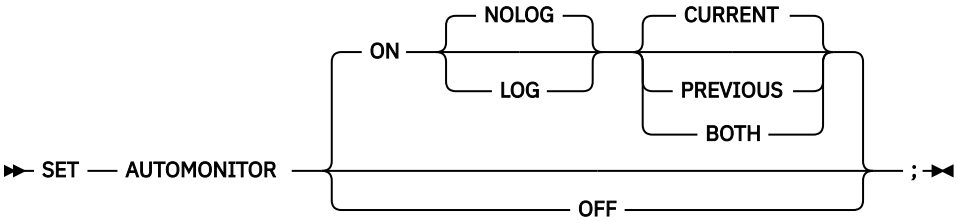
AUTOMONITOR works only for the following compile units:

- COBOL or PL/I compile units compiled with the SYM suboption of the TEST compiler option. COBOL programs compiled with Enterprise COBOL for z/OS, Version 4.1 or later, or PL/I programs compiled with Enterprise PL/I Version 4.4 or later do not need the SYM suboption of the TEST compiler option.
- assembler, disassembly, or LangX COBOL compile units
- C/C++ compile units compiled with the z/OS 2.1 XL C/C++ compiler or later, with the DEBUG(FORMAT(DWARF)) compiler option.

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, Version 4
- Enterprise COBOL for z/OS and OS/390, Version 3 Release 2 or later
- 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 or later
- High Level Assembler for MVS & VM & VSE, Version 1 Release 4 or later
- Any compiler used to generate an EQALANGX file for LangX COBOL
- z/OS 2.1 XL C/C++ compiler or later




**ON**
Enables monitoring of data items for the statement that z/OS Debugger will run next, the most recent statement that z/OS Debugger ran, or both. 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.

**CURRENT**
Monitor data items on the statement that z/OS Debugger will run next. This is the default.

**PREVIOUS**
Monitor data items on the most recent statement that z/OS Debugger ran.

**BOTH**
Monitor data items for the statement that z/OS Debugger will run next and the most recent statement that z/OS Debugger ran.

**Usage notes**

- For Enterprise COBOL for z/OS Version 5, if a statement uses `LENGTH OF <reference>`, it is the length of the reference that is shown in the automonitor output, not the value of `<reference>`.
- You can use this command in remote debug mode.
- 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 with the `PLAYBACK` commands. However, you cannot use the `BOTH` or `PREVIOUS` parameters.
- If you enter the `SET AUTOMONITOR ON LOG` command for a compile unit that was compiled with a compiler that does not support automonitoring, then z/OS Debugger 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 automonitoring, 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, z/OS Debugger displays only 32-bit general registers, floating-point registers, and storage operands. z/OS Debugger displays them in the following manner:
  - Register operands are displayed in numeric order.
  - Storage operands are displayed in the order S1, S2, and S4.
  - When the storage operand is a single symbol, the symbol name is displayed in the automonitor section of the Monitor window. Otherwise, the specified operand is displayed as a comment and the _STORAGE function is used to display the storage contents. For example, `_STORAGE(X'1F3C8'::4))` is used to display a four-byte storage operand at address X'1F3C8'.
- In an assembler compile unit, the `SET AUTOMONITOR` command provides information about a single machine instruction only. Even in the NOMACGEN view, `SET AUTOMONITOR` provides information about only one machine instruction and not all operands of the current macro invocation.
- For LangX COBOL, array references are not included in the AUTOMONITOR output. In addition, when a statement is continued to subsequent lines, operands coded on continuation lines will not be displayed for VS COBOL II and Enterprise COBOL.
- To disable monitoring of all data items, you can enter the `SET AUTOMONITOR OFF` or `CLEAR MONITOR` *n* commands, where *n* is the monitor number of an automonitor entry. You can also use `CL` prefix command on an entry in Monitor window.
- In the AUTOMONITOR section of the Monitor window, z/OS Debugger displays the values of the variables in their declared data type. You can change this behavior in the following ways:
  - To display the value in hexadecimal format for one time, enter the HEX command in the prefix area of that item. When you step through your program, z/OS Debugger reverts the display to the declared data type.
  - To continuously display the value in hexadecimal format, enter one of the following commands:
    - `MONITOR HEX` *n*, where *n* is the monitor number of an entry in the AUTOMONITOR section. When you step through your program, z/OS Debugger displays the value of the variable in hexadecimal format until you enter the `MONITOR DEF` *n* command, where *n* is the same number you used for the `MONITOR HEX` command.
    - HEX in the prefix area of the AUTOMONITOR line ("********** AUTOMONITOR **********". When you step through your program, z/OS Debugger displays the value of all the variables in the AUTOMONITOR section in hexadecimal format until you enter the DEF command in the prefix area of the AUTOMONITOR line.
- Use the `PREVIOUS` and `BOTH` options while you step (by using the STEP command) through a program to see the values of a variables before and after a statement is run.
- If you use The `PREVIOUS` or `BOTH` options and run through your program with the `GO` command, z/OS Debugger displays the value of a variable on the line that z/OS Debugger ran most recently, which might not be the line that you see in the Source window immediately before the current line.
- When control is transferred between enclaves and any of the following settings are in effect, z/OS Debugger cannot determine the data from the previous enclave:
  - `SET AUTOMONITOR ON LOG` with `PREVIOUS` or `BOTH`
  - `SET AUTOMONITOR ON NOLOG` with `PREVIOUS` or `BOTH`

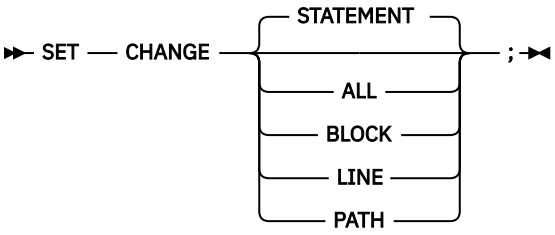
  z/OS Debugger displays a message.
- You can list a single element of an array only for programs compiled with Enterprise PL/I for z/OS, Version 4 or later.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

## SET CHANGE command

Controls the frequency of checking the `AT CHANGE` breakpoints. The initial setting is `STATEMENT/LINE`.




**STATEMENT**
Specifies that the AT CHANGE breakpoints are checked at all statements. STATEMENT is equivalent to 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, except for C and C++ nested blocks.

**LINE**
Is equivalent to 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.

```
SET CHANGE;
```

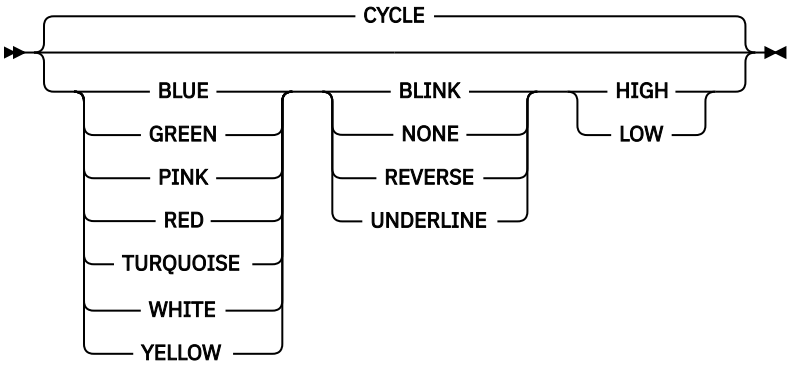
- Specify that AT CHANGE breakpoints are checked at all path points.

```
SET CHANGE PATH;
```

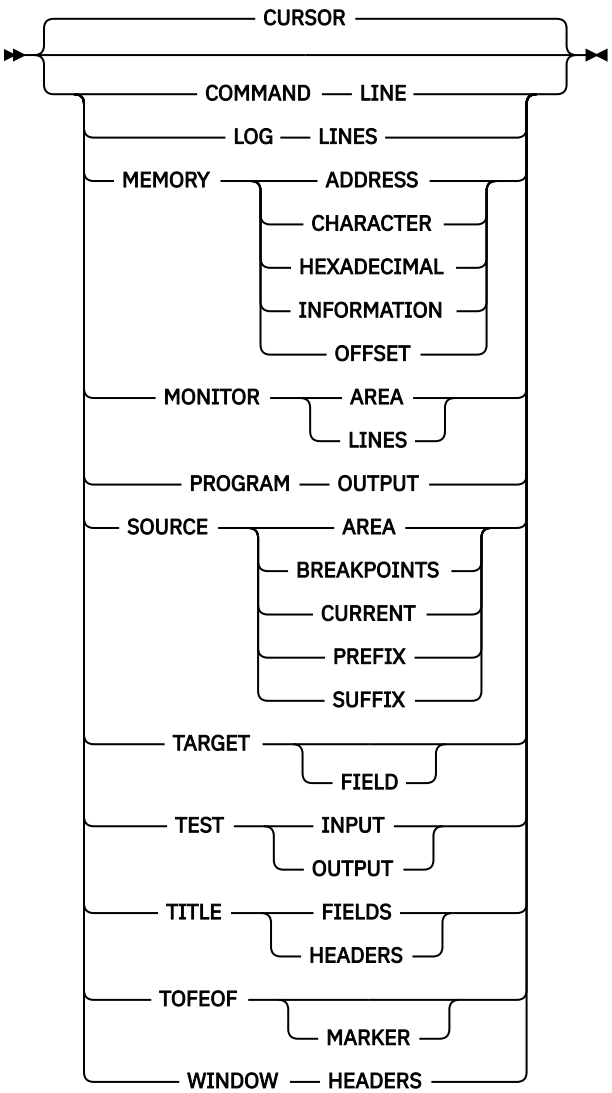
## SET COLOR command (full-screen and line mode)

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.


**color_attributes**




**UI_elements**




**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.

**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).

**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.

**MEMORY ADDRESS**
Selects the address column of the memory dump area.

**MEMORY BASE ADDRESS**
Selects the history lines and the base address of the information area.

**MEMORY CHARACTER**
Selects the character column of the memory dump area.

**MEMORY HEXADECIMAL**
Selects the hexadecimal column of the memory dump area.

**MEMORY INFORMATION**
Selects the history lines of the information area.

**MEMORY OFFSET**
Selects the offset column of the memory dump area.

**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 z/OS Debugger input displayed in the log window.

**TEST OUTPUT**
Selects the z/OS Debugger 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.

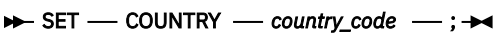
```
SET COLOR YELLOW REVERSE SOURCE AREA;
```

- Set the monitor window display area to high intensity green.

```
SET COLOR HIGH GREEN MONITOR AREA;
```

## SET COUNTRY command

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.




***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 z/OS Debugger.
- At the beginning of an enclave, the settings are those provided by Language Environment, your operating system, or the z/OS Debugger run-time options. For nested enclaves, the parent's settings are restored upon return from a child enclave.

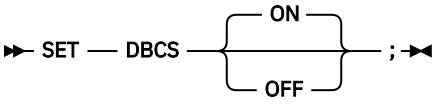
**Example**

Change the current country code to correspond to Japan.

```
SET COUNTRY JP;
```

## SET DBCS command

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.




**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`, z/OS Debugger resets the national language to UEN to remain compatible with DBCS characters.
- If `NATIONAL LANGUAGE` is set to `JPN` or `KOR` and you are using full-screen mode, enter the `SET DBCS ON` command so that z/OS Debugger displays messages correctly.

**Example**

Specify that shift-in and shift-out codes are interpreted.

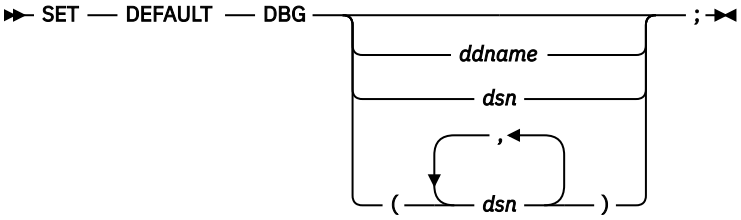
```
SET DBCS ON;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"SET NATIONAL LANGUAGE command" on page 245

# SET DEFAULT DBG command

Defines a default partitioned data set DD name or DS name that z/OS Debugger searches through to locate the .dbg files. The .dbg files are generated by the z/OS XL C/C++ compiler when you select the FORMAT(DWARF) suboption of the DEBUG compiler option. The compiler assigns a name to the file based on what you specified in the FILE suboption of the DEBUG compiler option.




***ddname***
Specifies a valid z/OS DD name. If the operand is less than nine characters long and does not contain a period, z/OS Debugger interprets it as a DD name.

***dsn***
Specifies a valid, fully-qualified z/OS partitioned data set name.

**( *dsn*, *dsn*, ...)**
Specifies a list of valid z/OS partitioned data set names.

**Usage notes**

- You can use this command in remote debug mode.
- If you do not specify a *ddname* or *dsn*, z/OS Debugger clears any previous default dbg setting.
- If the data set name is too long to be typed on one line, suffix it with a trailing hyphen.
- If you are debugging in a CICS or UNIX System Services environment, you cannot use the *ddname* parameter.

**Examples**

- Indicate that the default .dbg file is allocated to DS name SVTRSAMP.TS99992.MYDBG.

  ```
  SET DEFAULT DBG SVTRSAMP.TS99992.MYDBG;
  ```

- The .dbg file for the program MYPROG is in SVTRSAMP.TS99992.MYDBG, which was allocated by using the following command:

  ```
  ALLOC DDNAME(ITEM1) DSNAME('SVTRSAMP.TS99992.MYDBG') SHR
  ```

  To specify the location, enter the following command:

  ```
  SET DEFAULT DBG ITEM1;
  ```

- The .dbg file for the program MYPROG is in JSMITH.CPGMS.DBG, which was allocated by using the following command:

  ```
  ALLOC FI(DBGLIST) DAT('MJONES.OTHER.DBG' 'JSMITH.CPGMS.DBG')
  ```

  To specify the location, enter the following command:

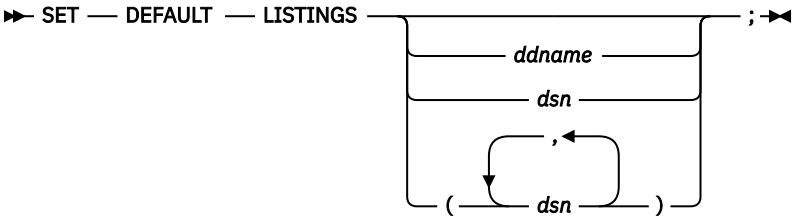
```
SET DEFAULT DBG DBGLIST;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
*IBM z/OS Debugger User's Guide*
“SET SOURCE command” on page 259
“SET DEFAULT MDBG command” on page 221
“SET MDBG command” on page 242
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517
“Specifying the location of source, listing, or separate debug file in remote debug mode by using environment variables” on page 522
"How does z/OS Debugger locate source, listing, or separate debug files?" in the IBM z/OS Debugger User's Guide

# SET DEFAULT LISTINGS command

Defines a default partitioned data set DD name or DS name whose members are searched for program source, listings, or separate debug files.




***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 you specify the EQAOPTS SUBSYS command to enable access to the source file in a library system.

***dsn***
Specifies a valid, fully-qualified z/OS partitioned data set name.

**( *dsn*, *dsn*, )**
Specifies a list of valid z/OS partitioned data set names.

**Usage notes**

- You can use this command in remote debug mode.
- 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 `LOADDEBUGDATA` command.
- If you are debugging in a CICS environment, you cannot 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 .dbg file nor the .mdbg file.

**Examples**

- Indicate that the default listings file is allocated to DS name SVTRSAMP.TS99992.MYLIST.

```
SET DEFAULT LISTINGS SVTRSAMP.TS99992.MYLIST;
```

- The listing for the program MYPROG is in SVTRSAMP.TS99992.MYLIST, which was allocated by using the following command:

```
ALLOC DDNAME(ITEM1) DSNAME('SVTRSAMP.TS99992.MYLIST') 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, z/OS Debugger 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:

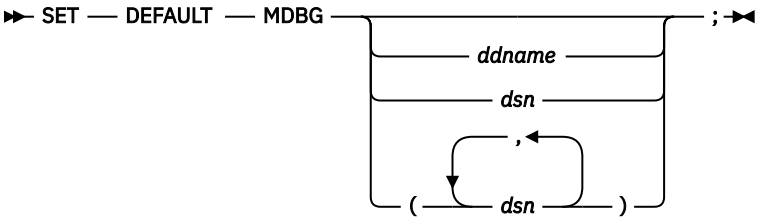
```
SET SOURCE ON (AVER) myid.source.listing(AVERLIST);
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
*IBM z/OS Debugger User's Guide*
“SET SOURCE command” on page 259
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517
"How does z/OS Debugger locate source, listing, or separate debug files?" in the IBM z/OS Debugger User's Guide

## SET DEFAULT MDBG command

Defines a default partitioned data set DD name or DS name that z/OS Debugger searches through to locate the .mdbg files. You create .mdbg files with the dbgld command or the CDADBGLD utility.




***ddname***
Specifies a valid z/OS DD name. If the operand is less than nine characters long and does not contain a period, z/OS Debugger interprets it as a DD name.

***dsn***
Specifies a valid, fully-qualified z/OS partitioned data set name.

**( *dsn*, *dsn*, ...)**
Specifies a list of valid z/OS partitioned data set names.

**Usage notes**

- Before you can use this command, you or your site must specify YES for the EQAOPTS MDBG command, as described in Chapter 6, “EQAOPTS commands,” on page 287. In environments that support environment variables, you can use the EQA_USE_MDBG environment variable to override this option for a specific debugging session.
- You can use this command in remote debug mode.
- If you do not specify a *ddname* or *dsn*, z/OS Debugger clears any previous default mdbg setting.
- If the data set name is too long to be typed on one line, suffix it with a trailing hyphen.
- The SET MDBG command has a higher precedence than the SET DEFAULT MDBG command.
- If you are debugging in a CICS or UNIX System Services environment, you cannot use the *ddname* parameter.

**Examples**

- Indicate that the default .mdbg file is allocated to DS name SVTRSAMP.TS99992.MYMDBG.

```
SET DEFAULT MDBG SVTRSAMP.TS99992.MYMDBG;
```

- The .mdbg file for the DLL MYPROG is in SVTRSAMP.TS99992.MYMDBG, which was allocated by using the following command:

```
ALLOC DDNAME(ITEM1) DSNAME('SVTRSAMP.TS99992.MYMDBG') SHR
```

  To specify the location, enter the following command:

```
SET DEFAULT MDBG ITEM1;
```

- The .mdbg file for load module MYLOAD is in JSMITH.CPGMS.MDBG, which was allocated by using the following command:

```
ALLOC FI(CMDBG) DAT('MJONES.OTHER.MDBG' 'JSMITH.CPGMS.MDBG')
```

  To specify the location, enter the following command:

```
SET DEFAULT MDBG CMDBG;
```

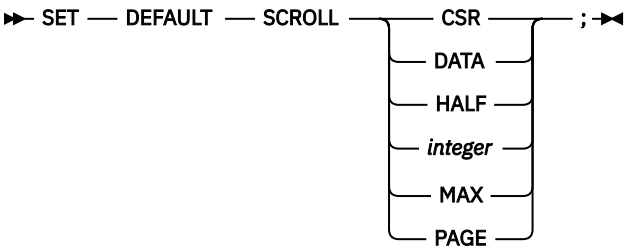
Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**

“SET SOURCE command” on page 259
“SET DEFAULT DBG command” on page 219
“SET MDBG command” on page 242
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517
“Specifying the location of source, listing, or separate debug file in remote debug mode by using environment variables” on page 522
"How does z/OS Debugger locate source, listing, or separate debug files?" in the IBM z/OS Debugger User's Guide.
"Specifying whether z/OS Debugger searches for .mdbg files" in the IBM z/OS Debugger Customization Guide

## SET DEFAULT SCROLL command (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.




**CSR**
Scrolls in the specified direction until the character where the cursor is positioned reaches the edge of the window.

**DATA**
Scrolls by one line less than the window size or by one character less than the window size (if moving left or right).

**HALF**
Scrolls by half the window size.

***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.

**PAGE**
Scrolls by the window size.

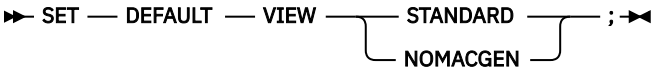
**Example**

Set the default amount to half the size of the window.

```
SET DEFAULT SCROLL HALF;
```

## SET DEFAULT VIEW command

Controls the default view for assembler compile units.




**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**

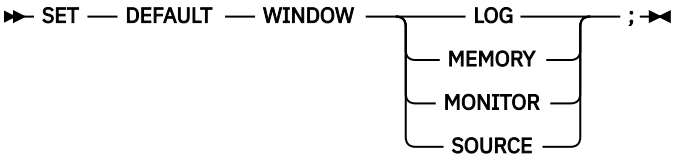
- `SET DEFAULT VIEW` applies only to assembler compile units.
- You can use this command in remote debug mode.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

## SET DEFAULT WINDOW command (full-screen mode)

Specifies which physical 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 physical window areas. The initial setting is SOURCE.




**LOG**
Selects the session log window.

**MEMORY**
Selects the Memory 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 command

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 *
  - 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.
- SET DISASSEMBLY ON is not supported in explicit debug mode. When explicit debug mode is active, z/OS Debugger forces SET DISASSEMBLY OFF.
- You can use this command in remote debug mode.

Refer to the following topics for more information related to the material discussed in this topic.

- **Related references**
- Appendix A, "z/OS Debugger commands supported in Debug Tool compatibility mode," on page 517

## SET DYNDEBUG command

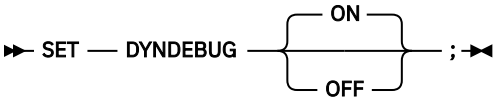
Controls the activation or deactivation of the Dynamic Debug facility.

The Dynamic Debug facility must be activated in order to debug the following types of programs:

- COBOL programs compiled with the Enterprise COBOL for z/OS Version 4 compiler (and earlier) compiled with the NONE or NOHOOK suboptions of the TEST compiler option.
- COBOL programs compiled with the Enterprise COBOL for z/OS Version 5 compiler compiled with the TEST compiler option.
- PL/I programs compiled with Enterprise PL/I for z/OS, Version 3 Release 4 or later, and the NOHOOK suboption of the TEST compiler option
- assembler programs
- disassembled programs (using the disassembly view)
- LangX COBOL programs[1]
- programs that run without the Language Environment run time[1]

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.

The initial setting of DYNDEBUG can be controlled by the EQAOPTS DYNDEBUG command. If no EQAOPTS DYNDEBUG command is used, the initial setting in ON.




**ON**
Activates the Dynamic Debug facility.

**OFF**
Deactivates the Dynamic Debug facility.

**Usage notes**

- After a dynamic debug hook is inserted, either explicitly or implicitly, into any program during a debug session, you cannot change the setting of DYNDEBUG.
- You can use this command in remote debug mode.
- This command does not support 64-bit programs.
- You can debug COBOL programs compiled with the NOHOOK suboption of the TEST compiler option of Enterprise COBOL for z/OS, Version 4, with the Dynamic Debug facility.

[1] In non-CICS environments, SVC screening must be enabled to debug LangX COBOL programs, programs that run without the Language Environment runtime, or programs that are loaded by using the MVS LOAD and LINK macros. See *IBM z/OS Debugger Customization Guide* for instructions on how to manage SVC screening.

- 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
- The Dynamic Debug facility does not support attention interrupts for assembler, disassembly, or LangX COBOL programs or for programs compiled using the following suboptions of the compilers:
  - NOHOOK suboption of the TEST compiler option for the following compilers:
    - Enterprise COBOL for z/OS, Version 4
    - Enterprise PL/I for z/OS, Version 3.4 or later
  - NONE suboption of the TEST compiler option for the following compilers:
    - Enterprise COBOL for z/OS and OS/390, Version 3
    - COBOL for OS/390 & VM, Version 2
- 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 run time, which z/OS Debugger uses instead of the compiled-in hooks. This can improve the performance of the program while running under the control of z/OS Debugger.
  - Any COBOL compiler supported by z/OS Debugger
  - Any C/C++ compiler supported by z/OS Debugger
  - Any PL/I compiler supported by z/OS Debugger
- 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 z/OS Debugger would stop if the same program were compiled in two different ways.

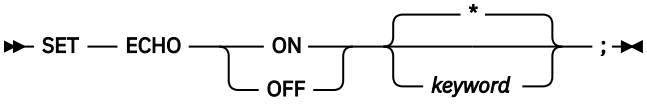
| Program compiled with TEST(ALL) | Program compiled with TEST(NONE) |
|---|---|
| 000001 MOVE... | 000001 MOVE... |
| 000002 ADD... | 000002 ADD... |
| ▶000003 LABEL: ... | 000003 LABEL: ... |
| 000004 MOVE... | ▶000004 MOVE... |

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

## SET ECHO command

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.




**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.

```
SET ECHO OFF;
```

- Specify that GO and STEP commands are displayed.

```
SET ECHO ON *;
```

## SET EQUATE command

Equates a symbol to a string of characters. The equated symbol can be used anywhere a keyword, identifier, or punctuation is used in a z/OS Debugger command. When an equated symbol is found in a z/OS Debugger command (other than the *identifier* operand in SET EQUATE and CLEAR EQUATE), the equated symbol is replaced by the specified string before parsing continues.


***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 LangX 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 z/OS Debugger 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 tstlen 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.

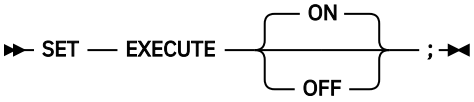
```
SET EQUATE tstlen = "struct1->member.b->c.len";
```

- Specify that the symbol VARVALUE is equated to the command LIST x.

```
SET EQUATE VARVALUE = "LIST x";
```

## SET EXECUTE command

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;
```

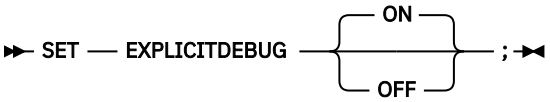
## SET EXPLICITDEBUG command

Controls whether explicit debug mode is active.

When explicit debug mode is not active, z/OS Debugger automatically loads debug data for all high-level language compile units compiled with the TEST or DEBUG compiler option unless you excluded the compile unit or its containing load module by using the NAMES EXCLUDE statement.

When explicit debug mode is active, z/OS Debugger loads debug data only in the following cases:

- z/OS Debugger always loads debug data for the compile unit which is active when z/OS Debugger first becomes active, and the first compile unit of each enclave. In most cases, this is the entry compile unit for the initial load module.
- When a compile unit appears, z/OS Debugger loads debug data whenever you previously entered a LOADDEBUGDATA (LDD) command for the load module and compile unit.
- When a compile unit appears, z/OS Debugger loads debug data for a compile unit that you previously specified on a NAMES INCLUDE CU statement and z/OS Debugger processed the containing load module for any of the following reasons:
  - It is the initial load module.
  - It is a load module that was previously specified on an LDD command.
  - It is a load module that was previously specified on a NAMES INCLUDE LOADMOD statement.
  - It is a load module for which an implicit NAMES INCLUDE LOADMOD has been generated.
- z/OS Debugger loads debug data for the target of a deferred AT ENTRY command.
- z/OS Debugger loads debug data for the entry point compile unit for a load module processed by an AT LOAD command.
- In CICS, when a matching DTCN or CADP profile is found, z/OS Debugger loads debug data for the LoadMod and CU specified in DTCN or the Program and Compile Unit specified in CADP, unless the specified name(s) contain a wildcard character (* or ?).




**ON**
Activate explicit debug mode.

**OFF**
Deactivate explicit debug mode. Explicit debug mode is initially not active.

**Usage notes**

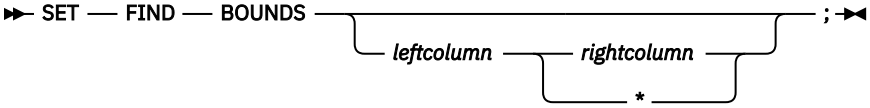
- You can use this command in remote debug mode.
- The SET EXPLICITDEBUG ON command takes effect when you enter the command. By the time you enter the command, z/OS Debugger has already processed the initial load module. To enable explicit debug mode before z/OS Debugger processes the initial load module, use the EQAOPTS EXPLICITDEBUG command.
- z/OS Debugger does not support the SET DISASSEMBLY ON command in explicit debug mode. z/OS Debugger forces it to OFF when you enable explicit debug mode.
- Use explicit debug mode to improve the performance of z/OS Debugger while debugging extremely large or complex programs.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"LOADDEBUGDATA command" on page 167

## SET FIND BOUNDS command

Specifies the default left and right columns for a FIND command in the Source window and in line mode that does not specify any columns information. It is ignored in the Log and Monitor windows.



***leftcolumn***
A positive integer that specifies the leftmost column for the search. This is supported only in the Source window and in line mode. It is ignored in the Log and Monitor windows.

***rightcolumn***
A positive integer that specifies the rightmost column for the search. This is supported only in the Source window and in line mode. It is ignored in the Log and Monitor windows.

*****
Specifies that the length of each source record is used as the right column for the search. This is supported only in the Source window and in line mode. It is ignored in the Log and Monitor windows.

**Usage notes**

- If SET FIND BOUNDS has not been set, the default is 1 for *leftcolumn* and * for *rightcolumn*.
- If you enter SET FIND BOUNDS without operands, the result is 1 for *leftcolumn* and * for *rightcolumn*.
- If you do not specify column boundaries in a FIND command for the Source window or in line mode, the boundaries set by the SET FIND BOUNdS command are used for the FIND command.

**Example**

If you want to find two different strings (paraa and variable-b) in COBOL's Area B, first enter the following command to set the boundaries of the search:

```
SET FIND BOUNDS 12 72;
```

Then enter the following FIND commands to search for the two strings:

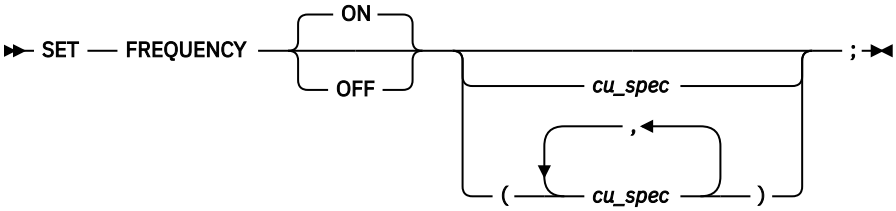
```
FIND paraa;
FIND variable-b;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"FIND command" on page 117
"QUERY command" on page 194

## SET FREQUENCY command

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`.

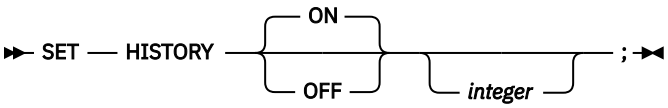
```
SET FREQUENCY ON (main, subr1);
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“cu_spec” on page 13
“LIST FREQUENCY command” on page 155
“SET SUFFIX command (full-screen mode)” on page 261

## SET HISTORY command

Specifies whether entries to z/OS Debugger 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.




**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.

```
SET HISTORY 50;
```

- Turn off history recording.

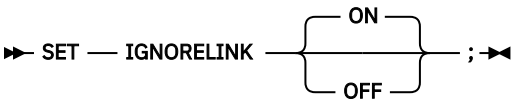
```
SET HISTORY OFF;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“LIST LAST command” on page 155

## SET IGNORELINK command

Specifies that any new LINK level (nested enclave) is ignored while the setting is ON. z/OS Debugger does not gather information or stop at the programs in this newly created enclave. The initial setting is OFF.




**ON**
Programs in new enclaves (links) are ignored. z/OS Debugger does not stop at programs in new enclaves.

**OFF**
Programs in new enclaves (links) are not ignored. z/OS Debugger stops at any breakpoint for a program in new enclaves.

**Usage notes**

- A new enclave is created by language constructs like EXEC LINK or EXEC XCTL, which invoke a new main program.
- This command is valid only in CICS programs.
- You can use this command in remote debug mode.
- DTCN or CADP profiles override the setting of `SET IGNORELINK`.
- You can use the `STEP INTO` command to step into a new enclave, which overrides the `SET IGNORELINK` setting. However, this does not change the setting of `SET IGNORELINK`.
- If you use the `STEP RETURN` command, you can only return to the parent enclave if it was not ignored by z/OS Debugger because at the time it was created, the setting of `SET IGNORELINK` was `OFF`. Otherwise, z/OS Debugger runs to the next breakpoint in a previous enclave that was not ignored by z/OS Debugger or it runs to the end of the application.
- The `DISABLE DTCN`, `ENABLE DTCN`, `DISABLE CADP`, and `ENABLE CADP` commands override the setting of `SET IGNORELINK`. This allows you to debug the new enclave, but does not change the setting of `SET IGNORELINK`.
- Breakpoints are not restored for a compile unit in a new enclave when the `SET IGNORELINK` setting is `ON`.
- z/OS Debugger does not stop for any deferred entry breakpoints for a compile unit in a new enclave when the `SET IGNORELINK` setting is `ON`.
- z/OS Debugger does not stop for any breakpoint in the new enclave when the `SET IGNORELINK` setting is `ON`.
- Conditions raised in the application are reported regardless of the setting of `SET IGNORELINK`.
- You can use this command in a preferences, commands, or global preferences file so that it is run at the beginning of every new debugging session.
- When both `SET IGNORELINK ON` and `SET EXPLICITDEBUG ON` are in effect, and you have run an `LDD`, `AT ENTRY`, or `NAMES INCLUDE` for the initial load module and compile unit for the `LINK` target, z/OS Debugger ignores the `SET IGNORELINK` for that specific `LINK`. However, this does not change the setting of `SET IGNORELINK`.

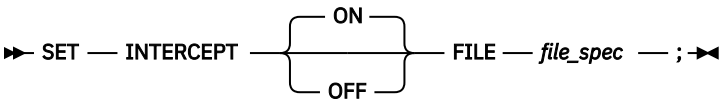
Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“DISABLE command” on page 107
“ENABLE command” on page 113
“QUERY command” on page 194
“STEP command” on page 269

## SET INTERCEPT command (C and C++)

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 entering `SET INTERCEPT OFF` command. The initial setting is `OFF`.




**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 `fopen()` file specifier including `stdin`, `stdout`, or `stderr`. The `FILE` keyword cannot be abbreviated.

**Usage notes**

- For Enterprise COBOL for z/OS Version 5, `ACCEPT` is not supported.
- Intercepted streams or files cannot be part of any C I/O redirection during the execution of a nested enclave.
- 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 `fopen()` file specifier `dd:mydd`. The current programming language setting is C.

```
SET INTERCEPT ON FILE dd:mydd;
```

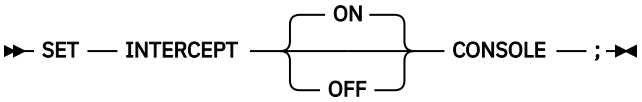
Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"INPUT command (C, C++, and COBOL)" on page 135
"SET REFRESH command (full-screen mode)" on page 252

## SET INTERCEPT command (COBOL, full-screen mode, line mode, batch mode)

Intercepts input to and output from the console. Output and prompts for input are displayed in the log.

Console I/O intercepts remain in effect for the entire debug session, unless you terminate them by entering `SET INTERCEPT OFF` command. The initial setting is `OFF`.




**ON**
Turns on console I/O interception. z/OS Debugger displays output in the log, preceded by the CONSOLE keyword to identify the output. Input causes a prompt entry in the log, with the CONSOLE identified. You can then enter input for the console on the command line by using the `INPUT` command.

**OFF**
Turns off console I/O interception.

**CONSOLE**
Turns I/O interception on or off 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**

- 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.

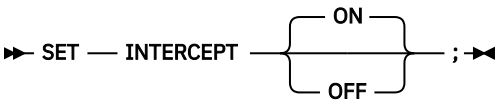
```
SET INTERCEPT CONSOLE;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“INPUT command (C, C++, and COBOL)” on page 135
“SET REFRESH command (full-screen mode)” on page 252

## SET INTERCEPT command (COBOL, remote debug mode)

Intercepts output from COBOL DISPLAY statements. Output is displayed in the Debug Console. Output intercepts remain in effect for the entire debug session, unless you terminate them by entering the SET INTERCEPT OFF command. The initial setting is OFF.




**ON**
Turns on output interception. Output appears in the Debug Console.

**OFF**
Turns off output interception.

**Examples**

Turn on the output interception for the console.

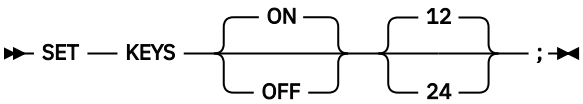
```
SET INTERCEPT ON;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“SET REWRITE command (full-screen mode)” on page 254

## SET KEYS command (full-screen mode)

Controls whether PF key definitions are displayed when the SCREEN setting is ON. The initial setting is ON.




**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.

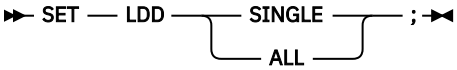
```
SET KEYS OFF;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"SET PFKEY command" on page 247

## SET LDD command

Controls how debug data is loaded for assemblies containing multiple CSECTs. The initial setting is 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 a LangX 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.
- You can use this command in remote debug mode.

**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 topics for more information related to the material discussed in this topic.

**Related tasks**
*IBM z/OS Debugger User's Guide*

**Related references**

"LOADDEBUGDATA command" on page 167
Appendix A, "z/OS Debugger commands supported in Debug Tool compatibility mode," on page 517

# SET LIST BY SUBSCRIPT command (COBOL)

Controls whether z/OS Debugger displays elements in an array as they are stored in memory.

The default setting is OFF.


**ON**
Indicates that z/OS Debugger displays elements of a COBOL array as they are stored in memory.

**OFF**
Indicates that z/OS Debugger displays elements of a COBOL array ordered by element.

**Usage Notes**

- For Enterprise COBOL for z/OS Version 5, the SET LIST BY SUBSCRIPT setting is ignored. The elements of a COBOL array are always displayed as they are stored in memory.
- You can use this command in remote debug mode.
- For the remote debugger, you cannot change the setting of `SET LIST BY SUBSCRIPT` while monitoring expressions. If you want to change the setting of `SET LIST BY SUBSCRIPT`, remove the monitored expressions from the Monitor and Variables Views.

**Examples**

- Assume you declare the following structure in your program:

```
      01  TEAM.
         05 MyTeam OCCURS 3 TIMES.
           10 Name.
             15 LastName   PIC X(20).
             15 FirstName  PIC X(15).
           10 Phone        PIC X(12).
           10 IBMLab       PIC X(20)
```

If you monitor `TEAM` by using `SET LIST BY SUBSCRIPT OFF` (the default setting), z/OS Debugger displays the following output in the monitor window for the remote debugger, when you expand all values:

```
TEAM
  MYTEAM
    NAME
      LASTNAME
        SUB(1) = 'Smith               '
        SUB(2) = 'Johnson             '
        SUB(3) = 'Williams            '
      FIRSTNAME
        SUB(1) = 'Eva            '
        SUB(2) = 'Francisco      '
        SUB(3) = 'Randy          '
    PHONE
      SUB(1) = '408-463-1111'
      SUB(2) = '408-463-2222'
      SUB(3) = '408-463-3333'
    IBMLAB
      SUB(1) = 'Silicon Valley Lab  '
      SUB(2) = 'Silicon Valley Lab  '
      SUB(3) = 'Lexington, Ky       '
```

If you monitor TEAM after running the SET LIST BY SUBSCRIPT ON command, z/OS Debugger displays the following output in the monitor window for the remote debugger:

```
TEAM
  MYTEAM
    MYTEAM(1)
      NAME
        LASTNAME = 'Smith              '
        FIRSTNAME = 'Eva            '
      PHONE = '408-463-1111'
      IBMLAB = 'Silicon Valley Lab  '
    MYTEAM(2)
      NAME
        LASTNAME = 'Johnson            '
        FIRSTNAME = 'Francisco      '
      PHONE = '408-463-2222'
      IBMLAB = 'Silicon Valley Lab  '
    MYTEAM(3)
      NAME
        LASTNAME = 'Williams           '
        FIRSTNAME = 'Randy          '
      PHONE = '408-463-3333'
      IBMLAB = 'Lexington, Ky       '
```

- Assume you declare the same structure as in the above example:

```
01  TEAM.
         05 MyTeam OCCURS 3 TIMES.
           10 Name.
             15 LastName   PIC X(20).
             15 FirstName  PIC X(15).
           10 Phone        PIC X(12).
           10 IBMLab       PIC X(20).
```

If you issue MONITOR LIST (TEAM) with SET LIST BY SUBSCRIPT OFF (the default setting), z/OS Debugger displays the following output in the Monitor window in MFI:

```
MONITOR -+----1----+----2----+----3----+----4----+----5----+----6 LINE: 1 OF 19
******************************** TOP OF MONITOR ********************************
                                  ----+----1----+----2----+----3----+----4----
0001  1 01 TEAM
0002    02 MYTEAM
0003    03 NAME
0004    04 LASTNAME
0005    SUB(1)                   'Smith              '
0006    SUB(2)                   'Johnson            '
0007    SUB(3)                   'Williams           '
0008    04 FIRSTNAME
0009    SUB(1)                   'Eva            '
0010    SUB(2)                   'Francisco      '
0011    SUB(3)                   'Randy          '
0012    03 PHONE
0013    SUB(1)                   '408-463-1111'
0014    SUB(2)                   '408-463-2222'
0015    SUB(3)                   '408-463-3333'
0016    03 IBMLAB
0017    SUB(1)                   'Silicon Valley Lab  '
0018    SUB(2)                   'Silicon Valley Lab  '
0019    SUB(3)                   'Lexington, Ky       '
****************************** BOTTOM OF MONITOR *******************************
```

If you issue MONITOR LIST (TEAM) with SET LIST BY SUBSCRIPT ON, z/OS Debugger displays the following output in the Monitor window in MFI:

```
MONITOR -+----1----+----2----+----3----+----4----+----5----+----6 LINE: 1 OF 20
******************************** TOP OF MONITOR ********************************
                                  ----+----1----+----2----+----3----+----4----
0001  1 01 TEAM
0002    02 MYTEAM
0003    02 MYTEAM(1)
0004    03 NAME
0005    04 LASTNAME              'Smith              '
0006    04 FIRSTNAME             'Eva            '
0007    03 PHONE                 '408-463-1111'
0008    03 IBMLAB                'Silicon Valley Lab  '
0009    02 MYTEAM(2)
```

```
0010    03 NAME
0011    04 LASTNAME             'Johnson            '
0012    04 FIRSTNAME            'Francisco      '
0013    03 PHONE                '408-463-2222'
0014    03 IBMLAB               'Silicon Valley Lab  '
0015    02 MYTEAM(3)
0016    03 NAME
0017    04 LASTNAME             'Williams           '
0018    04 FIRSTNAME            'Randy          '
0019    03 PHONE                '408-463-3333'
0020    03 IBMLAB               'Lexington, Ky       '
****************************** BOTTOM OF MONITOR ******************************
```

# SET LIST BY SUBSCRIPT command (Enterprise PL/I, full-screen mode only)

Controls whether z/OS Debugger displays elements in an array as they are stored in memory.

The default setting is OFF.


**ON**
Indicates that z/OS Debugger displays elements of a PL/I array as they are stored in memory.

**OFF**
Indicates that z/OS Debugger displays elements of a PL/I array ordered by element.

**Examples**

Assume you declare the following array in your program:

```
DCL 01 STRUCA(3),
           05 FIELD1 CHAR(9),
           05 FIELD2 CHAR(9);
```

If you run the command `LIST STRUCA`, with `SET LIST BY SUBSCRIPT OFF` (the default setting), z/OS Debugger displays the following results:

```
 LIST STRUCA ;
STRUCA.FIELD1(1) = 'MYFIELDB1'
STRUCA.FIELD1(2) = 'MYFIELDB2'
STRUCA.FIELD1(3) = 'MYFIELDB3'
STRUCA.FIELD2(1) = 'MYFIELDB1'
STRUCA.FIELD2(2) = 'MYFIELDB2'
STRUCA.FIELD2(3) = 'MYFIELDB3'
```

If you run the command `LIST STRUCA` after running the command `SET LIST BY SUBSCRIPT ON`, z/OS Debugger displays the following results:

```
 LIST STRUCA ;
STRUCA.FIELD1(1) = 'MYFIELDB1'
STRUCA.FIELD2(1) = 'MYFIELDB1'
STRUCA.FIELD1(2) = 'MYFIELDB2'
STRUCA.FIELD2(2) = 'MYFIELDB2'
STRUCA.FIELD1(3) = 'MYFIELDB3'
STRUCA.FIELD2(3) = 'MYFIELDB3'
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
*IBM z/OS Debugger User's Guide*

**Related references**
"LIST expression command" on page 149

## SET LIST TABULAR command

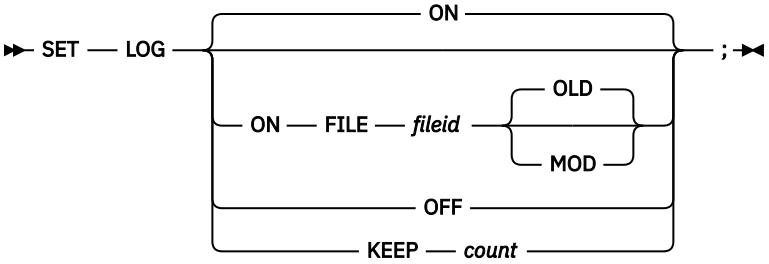
Controls whether to format the output of the LIST command in a tabular format. The default setting is OFF.


**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.

## SET LOG command

Controls whether z/OS Debugger writes each performed command and the resulting output to the log file and defines (or redefines) the name of the log file.




**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. The CICS region must have update authorization to the log file.

If *fileid* has the form of a DD name, z/OS Debugger checks to see if the file is allocated.

In full-screen mode, the log file should *not* be allocated to the 3270 terminal device.

**OLD**
Specifies that the new information is to replace any existing information in the specified file. This operand is ignored if *fileid* specifies a DD name.

**MOD**
Specifies that the new information is appended after any existing information in the specified file. This operand is ignored if *fileid* specifies a DD name.

**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**

- The following list describes how z/OS Debugger determines the initial setting for SET LOG:

  - If a default user log file was *not* specified through the EQAOPTS LOGDSN command, the following rules apply:
    - In a non-CICS environment, if you do *not* allocate INSPLOG DD, the initial setting is OFF.
    - In a non-CICS environment, if you do allocate INSPLOG DD, the initial setting is `ON FILE INSPLOG`.
    - In a CICS environment, the initial setting is OFF.
  - If a default user log file was specified through the EQAOPTS LOGDSN command, the following rules apply:
    - In batch mode, if you do *not* allocate INSPLOG DD, the initial setting is OFF.
    - In batch mode, if you do allocate INSPLOG DD, the initial setting is `ON FILE INSPLOG`.
    - In full screen mode and a non-CICS environment, if you do *not* allocate INSPLOG DD, the initial setting is `ON FILE` *`fileid`*. Specify *fileid* through the EQAOPTS LOGDSN command.
    - In full screen mode and a non-CICS environment, if you do allocate INSPLOG DD, the initial setting is `ON FILE INSPLOG`.
    - In a CICS environment, the initial setting is `ON FILE` *`fileid`*. Specify *fileid* through the EQAOPTS LOGDSN command.
- If the EQAOPTS LOGDSN command was specified, then the EQAOPTS LOGDSNALLOC command can be specified to indicate that, if the log file data set does not exist, z/OS Debugger creates it. This can be used to create the file for new z/OS Debugger users.

  For existing z/OS Debugger users, if you use a SAVESETS data set, then the file contains a `SET LOG` command. If a specification for the EQAOPTS LOGDSN command is created after you have saved settings into your SAVESETS file, z/OS Debugger does not change your saved `SET LOG` command and does not create a new log file data set.

  To learn how to specify the EQAOPTS commands LOGDSN and LOGDSNALLOC, see Chapter 6, “EQAOPTS commands,” on page 287.

  `FILE LOGDSN` is used for the `SET LOG ON` command when both of the following conditions are true:

  - `A SET LOG ON` without a `FILE` *fileid* is issued when LOG is OFF.
  - `ON FILE LOGDSN` was used as the initial setting of `SET LOG` via the `EQAOPTS LOGDSN` command.

  For CICS, if you are not logged in or are logged in under the default user ID, z/OS Debugger does not create or use the file specified for *fileid*.

  For Db2 stored procedures, do not set up z/OS Debugger to create or use the file specified for *fileid*. Since multiple users share the same default data set, multiple users can attempt to write to the data set at the same time. In this environment, if LOGDSN is specified, specify `NULLFILE` for *file-name-pattern*.
- 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.
- 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.
- 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.)
- For remote debug mode, you can only use the `SET LOG ON` and `SET LOG OFF` commands. The `SET LOG ON` command displays messages that explain why it stopped at the current location. The `SET LOG ON` command does not save the contents of the log to a permanent location. When the setting for SET LOG is OFF, messages related to breakpoints are not displayed. For example, the message "Program was stopped due to line/statement breakpoint at statement 232." is not displayed.

  If you enter `SET AUTOMONITOR ON LOG` command, the `SET LOG ON` and `SET LOG OFF` commands are ignored. All messages are displayed.

**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;
```

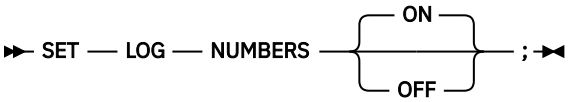
Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
*IBM z/OS Debugger User's Guide*

**Related references**
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

## SET LOG NUMBERS command (full-screen mode)

Controls whether line numbers are shown in the log window. The initial setting is `ON`.




**ON**
Shows line numbers in the log window.

**OFF**
Suppresses line numbers in the log window.

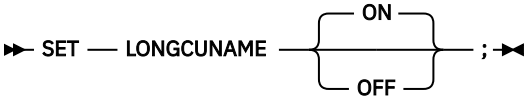
**Example**

Specify that log line numbers are not shown.

```
SET LOG NUMBERS OFF;
```

## SET LONGCUNAME command

Controls whether a short or long CU name is displayed.




**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 one of the following compilers and it is running in the following environment, this command has no effect.

  - Enterprise PL/I for z/OS, Version 3.6 or later
  - Enterprise PL/I for z/OS, Version 3.5, compiler with the PTFs for APARs PK35230 and PK35489 applied
  - Language Environment Version 1.6 through 1.8 with the PTF for APAR PK33738 applied, or later
- The CU name for programs compiled with C, C++, or Enterprise PL/I (before Enterprise PL/I for z/OS, Version 3.6) compilers can have one of the following forms:
  - Fully qualified partitioned data set name and member name
  - A sequential file name
  - An HFS or zFS 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.

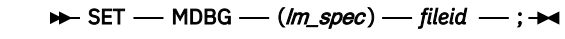
  For these forms of compile unit names, z/OS Debugger displays short names in one of the following manners:
  - For PDS file names, the short name is only the member name
  - For sequential file names, the short name is the lowest level qualifier (name segment)
  - For HFS or zFS file names, the short name is the file name, without path name
- z/OS Debugger 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`), z/OS Debugger displays an error message and does not set the breakpoint. However, if you enter the command `AT ENTRY ARRAY3 ::> 'ARRAY3' :>ARRAY3`, z/OS Debugger sets the breakpoint or defers setting the breakpoint until the entry point is known to z/OS Debugger.
- 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 MDBG command

Associates a .mdbg file to one load module or DLL.




***lm_spec***
The name of a valid load module or DLL.

***fileid***
Identifies the .mdbg file that contains the debug information for the load module or DLL.

In z/OS, *fileid* is a DD name, a fully qualified partitioned data set and member name, a sequential file, or an HFS or zFS path and file name.

In CICS, *fileid* is a fully-qualified data set name or an HFS or zFS path and file name.

If *fileid* is less than nine characters in length and does not contain a period, z/OS Debugger assumes it is a DD name. z/OS Debugger checks to see if it is allocated. If it is not allocated, then z/OS Debugger assumes *fileid* is a data set name.

**Usage notes**

- Before you can use this command, you or your site must specify YES for the EQAOPTS `MDBG` command, as described in Chapter 6, “EQAOPTS commands,” on page 287. In environments that support

environment variables, you can use the EQA_USE_MDBG environment variable to override this option for a specific debugging session.

- You can use this command if you created a .mdbg file that contains debug information, including captured source.
- You can create .mdbg files that contain debug information, including captured source, only if you compile your program with z/OS XL C/C++, Version 1.10, or later.
- z/OS Debugger does not search for the .mdbg file specified in *fileid* until the application loads that load module or DLL. The following list provides some examples of when z/OS Debugger searches for the .mdbg file:
  - If you enter the SET MDBG command and you specify the currently running load module or DLL in *lm_spec*, z/OS Debugger immediately searches for the .mdbg file specified in *fileid*. If z/OS Debugger cannot find the file, it displays an error message.
  - You specify the SET MDBG command in your commands file. When your application calls a function in that load module or DLL, then z/OS Debugger searches for the .mdbg file. If z/OS Debugger cannot find the file, it displays an error message.
  - You enter the SET MDBG command, then you set an AT LOAD breakpoint for that load module or DLL. When z/OS Debugger encounters that breakpoint, then it searches for the .mdbg file. If z/OS Debugger cannot find the file, it displays an error message.

**Examples**

- Specify that FANAYA.MYLOAD.MDBG is the location of the .mdbg file for load module MYLOAD. z/OS Debugger searches for this file when it needs to retrieve debug information for load module MYLOAD.

```
SET MDBG (MYLOAD) FANAYA.MYLOAD.MDBG;
```

- Indicate that the .mdbg file for DLL /u/userid/code/mydll is located in HFS or zFS under the path and file name /u/userid/code/mydll.mdbg:

```
SET MDBG ("/u/userid/code/mydll") /u/userid/code/mydll.mdbg;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related concepts**

"How does z/OS Debugger locate source, listing, or separate debug files?" in the IBM z/OS Debugger User's Guide

**Related tasks**

“Specifying the location of source, listing, or separate debug file in remote debug mode by using environment variables” on page 522

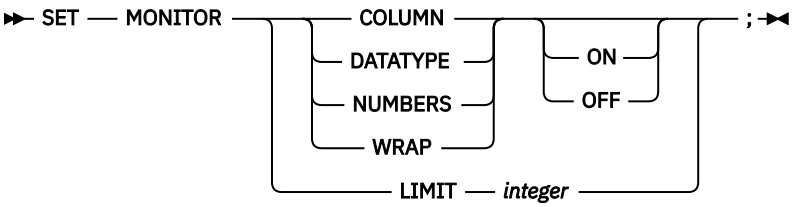
"Specifying whether z/OS Debugger searches for .mdbg files" in the IBM z/OS Debugger Customization Guide

**Related references**

“SET SOURCE command” on page 259
“SET DEFAULT DBG command” on page 219
“SET DEFAULT MDBG command” on page 221
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

## SET MONITOR command

Controls the format and layout of variable names and values displayed in the Monitor window.




**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`.

**LIMIT *integer***
Controls the number of scrollable lines that z/OS Debugger displays in the Monitor window. The default value for *integer* is 1000. If you specify a new value, it must be greater than or equal to 1000, but less than or equal to 30000.

**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 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 z/OS Debugger 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.

Monitoring large amounts of data might require large amounts of storage; this might cause a problem at some sites. Verify that there is enough storage to monitor large data items or data items that contain a large number of elements.

**Example**

- Enter the following command to specify that you do not want line numbers displayed in the Monitor window:

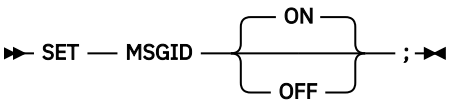
```
SET MONITOR NUMBERS OFF;
```

- 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 command

Controls whether the z/OS Debugger 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 command

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 z/OS Debugger activity only. The initial setting is supplied by Language Environment or the NATLANG z/OS Debugger 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, z/OS Debugger 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 3270 terminal emulator must be capable of displaying DBCS characters, and the VTAM LOGMODE MODEENT macroinstruction used for the terminal session must contain the following specification(s):
  1. For CICS, full-screen mode using the Terminal Interface Manager and TSO, the LOGMODE MODEENT macroinstruction must contain a PSERVIC parameter value that indicates that the terminal has extended data stream capability and that the terminal is to be queried for alternate screen size.
  2. For TSO, in addition, the LOGMODE MODEENT macroinstruction must contain a LANG parameter value where BIT 0 is 1 (ON). TSO/VTAM uses this bit to indicate that devices with extended data stream capability are queried for language information (DBCS capability).

     You can query this bit in ISPF in the following way:

     – In ISPF, select option 0 (Settings). Press Enter.
     – On the command line, enter: environ. Press Enter.
     – Tab to the section Terminal Status (TERMSTAT). In the Enable field, enter 2 (Query terminal information). Press Enter.
     – Several pages of statistics appear. In the section GTTERM Information, note the value of the highest bit in the second byte of the ATTRIBUTE BYTE (the Language Field). The value of this bit must be 1 (ON). For example, if the value of the ATTRIBUTE BYTE is x008000C9, then DBCS characters display correctly because the second byte is x80. However, if the value of the ATTRIBUTE BYTE is x000000C9, DBCS characters are not displayed properly.
- The language you select by using the SET NATIONAL LANGUAGE command affects both your application and z/OS Debugger.
- At the beginning of an enclave, the settings are those provided by Language Environment, your operating system, or the NATLANG z/OS Debugger run-time option. For nested enclaves, the parent's settings are restored upon return from a child enclave.
- If NATIONAL LANGUAGE is set to JPN or KOR and you are using full-screen mode, enter the SET DBCS ON command so that z/OS Debugger displays messages correctly.

**Examples**

- Set the current national language to Japanese.

```
SET NATIONAL LANGUAGE JPN;
SET DBCS ON;
```

- Set the current national language to United States English.

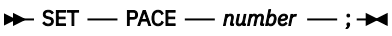
```
SET LANGUAGE ENU;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“SET DBCS command” on page 218

## SET PACE command

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.




***number***
A decimal number between 0 and 9999; it must be a multiple of 0.5.

**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 *IBM z/OS Debugger 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.

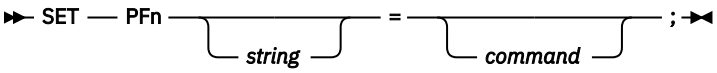
**Example**

Set the animated execution pace to 1.5 steps per second.

```
SET PACE 1.5;
```

## SET PFKEY command

Associates a z/OS Debugger command with a Program Function key (PF key). This setting is not supported in batch mode.




**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. For C and C++, the string must be surrounded by quotation marks ("). For COBOL, LangX COBOL, PL/I, assembler, and disassembly, the string can be surrounded by either quotation marks (") or apostrophes (').

***command***
A valid z/OS Debugger command, partial command, or multiple commands.

If you specify multiple commands, you must surround them with string delimiters. For C and C++, you must surround them with quotation marks ("). For COBOL, LangX COBOL, PL/I, assembler, and disassembly, you can surround them with either quotation marks (") or apostrophes (').

**Usage notes**

- If you specify the ? as the *command*, the ? is understood to be the command, not a request for syntax help.
- In z/OS Debugger, 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++:

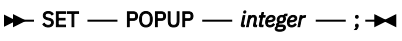
```
SET PF5 "Down" = IMMEDIATE SCROLL DOWN;
```

In all cases, the setting for PF17 remains the same.

## SET POPUP command

Controls the number of lines displayed in the Command pop-up window in the following situations:

- You enter the POPUP command without specifying the number of lines.
- z/OS Debugger opens the Command pop-up window when you enter a continuation character or an incomplete command in the command line.




***integer***
The number of lines in the Command pop-up window when z/OS Debugger opens it. The initial default number of lines is 15.

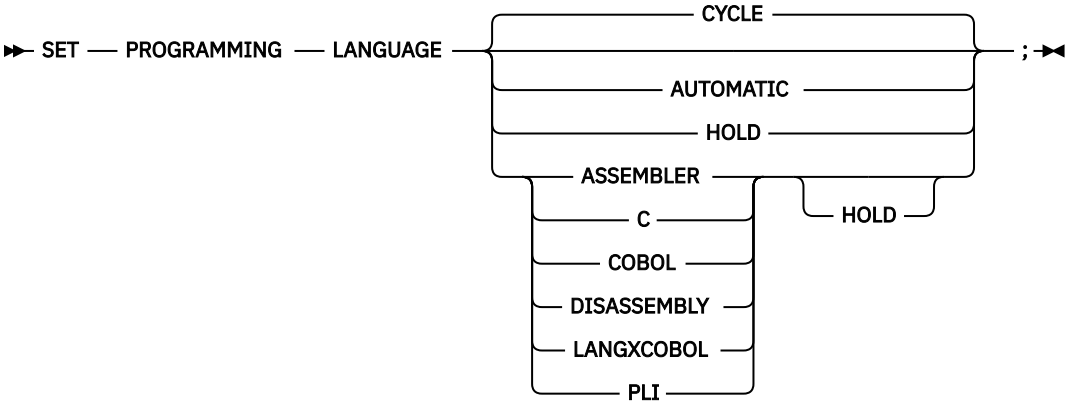
**Related references**

"POPUP command" on page 191

## SET PROGRAMMING LANGUAGE command

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 z/OS Debugger 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.




**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. z/OS Debugger 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.

**LANGXCOBOL**
Sets the current programming language to LangX 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 z/OS Debugger 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.

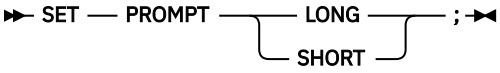
**Example**

Specify that C or C++ is the current programming language.

```
SET PROGRAMMING LANGUAGE C;
```

## SET PROMPT command (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.

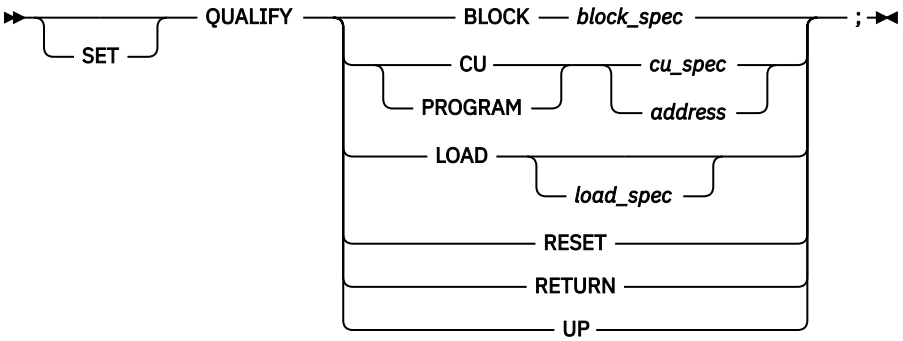
**Example**

Specify that the long form of prompt message is used.

```
SET PROMPT LONG;
```

# SET QUALIFY command

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, enter the SET QUALIFY command to change the qualification. The SET keyword is optional. The QUALIFY keyword can be abbreviated.




**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.

***address***
An address within the CU that you want to qualify to.

**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**

- In the Source window, you can type over the name displayed in the SOURCE field of the header area, then press Enter. z/OS Debugger creates and runs the corresponding SET QUALIFY CU command.
- 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 z/OS Debugger and are in the current enclave
  - Load modules, compile units, and blocks that are not known to z/OS Debugger
  - Non-Language Environment 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 GO command.
- The SET QUALIFY command cannot modify the point of view to a z/OS Debugger 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, z/OS Debugger 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.
- When you use SET QUALIFY *address*, *address* can be any address within the corresponding CU. This form can be especially useful when qualifying to a CU within a non-reentrant load module, when more than one copy of the load module exists in memory.
- If 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 z/OS Debugger, z/OS Debugger runs an implicit LOAD command for the load module. 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 GO 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.
- You cannot use the SET QUALIFY LOAD or SET QUALIFY CU command to implicitly load a DLL.
- If 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, z/OS Debugger creates an implicit CU. 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 GO 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 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 or to create a Language Environment compile unit. You can only use these commands to load a Language Environment load module or create a Language Environment compile unit after Language Environment has been initialized in the current enclave.
- You can use the SET QUALIFY CU and SET QUALIFY LOAD commands in remote debug mode. However, these commands in remote debug mode do not set the current point of view to the specified compile unit or specified load module. Instead, the user should receive a message similar to 'EQA2476I An implicit LOAD was issued for module *module_name*' in the Debug Console. The user must select the specified compile unit or load module in the Modules view to update the current point of view.

**Examples**

- Indicate to z/OS Debugger 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`.

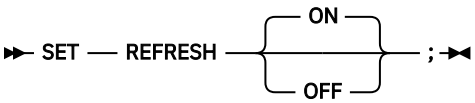
```
SET QUALIFY BLOCK blockx;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“block_spec” on page 12
“cu_spec” on page 13
“load_spec” on page 15
Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

## SET REFRESH command (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.




**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.

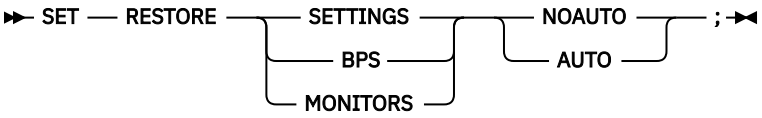
**Example**

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 command

Controls the restoring of settings, breakpoints, and monitor specifications.




**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 z/OS Debugger 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 z/OS Debugger at startup.

**Usage notes**

- When SETTINGS are restored, they are restored before any preference or commands files are processed.
- If you use any of the following combinations of commands, z/OS Debugger restores only specific settings:
  - `SET RESTORE BPS AUTO` with `SET SAVE SETTINGS AUTO` and `SET RESTORE SETTINGS NOAUTO`
  - `SET RESTORE MONITORS AUTO` with `SET SAVE SETTINGS AUTO` and `SET RESTORE SETTINGS NOAUTO`

  With any of these combinations of commands, z/OS Debugger restores the following settings:
  - SET RESTORE BPS
  - SET RESTORE MONITORS
  - SET SAVE BPS
  - SET SAVE MONITORS

  If you use `SET RESTORE BPS NOAUTO` and `SET RESTORE MONITORS NOAUTO` with `SET RESTORE SETTINGS NOAUTO`, z/OS Debugger does not restore any settings.
- 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, you must do one of the following tasks:
  - 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 specify that `NULLFILE` be the name of the default data set through the EQAOPTS commands SAVESETDSN and SAVEBPDSN.
- If the EQAOPTS commands SAVESETDSNALLOC and SAVEBPDSNALLOC are specified, z/OS Debugger creates the data sets, if they don't exist, then runs the corresponding `SET RESTORE SETTINGS AUTO` or `SET RESTORE BPS AUTO` commands to enable their usage. To learn how to specify the EQAOPTS commands SAVESETDSNALLOC and SAVEBPDSNALLOC, see Chapter 6, “EQAOPTS commands,” on page 287.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“RESTORE command” on page 201
“SET SAVE command” on page 255

## SET REWRITE command (full-screen mode)

Forces a periodic screen rewrite during long sequences of output.


***number***
Specifies how many lines of intercepted output are written by the application program before z/OS Debugger refreshes the screen. The initial setting is 50.

**Examples**

Force screen rewrite after each 100 lines of screen output.

```
SET REWRITE EVERY 100;
```

# SET REWRITE command (remote debug mode)

Sets the maximum number of COBOL DISPLAY statements that the remote debugger displays in the Debug Console.


***number***
Specifies the maximum number of COBOL DISPLAY statements that the remote debugger displays in the Debug Console. The initial setting is 50.

**Usage note**

If the remote debugger needs to display more than *number*, the remote debugger begins to delete the oldest DISPLAY statements so that it can display the newest DISPLAY statements.

**Examples**

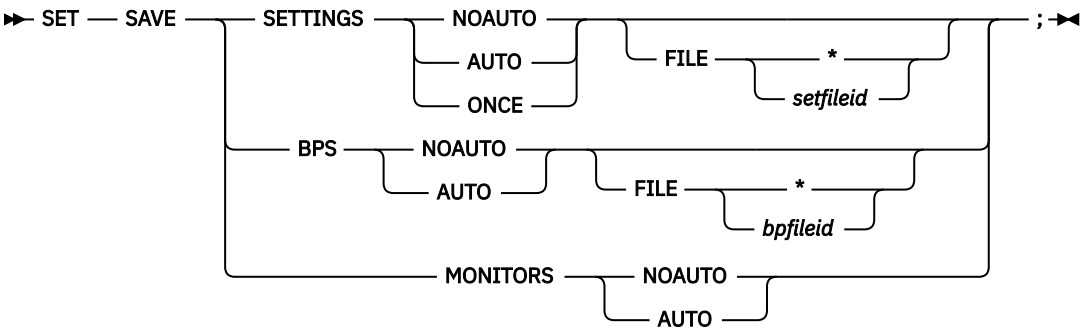
Set the maximum number of COBOL DISPLAY statements to display to 100:

```
SET REWRITE 100;
```

**Related references**
"SET INTERCEPT command (COBOL, remote debug mode)" on page 234

# SET SAVE command

Controls the saving of settings, breakpoints, and monitor specifications.




**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 all monitor and LOADDEBUGDATA (LDD) specifications are to be saved.

**NOAUTO**
Indicates that at z/OS Debugger 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 z/OS Debugger 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 specifying the EQAOPTS commands SAVESETDSN and SAVEBPDSN.

**FILE *setfileid***
Indicates the data set name to be used to save and restore settings. The data set must exist before running this command.

In z/OS, *setfileid* is a DD name or a fully-qualified data set name (without apostrophes (')). In CICS, *setfileid* is a fully-qualified data set name.

If *setfileid* is less than nine characters in length and does not contain a period, z/OS Debugger 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 before running this command.

In z/OS, *bpfileid* is a DD name or a fully-qualified data set name (without apostrophes (')). In CICS, *bpfileid* is a fully-qualified data set name.

If *bpfileid* is less than nine characters in length and does not contain a period, z/OS Debugger 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. z/OS Debugger 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, you must do one of the following tasks:
  - 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 specify that NULLFILE be the name of the default data set through the EQAOPTS commands SAVESETDSN and SAVEBPDSN.
- 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.
- If the EQAOPTS commands SAVESETDSNALLOC and SAVEBPDSNALLOC are specified, z/OS Debugger creates the data sets, if they don't exist, and then runs the corresponding SET SAVE SETTINGS AUTO or SET SAVE BPS AUTO commands to enable their usage. To learn how to specify the EQAOPTS commands SAVESETDSNALLOC and SAVEBPDSNALLOC, see Chapter 6, “EQAOPTS commands,” on page 287.

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**

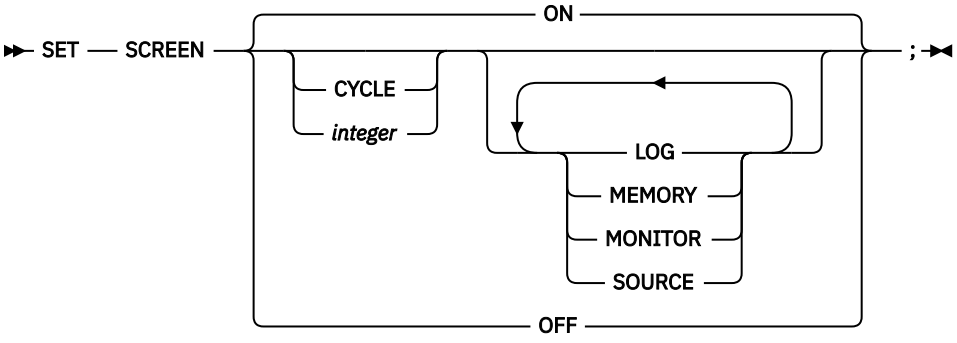
*IBM z/OS Debugger User's Guide*

**Related references**

"SET RESTORE command" on page 253
"RESTORE command" on page 201

## SET SCREEN command (full-screen mode)

Controls how information is displayed on the screen. The initial setting is ON.




**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 or MEMORY**
Specifies the sequence of window assignments within the selected configuration (left to right, top to bottom). There must be three objects specified and they must all be different. You cannot specify both MEMORY and LOG in the same sequence.

**ON**
Activates the z/OS Debugger 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 the Terminal Interface Manager 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 the Terminal Interface Manager.
- SET SCREEN OFF is ignored in CICS full-screen mode and in z/OS batch while debugging in full-screen mode using the Terminal Interface Manager.

### Examples

- Indicate that the z/OS Debugger 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 SOURCE;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
"SET SCREEN command (full-screen mode)" on page 258
"Customizing your full-screen session" in the *IBM z/OS Debugger User's Guide*

## SET SCROLL DISPLAY command (full-screen mode)

Controls whether the scroll field is displayed when operating in full-screen mode. The initial setting is ON.


**ON**
Displays scroll field.

**OFF**
Suppresses scroll field.

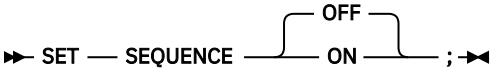
**Example**

Specify that the scroll field is suppressed.

```
SET SCROLL DISPLAY OFF;
```

## SET SEQUENCE command (PL/I)

Controls whether z/OS Debugger interprets data after column 72 in a commands or preference file as a sequence number.




**ON**
Allows sequence numbers in 73-80 columns in the commands or preferences file.

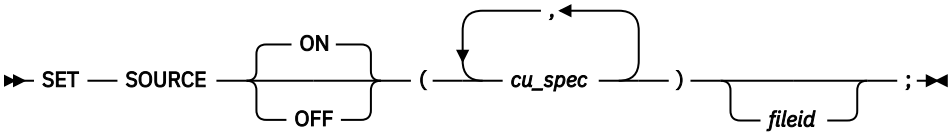
**OFF**
Does not allow sequence numbers in the commands or preferences file.

**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. Afterward, z/OS Debugger processes 1-72 columns and ignores everything after column 72.

## SET SOURCE command

Associates a source file, compiler listing or separate debug file with one or more compile units and specifies whether the source file or listing is displayed when the compile unit is active.




**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 or zFS path and file name.

In CICS, *fileid* is a fully-qualified data set name or an HFS or zFS path and file name.

If *fileid* is less than nine characters in length and does not contain a period, z/OS Debugger assumes it is a DD name. z/OS Debugger 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 z/OS Debugger (by using a previous SET SOURCE command) or the default *fileid* is valid.

*Fileid* cannot be a DD name if the data set allocated to it is C, C++ or Enterprise PL/I source and you specify the EQAOPTS SUBSYS command to enable access to the source file in a library system.

**Usage notes**

- This command is not supported for the Enterprise COBOL for z/OS Version 5 compiler.
- 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 .dbg or .mdbg 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.
- When you specify a *cu_spec* that identifies a compile unit that is not currently known to z/OS Debugger, z/OS Debugger looks for a deferred LOADDEBUGDATA command with the specific *cu_spec*. If z/OS Debugger finds such a deferred LOADDEBUGDATA command, z/OS Debugger associates the *fileid* with the deferred LOADDEBUGDATA command. When the compile unit appears and is activated, z/OS Debugger loads the EQALANGX data from the specified file.
- 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 quotation marks (") or apostrophes (').
- 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 z/OS Debugger and issue:

```
SET SOURCE ON (prog1) mainprog;
```

  When prog1 is made current during the debug session, z/OS Debugger 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 or zFS 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 a version of the Enterprise PL/I compiler that is earlier than Enterprise PL/I for z/OS, Version 3.5 with the PTFs for APARs PK35230 and PK35489 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 LONGCUNAME OFF;
SET SOURCE ON (myprog) CODE.PLILIB.SOURCE(myprog)
```

- A PL/I program is compiled with one of the following compilers and it is running in the following environment:
  - Enterprise PL/I for z/OS, Version 3.6 or later
  - Enterprise PL/I for z/OS, Version 3.5, compiler with the PTFs for APARs PK35230 and PK35489 applied
  - Language Environment Version 1.6 through 1.8 with the PTF for APAR PK33738 applied, or later

  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):

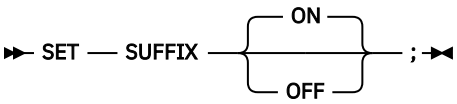
```
SET SOURCE ON (myprog) CODE.PLILIB.SOURCE(myprog)
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“cu_spec” on page 13
“LIST command” on page 140
“SET DEFAULT LISTINGS command” on page 220
“SET DEFAULT DBG command” on page 219
“SET DEFAULT MDBG command” on page 221
“SET MDBG command” on page 242

## SET SUFFIX command (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.




**ON**
Displays the suffix column.

**OFF**
Suppresses the suffix column.

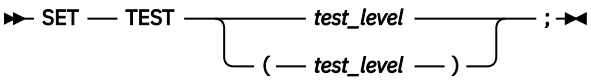
**Example**

Specify that the suffix column is displayed.

```
SET SUFFIX ON;
```

# SET TEST command

Overrides the initial TEST run-time options specified at invocation. The initial setting is ALL.




***test_level***
Specifies what exception conditions cause z/OS Debugger to gain control, even though no breakpoint exists. The parentheses are optional.

`Test_level` can include the following:

**ALL**
Specifies that z/OS Debugger gains control when any of the following conditions occur:

- An attention interrupt occurs.
- A Language Environment enclave is abnormally terminated or there is an MVS or CICS ABEND when a program is running without the Language Environment run time.
- Language Environment terminates normally due to a COBOL STOP RUN, PL/I STOP, or EXEC CICS RETURN.
- Language Environment raises a condition of severity 1 or above. If the FINISH, CEE066 or CEE067 thread termination condition is raised by Language Environment and the EQAOPTS `THREADTERMCOND` command is specified, z/OS Debugger does not gain control. You or your system administrator can specify this command by creating an EQAOPTS load module or providing the command at run time.

If a condition occurs and a breakpoint exists for the condition, z/OS Debugger runs the commands specified in the breakpoint. If a condition occurs and a breakpoint does not exist for that condition, or if an attention interrupt occurs, z/OS Debugger does one of the following options:

- In interactive mode, z/OS Debugger reads commands from a commands file (if it exists) or prompts you for commands.
- In noninteractive mode, z/OS Debugger reads commands from the commands file.

**ERROR**
Specifies that only the following conditions cause z/OS Debugger 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
- 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 z/OS Debugger
  - 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, z/OS Debugger reads commands from a commands file or prompts you for commands in interactive mode.

**NONE**
Specifies that z/OS Debugger 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.

**Usage note**

If the EQAOPTS THREADTERMCOND command prevents z/OS Debugger from stopping when a FINISH, CEE066, or CEE067 thread termination condition is raised by Language Environment, z/OS Debugger does not gain control when these conditions are raised. If you want z/OS Debugger to gain control when these conditions are raised, you can set an AT OCCURRENCE breakpoint or change the EQAOPTS THREADTERMCOND command to allow z/OS Debugger to gain control.

**Examples**

- Indicate that only an attention interrupt or exception causes z/OS Debugger to gain control when no breakpoint exists.

```
SET TEST ERROR;
```

- Indicate that no condition causes z/OS Debugger to gain control unless a breakpoint exists for that condition.

```
SET TEST NONE;
```

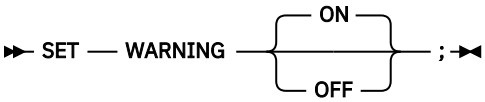
Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
*IBM z/OS Debugger User's Guide*

**Related references**
“AT OCCURRENCE command” on page 65
*z/OS Language Environment Debugging Guide*

## SET WARNING command (C, C++, COBOL, and PL/I)

Controls display of the z/OS Debugger 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 z/OS Debugger 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 z/OS Debugger 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**

- For programs compiled with Enterprise COBOL for z/OS Version 5 or later, either with the OPTIMIZE option and the TEST(NOEJPD) option specified or optimized by Automatic Binary Optimizer for z/OS, you can control whether the GOTO or JUMPTO commands are used by executing SET WARNING OFF. This functionality is provided with a strong warning, so be aware that the use of GOTO or JUMPTO might cause abends in this configuration. When you use GOTO or JUMPTO with a program that is compiled by using OPT and TEST(NOEJPD), any type of failures cannot be investigated by IBM service.

  For example, if you get an abend on a GOTO command when debugging a COBOL program that is compiled with OPT and TEST(NOEJPD), you might want to try the same GOTO in a program compiled with OPT and TEST(EJPD). When a program is compiled with OPT and TEST(NOEJPD) , failures or abends caused by GOTO or JUMPTO commands are not investigated by IBM Service.

  You can get the best behavior of GOTO and JUMPTO in programs compiled with OPT and TEST(NOEJPD) if any one of the following conditions is true:

  – The target of the GOTO or JUMPTO command is a paragraph name or a section name (label).
  – The target of the GOTO or JUMPTO command is the first statement in the paragraph or section.

  These statements are targets of COBOL statements PERFORM or GOTO in the COBOL program.

  To get GOTO and JUMPTO behavior that is fully supported, you must compile programs with NOOPT and TEST, or with OPT and TEST(EJPD). However, these programs do not run as fast as programs compiled with OPT and TEST(NOEJPD).
- After applying the z/OS Debugger PTF for APAR PM75819 and the COBOL runtime PTF for APAR PM80361, you can control whether the GOTO or JUMPTO commands can be used when debugging a COBOL program that is compiled with the OPTIMIZE option and the TEST(NOHOOK, NOEJPD) or TEST(NONE) option. In those cases, the compiler has not enabled GOTO and JUMPTO, but if SET WARNING OFF is used, you can use the GOTO and JUMPTO commands.

  This functionality is provided with a strong warning, so be aware that the use of GOTO or JUMPTO might cause abends in this configuration. When you use GOTO or JUMPTO with a program that is compiled by using OPT and TEST(NOEJPD) or TEST(NONE) options, any type of failures cannot be reviewed by IBM service.

  For example, if you get an abend on a GOTO command when debugging a COBOL program that is compiled with OPT and TEST(NOEJPD), you might want to try the same GOTO in a program compiled with OPT and TEST(EJPD). When a program is compiled with OPT and TEST(NOEJPD) or TEST(NONE) options, failures or abends caused by GOTO or JUMPTO commands are not investigated by IBM Service.

  Without the PTF changes referenced above, GOTO and JUMPTO commands are disabled for COBOL programs that are compiled with OPT and TEST(NOEJPD) or TEST(NONE), regardless of the SET WARNING setting used.

  With the PTF changes referenced above, you can compile your code for maximum performance and use the GOTO and JUMPTO commands when SET WARNING OFF is set to ON. However, the results of using the commands might be unpredictable. You can get the best behavior of GOTO and JUMPTO in programs compiled with OPT and TEST(NOEJPD) or TEST(NONE) if any one of the following conditions is true:

  - The target of the GOTO or JUMPTO command is a paragraph name or a section name (label).
  - The target of the GOTO or JUMPTO command is the first statement in the paragraph or section.
  - These procedures are targets of COBOL statements PERFORM or GOTO in the COBOL program.

  To get GOTO and JUMPTO behavior that is fully supported, you must compile programs with NOOPT and TEST, or with OPT and TEST(EJPD). However, these programs do not run as fast as programs compiled with OPT and TEST(NOEJPD).
- You can use this command in remote debug mode.
- z/OS Debugger 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.
- z/OS Debugger 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 the following compilers:
  - Enterprise COBOL for z/OS, Version 4.1
  - Enterprise COBOL for z/OS and OS/390, Version 3 Release 2 or later
  - Enterprise COBOL for z/OS and OS/390, Version 3 Release 1 with APAR PQ63235 installed
  - COBOL for OS/390 & VM, Version 2 Release 2
  - COBOL for OS/390 & VM, Version 2 Release 1 with APAR PQ63234 installed

  However, results might be unpredictable. To obtain more predictable results, compile your program with Enterprise COBOL for z/OS, Version 4.1, and specify the EJPD suboption of the TEST compiler option. However, variables that are declared with the VALUE clause to initialize them cannot be modified.
- When z/OS Debugger evaluates a conditional expression (for example, the *condition* of the WHEN clause of the AT CHANGE command) and the conditional expression is invalid, then z/OS Debugger does one of the following actions:
  - If SET WARNING is set to ON, z/OS Debugger stops and displays a message that it could not evaluate the conditional expression. You need to enter a command to indicate what action you want z/OS Debugger to take.
  - If SET WARNING is set to OFF, z/OS Debugger does not stop nor display a message that it could not evaluate the conditional expression. z/OS Debugger continues running the program.

**Example**

Specify that conditions result in a diagnostic message.

```
SET WARNING ON;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**

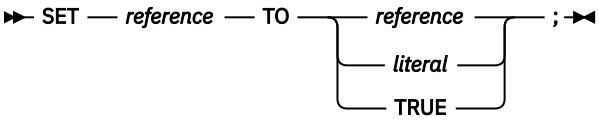
"Controlling z/OS Debugger when a comparison is invalid" in the IBM z/OS Debugger User's Guide

**Related references**

Appendix A, “z/OS Debugger commands supported in Debug Tool compatibility mode,” on page 517

# SET command (COBOL)

The SET command assigns a value to a COBOL reference. The SET keyword cannot be abbreviated.




***reference***
A valid z/OS Debugger COBOL reference.

***literal***
A valid COBOL numeric literal constant.

**TRUE**
The value assigned to a COBOL 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 6, Version 1 Release 7, and Version 1 Release 8.

**Usage notes**

- For Enterprise COBOL for z/OS Version 5, you cannot use the SET command to set an index with a non-integer data item (e.g. `PIC 9v9`).
- For Enterprise COBOL for z/OS Version 5, you can set a pointer to the address of an array. The pointer is set to the start of the array.
- You can assign the value TRUE only to a COBOL level-88 *reference*.
- If z/OS Debugger 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 z/OS Debugger 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.
- z/OS Debugger provides a hexadecimal constant that can be used with the SET command, where the hexadecimal value is preceded 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 topics for more information related to the material discussed in this topic.

**Related tasks**
*IBM z/OS Debugger User's Guide*

**Related references**
“Allowable moves for the z/OS Debugger SET command” on page 267

## Allowable moves for the z/OS Debugger SET command

The following table shows the allowable moves for the z/OS Debugger SET command.

| Source field | | Receiving field | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | AO | IN | IDI | PTR | ED | BI | ID | OR |
| Address Of (AO) | Y | | | Y | | | | |
| Index Name (IN) | | Y | Y | | Y | Y | Y | |
| Index Data Item (IDI) | | Y | Y | | | | | |
| Pointer Data Item (PTR) | Y | | | Y | | | | |
| Address Hex Literal[1] | Y | | | Y | | | | |
| NULL (NUL) | Y | | | Y | | | | |
| Integer Literal | | Y[2] | | | | | | |
| External Decimal (ED) | | Y | | | | | | |
| Binary (BI) | | Y | | | | | | |
| Internal Decimal (ID) | | Y | | | | | | |
| Object Reference (OR) | | | | | | | | Y |

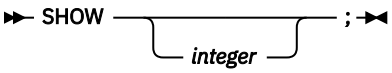
**Notes:**

**1**
Must be hexadecimal characters only, delimited by either quotation marks (") or apostrophes (') 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.




***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
```

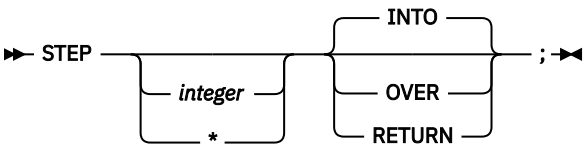
No space is needed as a delimiter between the keyword and the integer; hence, SHOW 3 is equivalent to SHOW3.

## STEP command

The STEP command causes z/OS Debugger 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
- a programming language or Language Environment condition or exception




***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.

**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.

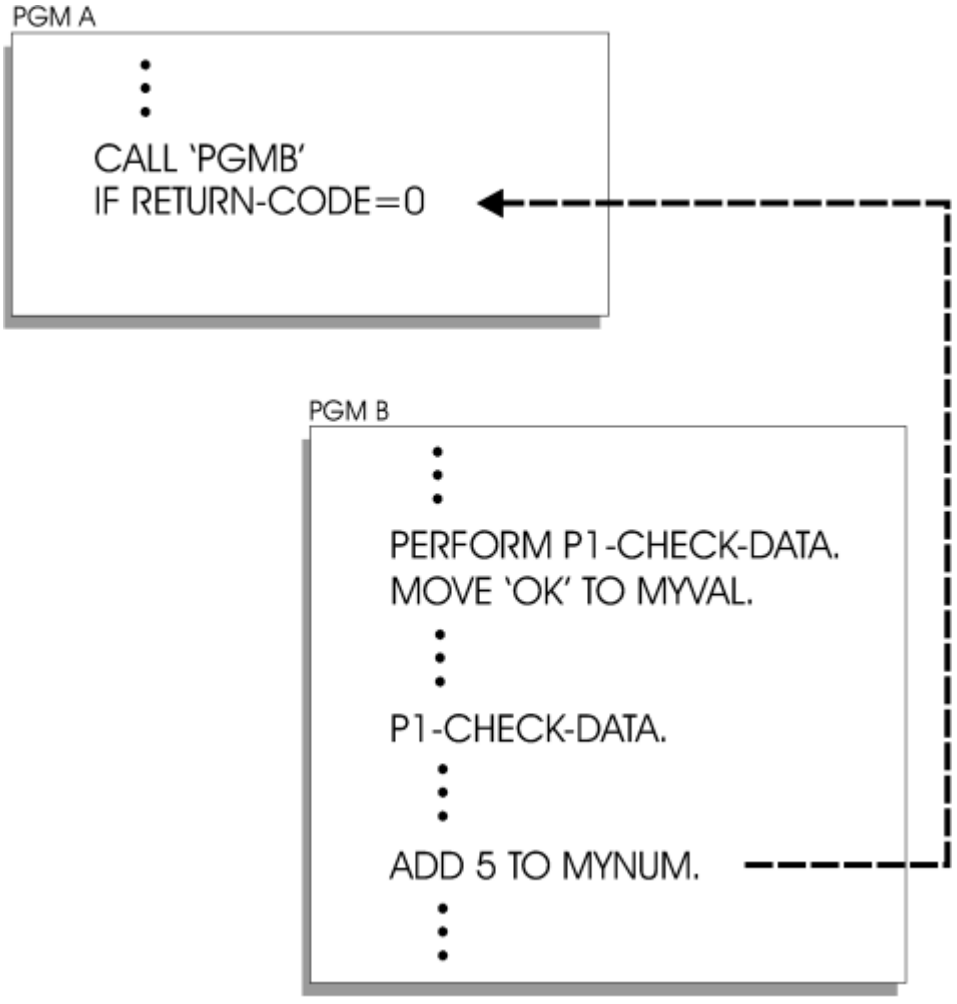
If you are debugging a disassembled program, verify that you have set a breakpoint in the calling program. Without the breakpoint, z/OS Debugger cannot resume STEP mode on return and the application continues to run until it ends.

**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 a LangX COBOL or disassembled program, do not use the STEP RETURN command because z/OS Debugger 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, z/OS Debugger 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, z/OS Debugger cannot stop the application. After you have set the breakpoint, you can resume running your application by entering a z/OS Debugger 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, z/OS Debugger 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.
- You can use the STEP command in remote debug mode by entering it in the Debug Console or the **Action** field, which is in the **Optional Parameters** section of the **Add a Breakpoint** task.
- With Enterprise COBOL V5 (and later), STEP OVER, an out-of-line PERFORM, behaves the same as it does when stepping over a called subroutine. However, you cannot use STEP RETURN from within the out-of-line PERFORM to return to the statement after the PERFORM.

**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.

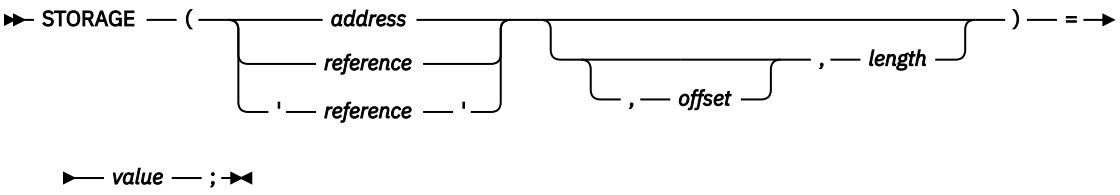
```
STEP 25 OVER;
```

- 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.




***address***
The address of the first byte of storage that you want to alter.

***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*'**
A LangX COBOL variable whose storage location is to be changed. In LangX COBOL, reference must be enclosed in apostrophes (').

***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 same syntax rules as *address* above. The default is 0.

***length***
The decimal number of bytes you want to alter. This must equal the length of *value*.

***value***
The value you want to store. The notation for *value* must be one of the following:

- An address.
- A hexadecimal value surrounded by apostrophes (') and preceded by "X". You can also use a different notation for the following programming languages:
  - For PL/I, the hexadecimal value enclosed in quotation marks (") or apostrophes (') followed by PX.
  - For assembler, COBOL, LangX COBOL, or disassembly, the hexadecimal value enclosed in quotation marks (") and preceded by "X".
- A decimal value. For any decimal value, four bytes are altered. For example, STORAGE `(H'12345678') = 3` is the same as STORAGE`(H'12345678') = H'00000003'`.
- A character string up to 256 bytes long, using the character string notation appropriate for each programming language or, for all programming languages, you can use enclose the string in quotation marks (").

**Usage notes**

- If you specify only two parameters, z/OS Debugger assumes the second parameter is the length.
- If you specify only one parameter, z/OS Debugger assumes the offset is 0 and that the length is equal to the length of *value*.
- The STORAGE command cannot be used while you replay recorded statements by using the PLAYBACK commands.
- If you specify *address* with more than 8 significant digits or if *reference* references 64-bit addressable storage, z/OS Debugger assumes that the storage location is 64-bit addressable storage. Otherwise, z/OS Debugger assumes that the storage location is 31-bit addressable storage.
- If *reference* is a pointer, z/OS Debugger changes the contents at the address given by that pointer.

**Examples**

- For any programming language, enter the following command to alter two bytes of storage at address X'12345678':

```
STORAGE (X'12345678') = 0x1234;
```

- For C, enter the following command to alter two bytes of storage at address X'12345678':

```
STORAGE (0x12345678) = 0x1234;
```

- For COBOL, enter the following command to alter four bytes of storage at address 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 X'12345678':

```
STORAGE (H'12345678') = X'C1C1C1C1C1C1'
```

- For PL/I, enter the following command to alter six bytes of storage at address X'12345678':

```
STORAGE ('12345678'PX) = 'C1C1C1C1C1C1'X
```

- For PL/I enter the following command to alter 23 bytes of storage starting at address X'12345678':

```
STORAGE ('12345678'PX) = 'aaaaaaaaaaaaaaaaaaaaaaa'
```

- Enter the following command to alter 10 bytes of storage at MYVAR, starting at offset 2:

```
STORAGE (MYVAR, 2, 10) = 'new text: ';
```

- Enter the following command to alter 4 bytes of storage at address X'20CD0', starting at offset 10:

```
STORAGE ('20CD0'PX, 10, 4) = 99;
```

- Enter the following command to alter storage at MYVAR, starting at offset 0, for the same number of bytes as the length of variable MYVAR:

```
STORAGE (MYVAR) = 10;
```

- For C, update the storage pointed by an address 1A3BE910, starting at offset -20 for 20 bytes:

```
 STORAGE (0x1A3BE910,-20,20) = 'first and last name ';
```

- Update 20 bytes of storage pointed by an address 162F0, language is COBOL, offset is 0:

```
STORAGE ( H'162F0', 20 ) = 'clear that string ' ;
```

- For Assembler, update the storage pointed by address 00020CD0, starting at offset 16 for 4 bytes, and the offset is specified as a hex constant:

```
 STORAGE ( X'00020CD0', X'10', 4 ) = 5 ;
```

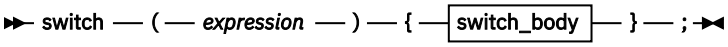
Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

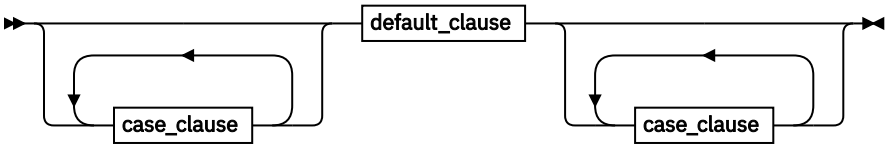
“address” on page 11

## switch command (C and C++)

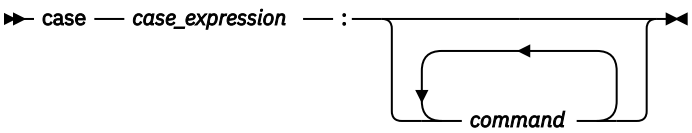
The `switch` command enables you to transfer control to different commands within the `switch` body, depending on the value of the `switch` expression. The `switch`, `case`, and `default` keywords must be lowercase and cannot be abbreviated.



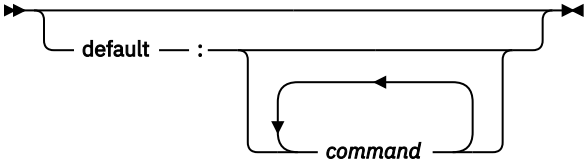

**switch_body**




**case_clause**




**default_clause**




***expression***
A valid z/OS Debugger C expression.

***case_expression***
A valid character or optionally signed integer constant.

***command***
A valid z/OS Debugger 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 z/OS Debugger 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.

```
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'.

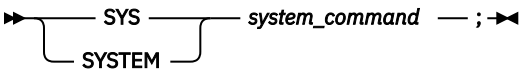
```
char text[100];
int  capa, i, lettera, total;

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 TS0 commands during a z/OS Debugger session. The SYSTEM keyword can only be abbreviated as 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 z/OS Debugger 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 topics for more information related to the material discussed in this topic.

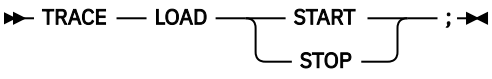
**Related references**

"TSO command (z/OS)" on page 280

# TRACE command

The TRACE command creates a TRACE LOAD table of all load modules and DLLs loaded during a debug session. The TRACE LOAD table can then be listed at any point during the debug session.

The syntax of the TRACE command is as follows:




**START**
Instructs z/OS Debugger to start tracing new load modules and DLLs.

**STOP**
Instructs z/OS Debugger to stop tracing new load modules and DLLs.

**Usage notes**

- When you enter the TRACE LOAD START command, any load module that is already known to z/OS Debugger is added to the TRACE LOAD table.
- You can list the TRACE LOAD table with the command LIST TRACE LOAD.
- The TRACE LOAD table is deleted when you issue TRACE LOAD STOP, and thus a subsequent LIST TRACE LOAD will not return any information.

**Examples**

To instruct z/OS Debugger to start tracing loads of load modules and DLLs, and then list the traced load modules/DLLs and their library names, enter the following commands:

```
TRACE LOAD START;
(Various z/OS Debugger commands that step, go, or run though the code.)
LIST TRACE LOAD;
```

You might get results similar to the following output:

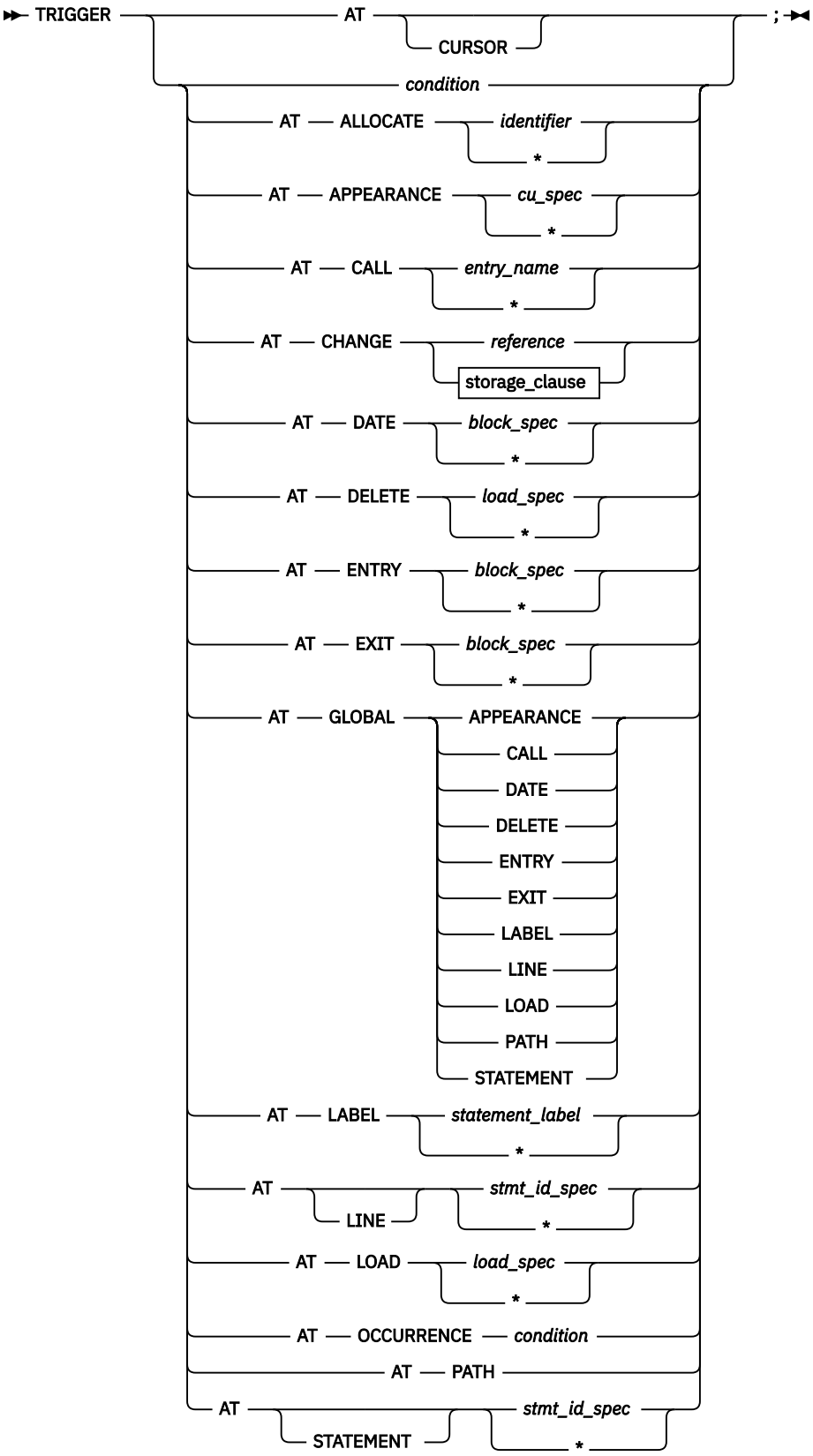
```
The following were loaded.
IBCD010 loaded from TSFANAY.TEST.LOAD
```

**Related references**

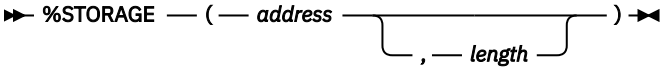
"LIST TRACE LOAD command" on page 165

# TRIGGER command

The TRIGGER command raises the specified AT-condition in z/OS Debugger, or it raises the specified programming language condition in your program.




**storage_clause**



***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.

```
SIGABND                 SIGILL                  SIGTERM
SIGABRT                 SIGINT                  SIGUSR1
SIGFPE                  SIGIOERR                SIGUSR2
                        SIGSEGV
```

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 S*xxx* and U*xxx* 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 z/OS Debugger 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.

***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**

- If the EQAOPTS THREADTERMCOND command prevents z/OS Debugger from stopping when a FINISH, CEE066, or CEE067 thread termination condition is raised by Language Environment, z/OS Debugger

does not gain control when these conditions are raised. If you want z/OS Debugger to gain control when these conditions are raised, you can set an AT OCCURRENCE breakpoint or change the EQAOPTS THREADTERMCOND command to allow z/OS Debugger to gain control.

- 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 z/OS Debugger 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 z/OS Debugger is started for the STATEMENT breakpoint and you want to trigger the commands associated with the AT CHANGE breakpoint, enter:

```
TRIGGER AT CHANGE x;
```

z/OS Debugger displays the value of x.

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
*z/OS Language Environment Programming Guide*

**Related references**
“ON command (PL/I)” on page 181
“address” on page 11
“cu_spec” on page 13
“references” on page 15
“load_spec” on page 15
“block_spec” on page 12
“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 z/OS Debugger session and is valid only in a TSO environment. The TSO keyword cannot be abbreviated.


***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.

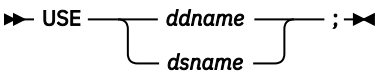
```
TSO LISTCAT;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"SYSTEM command (z/OS)" on page 275

# USE command

The USE command causes the z/OS Debugger 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.




***ddname***
A valid ddname in z/OS.

***dsname***
A z/OS data set containing the z/OS Debugger commands to be performed. If *dsname* is not enclosed in apostrophes ('), z/OS Debugger 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 OFF.
  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 z/OS Debugger 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 z/OS Debugger 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 z/OS Debugger 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, z/OS Debugger 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 z/OS Debugger 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 z/OS Debugger commands in the partitioned data set member `USERID.PDS(CMDS)`.

```
USE PDS(CMDS)
```

- For CICS, perform z/OS Debugger commands in the fully-qualified data set `TS64081.USE.FILE`.

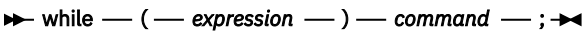
```
USE 'TS64081.USE.FILE';
```

  In addition to using sequential files, you can perform z/OS Debugger 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.




**`expression`**
A valid z/OS Debugger C expression.

**`command`**
A valid z/OS Debugger 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.

```
x = 1;
while (x +=2, x < 10)
  LIST x;
```

- While `--index` is greater than or equal to zero (`0`), triple the value of the expression `item[index]`.

```
while (--index >= 0) {
  item[index] *= 3;
  printf("item[%d] = %d\n", index, item[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.

| Command | Description |
|---|---|
| "WINDOW CLOSE command" on page 282 | Closes the specified window in the z/OS Debugger full-screen session panel. |
| "WINDOW OPEN command" on page 283 | Opens a previously-closed window in the z/OS Debugger full-screen session panel. |
| "WINDOW SIZE command" on page 284 | Controls the relative size of currently visible windows in the z/OS Debugger full-screen session panel. |
| "WINDOW SWAP command" on page 284 | Replaces the logical window being displayed in a physical window with another logical window. |
| "WINDOW ZOOM command" on page 285 | Expands the indicated window to fill the entire screen. |

**Usage notes**

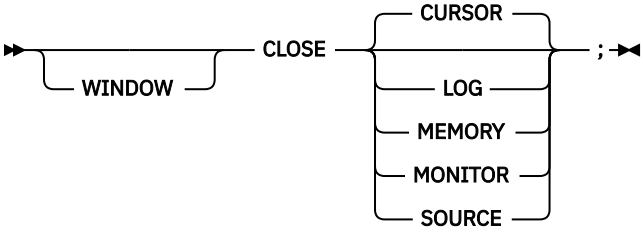
- If no operand is specified and the cursor is on the command line, then the default window id set by `SET DEFAULT WINDOW` is used (if it is open, otherwise the precedence is `SOURCE`, `LOG`, `MONITOR`).

## WINDOW CLOSE command

Closes the physical window of the specified logical window in the z/OS Debugger full-screen session panel. The remaining open physical windows expand to fill the remainder of the screen. Closing a physical window does not effect the logical window. For example, closing the physical window that is displaying the Monitor window does not stop the monitoring of variable values assigned by the `LIST MONITOR` command.

If you specify a logical window that is not assigned to a physical window, z/OS Debugger displays an error message.

If there is only one physical 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.

**MEMORY**
Selects the Memory window.

**MONITOR**
Selects the monitor window.

**SOURCE**
Selects the source listing window.

**Example**

Close the window containing the cursor.

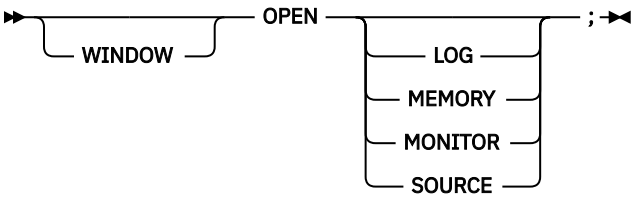
```
WINDOW CLOSE CURSOR;
```

## WINDOW OPEN command

Opens a previously-closed physical window in the z/OS Debugger full-screen session panel. Any existing physical windows are resized according to the configuration selected with the PANEL LAYOUT command.

If you specify a logical window that is not assigned to a physical window, z/OS Debugger displays an error message.

If the OPEN command is issued without an operand, z/OS Debugger opens the last closed physical window.




**LOG**
Selects the session log window.

**MEMORY**
Selects the Memory window.

**MONITOR**
Selects the monitor window.

**SOURCE**
Selects the source listing window.

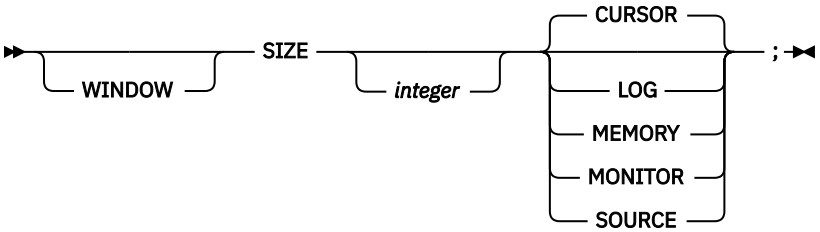
**Example**

Open the monitor window.

```
WINDOW OPEN MONITOR;
```

## WINDOW SIZE command

Controls the relative size of the currently visible physical windows in the z/OS Debugger 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.

**MEMORY**
Selects the Memory 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 SWAP command

The SWAP command replaces the logical window being displayed in a physical window with another logical window. The order of the operands is not important. The physical window retains its attributes. For

example, if the physical window was closed, it remains closed when you entered the SWAP command, it remains closed until you enter the WINDOW OPEN command.


**MEMORY**
Selects the Memory window.

**LOG**
Selects the Log window.

**Examples**

- Replace the Log window, which is currently displayed in a physical window, with the Memory window, which is not being displayed in a physical window by entering the following command:

```
SWAP MEM LOG
```

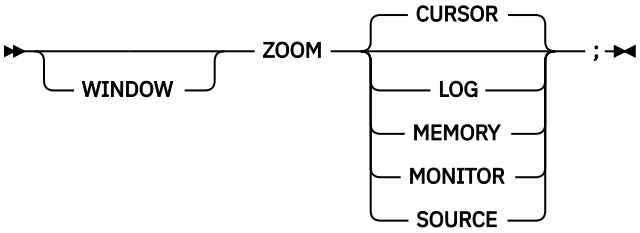
The Memory window assumes the size and location of the physical window.

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
"z/OS Debugger session panel in the IBM z/OS Debugger User's Guide.
"Switching between the Memory window and Log window" in the IBM z/OS Debugger User's Guide

## WINDOW ZOOM command

Expands the specified logical window to fill the entire screen or restores the screen to the currently defined physical window configuration. The logical window does not have to be assigned to a physical window. This command provides a convenient way to display any logical window without having to reassign physical windows. For example, because the MEMORY window and LOG window cannot be displayed at the same time, you can use the WINDOW ZOOM LOG command to display the Log window while the Memory window remains assigned to its physical window.




**CURSOR**
Selects the window where the cursor is currently positioned unless on the command line.

**LOG**
Selects the session log window.

**MEMORY**
Selects the Memory 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. EQAOPTS commands

EQAOPTS commands are commands that alter some of the basic behavior of z/OS Debugger. These commands must be processed before normal z/OS Debugger command processing is available. You can specify most EQAOPTS commands in the following ways:

- Add dynamically at run time, as described in "Providing EQAOPTS commands at run time" on page 294, a text data set that contains the commands.
- Add to the search sequence, before the copy of EQAOPTS distributed by z/OS Debugger, a customized version of the EQAOPTS load module.

If you want the commands to apply to only a few debugging sessions, it might be easier to supply the EQAOPTS command dynamically at run time. If you want the commands to apply to a group of debugging sessions, it might be better to supply the EQAOPTS commands through the EQAOPTS load module.

Except for commands that can be validly specified more than once (for example, the NAMES commands), if z/OS Debugger finds a command more than once, it uses the first specification of the command. z/OS Debugger processes EQAOPTS commands specified at run time before those specified through the EQAOPTS load module. This means that commands specified at run time override duplicate commands specified in the EQAOPTS load module.

Any or all of the following people can create EQAOPTS specifications:

- The system programmer that installs z/OS Debugger.
- Specific groups in the organization.
- An individual user.

If you are the system programmer or you are creating EQAOPTS specifications for specific groups, you might change the EQAOPTS specifications less frequently, so specifying them by generating a new EQAOPTS load module might be more efficient. If you are an individual user, you might change the EQAOPTS specifications more frequently, so specifying them dynamically at run time might be more efficient.

Table 10 on page 287 summarizes the available EQAOPTS commands and indicates whether a system programmer (S), a specific group (G), or an individual user (U) most commonly uses a command.

*Table 10. A brief description of each EQAOPTS command and the type of user most likely to use that command*

| Command | Description | Commonly used by |
|---|---|---|
| ALTDISP | Controls whether to add a character indicator to the MFI screen to indicate a breakpoint, the current line, or the line with found text. | S, U |
| BROWSE | Allows users with the authority to use z/OS Debugger in normal mode to restrict their access to Browse Mode. | U |
| CACHENUM | Controls the size of the z/OS Debugger cache to minimize rereading the debug information. | U, G |
| CCOUTPUTDSN | Specifies the default naming pattern that z/OS Debugger uses to name the Code Coverage Observation file. | U, G, S |
| CCOUTPUTDSNALLOC | Specifies the allocation parameters that z/OS Debugger uses when it creates the Code Coverage Observation file. | U, G, S |
| CCPROGSELECTDSN | Specifies the default naming pattern that z/OS Debugger uses to name the Code Coverage Options file. | U, G, S |

*Table 10. A brief description of each EQAOPTS command and the type of user most likely to use that command (continued)*

| Command | Description | Commonly used by |
|---|---|---|
| CEEREACTAFTERQDBG | Restarts z/OS Debugger if a CEETEST call is encountered after you use `QUIT DEBUG` to end a debug session. | S |
| CODEPAGE | Controls the codepage used by z/OS Debugger. | U, G, S |
| COMMANDSDSN | Specifies the default naming pattern that z/OS Debugger uses to name the user's commands file. | U, G, S |
| DEFAULTVIEW | Controls the default view of assembler programs. | U, G |
| DISABLERLIM | Disables Omegamon resource limiting (RLIM) during debug sessions. | S |
| DLAYDBG | Allows users to use delay debug mode. | U, G, S |
| DLAYDBGCND | Specifies monitoring condition events in the delay debug mode. | U, G, S |
| DLAYDBGDSN | Specifies delay debug profile data set naming pattern. | U, G, S |
| DLAYDBGTRC | Specifies delay debug pattern match trace message level. | U, G, S |
| DLAYDBGXRF | Specifies that z/OS Debugger uses a cross reference to find the user ID when z/OS Debugger constructs the delay debug profile data set name.<br>This is used when an IMS transaction or DB/2 stored procedure is initiated from the web or MQ gateway, and thus the transaction is run with a generic ID.<br>z/OS Debugger uses either the cross reference file or the Terminal Interface Manager repository to find the ID of the user who wants to debug the transaction or stored procedure. | U, G, S |
| DOPTACBDSN | Specifies the data set which will contain DOPT PSBs generated by the IMS Transaction Isolation Facility. | S |
| DTCNDELETEDEADPROF | Controls the deletion of dead DTCN profiles. | S |
| DTCNFORCE*xxxx* | Controls whether to require certain fields in DTCN. | S |
| DYNDEBUG | Controls the initial (default) value of SET DYNDEBUG. | U, G, S |
| EQAQPP | Enables z/OS Debugger to debug MasterCraft Q++ programs. | U, G, S |
| EXPLICITDEBUG | Enables explicit debug mode. | U |
| GPFDSN | Specifies that z/OS Debugger process a global preferences file. | U, G, S |
| HOSTPORTS | Specifies a host port or range of ports to use for a TCP/IP connection to the workstation for the remote debugger. | S |
| IGNOREODOLIMIT | Specifies that z/OS Debugger can display COBOL table data items even when an ODO value is out of range. | U, G, S |
| LOGDSN | Specifies the default naming pattern that z/OS Debugger uses to name the user's log file. | U, G, S |
| LOGDSNALLOC | Specifies the allocation parameters that z/OS Debugger uses when it creates the log file. | U, G, S |

*Table 10. A brief description of each EQAOPTS command and the type of user most likely to use that command (continued)*

| Command | Description | Commonly used by |
|---|---|---|
| MAXTRANUSER | Specifies the maximum number of IMS transactions that a single user may register to debug using the IMS Transaction Isolation Facility. | S |
| MDBG | Allows users of programs compiled with z/OS XL C/C++ Version 1.10, or later, to indicate whether z/OS Debugger searches for .mdbg files. | U, G |
| MULTIPROCESS | Controls the behavior of z/OS Debugger when a new POSIX process is created by fork() or exec(). | U, G, S |
| NAMES | Controls whether z/OS Debugger processes or ignores certain load module or compile unit names. | U, G |
| NODISPLAY | Controls the z/OS Debugger behavior when the display requested by the z/OS Debugger user is not available. | U, G, S |
| PREFERENCESDSN | Specifies the default naming pattern that z/OS Debugger uses to name the preferences file. | U, G, S |
| SAVEBPDSN, SAVESETDSN | Specifies the default naming pattern for the data sets used to save and restore the breakpoints and monitors (SAVEBPS) and the settings (SAVESETS). | U, G, S |
| SAVEBPDSNALLOC, SAVESETDSNALLOC | Specifies the allocation parameters that z/OS Debugger uses when it creates the SAVEBPS and SAVESETS data sets. | U, G, S |
| SESSIONTIMEOUT | Establishes a timeout for idle z/OS Debugger sessions that use the Terminal Interface Manager. Timed out sessions are canceled after a specified period of no user activity. | S |
| STARTSTOPMSG | Controls whether to issue a message when each debugging session is initiated or terminated. | S |
| STARTSTOPMSGDSN | Specifies a message file for start and stop debug session messages. | S |
| SUBSYS | Specifies a subsystem used by certain library systems. | G, S |
| SVCSCREEN | Controls whether and how z/OS Debugger uses SVC screening to intercept LOAD and LINK SVC's. This is necessary for debugging non-Language Environment assembler and LangX COBOL programs. | S |
| TCPIPDATADSN | Instructs z/OS Debugger to dynamically allocate the specified file-name to the DDNAME SYSTCPD for the TCP/IP connection to the workstation for the remote debugger. | S |
| THREADTERMCOND | Controls whether z/OS Debugger prompts the user when it encounters a FINISH, enclave termination, or thread termination condition. | U, G |
| TIMACB | Specifies that the z/OS Debugger Terminal Interface Manager (TIM) use a name other than EQASESSM. | S |
| END | Specifies the end of a list of EQAOPTS commands. You must specify END. | U, G, S |

Use the following list to help you record the commands and value you want to implement:

- EQAXOPT ALTDISP, then select one of the following options:

  ON
  OFF
- EQAXOPT BROWSE, then select one of the following options:

  RACF
  ON
  OFF
- EQAXOPT CACHENUM,*number:_____*
- EQAXOPT CCOUTPUTDSN,'*file_name_pattern:_____*'

  Append ,LOUD if you want z/OS Debugger to display WTO messages, which helps you debug processing done by this command.
- EQAXOPT CCOUTPUTDSNALLOC,*allocation_parameters:_____*

  Append ,LOUD if you want z/OS Debugger to display WTO messages, which helps you debug processing done by this command.
- EQAXOPT CCPROGSELECTDSN,'*file_name_pattern:_____*'

  Append ,LOUD if you want z/OS Debugger to display WTO messages, which helps you debug processing done by this command.
- EQAOPTS CEEREACTAFTERQDBG, then select one of the following options:

  YES
  NO
- EQAXOPT CODEPAGE,*code_page_number:_____*
- EQAXOPT COMMANDSDSN,'*file_name_pattern:_____*'

  Append ,LOUD if you want z/OS Debugger to display WTO messages, which helps you debug processing done by this command.
- EQAXOPT DEFAULTVIEW, then select one of the following options:

  STANDARD
  NOMACGEN
- EQAXOPT DISABLERLIM, then select one of the following options:

  YES
  NO
- EQAXOPT DLAYDBG, then select one of the following options:

  YES
  NO
- EQAXOPT DLAYDBGCND, then select one of the following options:

  YES
  NO
- EQAXOPT DLAYDBGDSN,'*file_name_pattern:_____*'
- EQAXOPT DLAYDBGTRC,*pattern_match_trace_level:_____*
- EQAXOPT DLAYDBGXRF, then select one of the following options:

  DSN,'*file_name:_____*'
  REPOSITORY
- EQAXOPT DOPTACBDSN,'*file_name:_____*'
- EQAXOPT DTCNDELETEDEADPROF, then select one of the following options:

  YES

NO

- EQAXOPT DTCNFORCECUID, then select one of the following options:

  YES
  NO

  This option performs the same function as DTCNFORCEPROGID. If you select YES for DTCNFORCEPROGID, you do not need to specify this option.
- EQAXOPT DTCNFORCEIP, then select one of the following options:

  YES
  NO
- EQAXOPT DTCNFORCELOADMODID, then select one of the following options:

  YES
  NO
- EQAXOPT DTCNFORCENETNAME, then select one of the following options:

  YES
  NO
- EQAXOPT DTCNFORCEPROGID, then select one of the following options:

  YES
  NO
- EQAXOPT DTCNFORCETERMID, then select one of the following options:

  YES
  NO
- EQAXOPT DTCNFORCETRANID, then select one of the following options:

  YES
  NO
- EQAXOPT DTCNFORCEUSERID, then select one of the following options:

  YES
  NO
- EQAXOPT DYNDEBUG, then select one of the following options:

  ON
  OFF
- EQAXOPT EQAQPP, then select one of the following options:

  ON
  OFF
- EQAXOPT EXPLICITDEBUG, then select one of the following options:

  ON
  OFF
- EQAXOPT GPFDSN,'*file_name:*_____'
- EQAXOPT HOSTPORTS,*range_of_ports:*_____
- EQAXOPT IGNOREODOLIMIT, then select one of the following options:

  YES
  NO
- EQAXOPT LOGDSN,'*file_name_pattern:*_____'

  Append ,LOUD if you want z/OS Debugger to display WTO messages, which helps you debug processing done by this command.

- EQAXOPT LOGDSNALLOC,*allocation_parameters:*_____

  Append ,LOUD if you want z/OS Debugger to display WTO messages, which helps you debug processing done by this command.
- EQAXOPT MAXTRANUSER,*number:*_____
- EQAXOPT MDBG, then select one of the following options:

  YES
  NO
- EQAXOPT MULTIPROCESS, then select one of the following options:

  PARENT
  CHILD
  PROMPT

  Select one of the following options to indicate what you want z/OS Debugger to do with a process that executes itself:

  EXEC=ANY
  EXEC=NONE
- EQAXOPT NAMES, then select one of the following options:

  EXCLUDE,LOADMOD,*pattern:*_____
  EXCLUDE,CU,*pattern:*_____
  INCLUDE,LOADMOD,*name:*_____
  INCLUDE,CU,*name:*_____
- EQAXOPT NODISPLAY, then select one of the following options:

  DEFAULT
  QUITDEBUG
- EQAXOPT PREFERENCESDSN,'*file_name_pattern:*_____'

  Append ,LOUD if you want z/OS Debugger to display WTO messages, which helps you debug processing done by this command.
- EQAXOPT SAVEBPDSN,'*file_name_pattern:*_____'
- EQAXOPT SAVESETDSN,'*file_name_pattern:*_____'
- EQAXOPT SAVEBPDSNALLOC,*allocation_parameters:*_____

  Append ,LOUD if you want z/OS Debugger to display WTO messages, which helps you debug processing done by this command.
- EQAXOPT SAVESETDSNALLOC,*allocation_parameters:*_____

  Append ,LOUD if you want z/OS Debugger to display WTO messages, which helps you debug processing done by this command.
- EQAXOPT SESSIONTIMEOUT, then select one of the following options:

  NEVER
  QUITDEBUG,*hhmmssnn*
  QUIT,*hhmmssnn*
- EQAXOPT STARTSTOPMSG, then select one of the following options:

  NONE
  ALL
  CICS
  TSO
  BATCHTSO
  IMS
  OTHER

or any of CICS, TSO, BATCHTSO, IMS, and OTHER, or all of them enclosed in parenthesis and separated by commas.

Append ,WTO if you want z/OS Debugger to display the messages via WTO.

- EQAXOPT STARTSTOPMSGDSN,'*file_name:*_____'

  Append ,LOUD if you want z/OS Debugger to display WTO messages, which helps you debug processing done by this command.
- EQAXOPT SUBSYS,*subsystem_name:*_____
- EQAXOPT SVCSCREEN, then select one of the following options:

  ON
  OFF
  (OFF,QUIET)

  Select one of the following options to indicate what you want z/OS Debugger to do if there is an existing SVC screening environment:

  CONFLICT=OVERRIDE
  CONFLICT=NOOVERIDE

  Select one of the following options to indicate whether you want z/OS Debugger to temporarily replace the existing SVC screening environment:

  NOMERGE
  MERGE=(COPE)
- TCPIPDATADSN,'*file_name:* _____'
- EQAXOPT THREADTERMCOND, then select one of the following options:

  PROMPT
  NOPROMPT
- EQAXOPT TIMACB,*ACB_name:*_____
- EQAXOPT END Always specify this command.

After you have made all of you selections, define the options as described in "Creating EQAOPTS load module" on page 295.

## Format of the EQAOPTS command

When you specify EQAOPTS commands through the EQAOPTS load module, you create them as assembler macro invocations and you must subsequently assemble and link-edit them into the EQAOPTS load module. To provide a consistent format for all forms of EQAOPTS commands, when you specify the EQAOPTS commands at run time, you must use the assembler macro invocation format. The following format rules apply to all EQAOPTS commands:

- EQAOPTS commands must be contained in fixed-length, eighty-byte records.
- The commands must be contained between columns one and seventy-one, with column seventy-two reserved for a continuation indicator. z/OS Debugger ignores columns seventy-three through eighty.
- Specify an asterisk (*) in column one to indicate a comment. z/OS Debugger ignores comments. Column one must be blank for all non-comment statements.
- The op-code for each EQAOPTS statement must be EQAXOPT and must begin in or after column two and followed by at least one blank.
- A list of one or more operands must follow the EQAXOPT op-code. Separate these operands by a comma and do not embed blanks.
- If a command exceeds the length of one line, you can continue the command in one of the following ways:

- You can end at the comma following an operand and place a non-blank character in column seventy-two.
- You can use all of the columns through column seventy-one and place a non-blank character in column seventy-two.

In either case, the statement that follows must be blank in columns one through fifteen and begin in column sixteen.

## EQAOPTS commands that have equivalent z/OS Debugger commands

Some EQAOPTS commands have equivalent z/OS Debugger commands. Table 11 on page 294 shows a few examples.

*Table 11. Examples of EQAOPTS commands and their equivalent z/OS Debugger commands*

| **EQAOPTS command** | **z/OS Debugger command** |
|---|---|
| DEFAULTVIEW | SET DEFAULTVIEW |
| DYNDEBUG | SET DYNDEBUG |
| EXPLICITDEBUG | SET EXPLICITDEBUG |
| NAMES | NAMES |

For these commands, specifying them as EQAOPTS commands or z/OS Debugger commands produces the same action. The timing (when these commands take effect) differs between EQAOPTS commands and z/OS Debugger commands.

z/OS Debugger processes z/OS Debugger commands after it processes the initial load module and creates the compile units contained in the initial load modules. z/OS Debugger processes EQAOPTS commands during z/OS Debugger initialization, prior to processing the initial load module. This means that when z/OS Debugger processes the initial load module, z/OS Debugger commands like NAMES are not in effect but the corresponding EQAOPTS commands are in effect and are applied to the initial load module.

EQAOPTS commands like DEFAULTVIEW provide a way of specifying a site- or group-wide default for the corresponding z/OS Debugger command. However, a better way to specify a site- or group-wide default for these types of commands is by putting the z/OS Debugger command in a global preferences file.

## Providing EQAOPTS commands at run time

You can provide EQAOPTS commands to z/OS Debugger at run time. You must save the commands in a data set with 80-byte, fixed-length records. The following list describes the methods of specifying this data set to z/OS Debugger:

- In CICS, include the EQAOPTS commands through DTCN.
- In UNIX System Services, specify the name of the data set containing the EQAOPTS commands through the EQA_OPTS_DSN environment variable.
- In IMS and Db2, specify the name of the data set containing the EQAOPTS commands through the z/OS Debugger Language Environment user exit.
- In other environments, specify the name of the data set containing the commands through the EQAOPTS DD statement.

The following example shows what the data set might contain:

```
EQAXOPT MDBG,YES
EQAXOPT NODISPLAY,QUITDEBUG
EQAXOPT NAMES,EXCLUDE,LOADMOD,USERMOD1
EQAXOPT NAMES,EXCLUDE,LOADMOD,USERMOD7
EQAXOPT END
```

The instructions in “Creating EQAOPTS load module” on page 295 contain examples with specifications for CSECT, AMODE, RMODE, and END (without EQAXOPTS) statements. Do not include these specifications if you provide EQAOPTS command at run time.

# Creating EQAOPTS load module

If you have chosen to use the EQAOPTS load module to specify your EQAOPTS commands, do the following steps:

1. Copy the EQAOPTS[2] member from the *hlq*.SEQASAMP library to a private library.
2. Edit this copy of EQAOPTS and code the EQAOPTS command or commands you want. To this minimum source, add each EQAXOPT option you want to include. The following example describes the minimum assembler source required to generate the EQAOPTS load module:

```
EQAOPTS  CSECT ,
EQAOPTS  AMODE 31
EQAOPTS  RMODE ANY
         Add your customized EQAXOPT statements here.  For example:
         EQAXOPT MDBG,YES
         EQAXOPT NODISPLAY,QUITDEBUG
         EQAXOPT NAMES,EXCLUDE,LOADMOD,USERMOD1
         EQAXOPT NAMES,EXCLUDE,LOADMOD,USERMOD7
         EQAXOPT END
         END ,
```

   **Note:** You can specify AMODE 64. However, AMODE 64 does not support Delay Debug for 64-bit applications. For more information, see Appendix C, “Limitations of 64-bit support in Debug Tool compatibility mode,” on page 527.
3. Follow the directions in the EQAOPTS sample to generate a new EQAOPTS load module. These directions describe how to assemble the source and link-edit the generated object into a load module named EQAOPTS.
4. Place the EQAOPTS load module in a private data set that is in the load module search path and appears before *hlq*.SEQAMOD.

# Descriptions of EQAOPTS commands

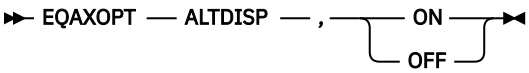
To learn how EQAOPTS commands work and how to specify them, see Chapter 6, “EQAOPTS commands,” on page 287.

## ALTDISP

You can use the EQAOPTS ALTDISP command to add a character indicator to the MFI screen to indicate a breakpoint, the current line, or the line with found text. By default, z/OS Debugger uses coloring to indicate these situations.

Use this command only if your 3270 color configuration and attributes make it difficult to detect the coloring in the line. It is valid only when you are using interactive MFI mode.

The following diagram describes the syntax of the ALTDISP command:




The following list describes the parameters of the EQAOPTS ALTDISP command:

**ON**
Indicates to add a character indicator to indicate a breakpoint, the current line, or the line with found text.

[2] USERMOD EQAUMODE is provided for updating EQAOPTS. See "SMP/E USERMODs" in the *IBM z/OS Debugger Customization Guide* for an SMP/E USERMOD for this customization.

**OFF**
Indicates not to add a character indicator to indicate a breakpoint, the current line, or the line with found text. This is the default value.

**Example**

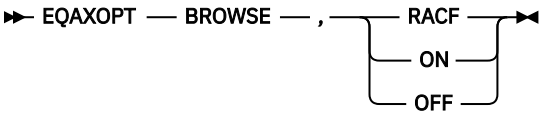
```
EQAXOPT ALTDISP,ON
```

# BROWSE

z/OS Debugger browse mode can be controlled by either the browse mode RACF® facility, through the EQAOPTS BROWSE command, or both. For a description of how to control browse mode through RACF, see "Debugging in browse mode" in the IBM z/OS Debugger User's Guide.

Users who have sufficient RACF authority can specify the EQAOPTS BROWSE command to indicate that the current invocation of z/OS Debugger be in browse mode.

The following diagram describes the syntax of the BROWSE command:




The following list describes the parameters of the EQAOPTS BROWSE command:

**RACF**
Indicates that you want z/OS Debugger to use the browse mode access as determined by the current user's RACF access to the applicable RACF profile. If you do not specify the BROWSE command, z/OS Debugger defaults to RACF.

**ON**
Indicates that unless the user's RACF access is NONE, set BROWSE MODE to ON.

**OFF**
Indicates that if no RACF profile exists or if the user has UPDATE access or higher, set BROWSE MODE to OFF.

**Examples**

```
EQAXOPT BROWSE,ON
EQAXOPT BROWSE,RACF
```

# CACHENUM

To reduce CPU consumption, z/OS Debugger stores information about the application programs being debugged in a cache. By default, for each debug session, z/OS Debugger stores the information for a maximum of 10 programs. Application programs that do a LINK, LOAD, or XCTL to more than 10 programs can degrade z/OS Debugger's CPU performance. You can enhance the CPU performance of z/OS Debugger for these application programs by specifying an increased CACHENUM value in EQAOPTS. An increased value causes z/OS Debugger to use more storage for each debugging session.

The following diagram describes the syntax of the CACHENUM command:


***cache_value***
Specifies the size of the z/OS Debugger cache. It must be no smaller than 10 and no larger then 999.

**Example**

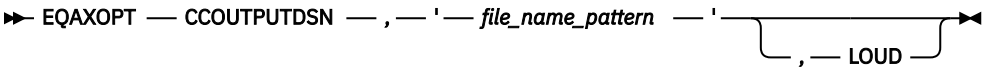
```
EQAXOPT CACHENUM,40
```

## CCOUTPUTDSN

This option provides the data set name to be used for the Code Coverage Observation file. Specify NULLFILE if no Observation file is to be written to.

This data set must be preallocated as a sequential data set if CCOUTPUTDSNALLOC is not specified. RECFM=VB, LRECL=255 is suggested.

The following diagram describes the syntax of the CCOUTPUTDSN command:




***file_name_pattern***
Specifies a naming pattern that determines the name of the data set that contains this file. Follow these guidelines when you create the naming pattern:

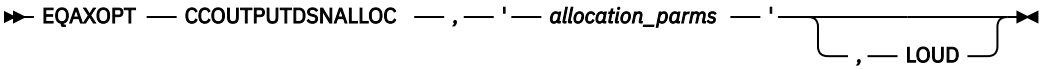
- Create a data set name that includes &&USERID. as one of the qualifiers. z/OS Debugger substitutes the user ID of the current user for this qualifier when it determines the name of the data set.
- Specify NULLFILE to indicate that you do not want z/OS Debugger to process this file.

**LOUD**
Specifies that z/OS Debugger displays WTO messages, which helps you debug processing done by this command. z/OS Debugger normally does not display any messages if it does not find the data set. If you are trying to determine why z/OS Debugger is not processing this file, specify LOUD to see if it displays a message that it can not find the data set.

## CCOUTPUTDSNALLOC

This option is used to create the CCOUTPUTDSN data set for a new user and provides the allocation parameters (in BPXWDYN format).

The following diagram describes the syntax of the CCOUTPUTDSNALLOC command:




***allocation_parms***
Specifies the allocation parameters you want z/OS Debugger to use when it creates the data set. You can specify only the keys in the following list:

- BLKSIZE
- BLOCK
- CYL
- DATACLAS
- DSNTYPE
- DSORG
- LRECL
- MGMTCLAS
- RECFM
- SPACE
- STORCLAS
- TRACKS
- UNIT
- VOL

Separate the keys by one or more blanks. z/OS Debugger does not provide defaults for any of the keys.

For information about the format of the keys, see the chapter "BPXWDYN: a text interface to dynamic allocation and dynamic output" in the *z/OS Using REXX and z/OS UNIX System Services* manual. Specify that the data set be sequential. To learn about other formatting rules for the log file, see "Data sets used by z/OS Debugger" of the IBM z/OS Debugger User's Guide.

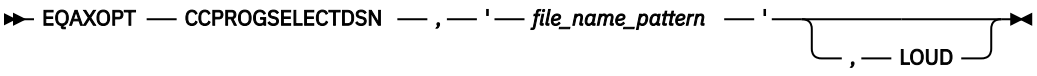
**LOUD**
Specifies that z/OS Debugger displays WTO messages, which helps you debug processing done by this command. z/OS Debugger normally does not display any messages when it creates this data set. If you are trying to determine why this file was not created, specify LOUD to view any messages.

## CCPROGSELECTDSN

This option provides the data set name that contains the Code Coverage Options file (which specifies the Group IDs and the PROGRAM IDs of the COBOL routines that are to be processed). Specify `NULLFILE` if no Code Coverage Options file is to be read.

This data set must be preallocated as a sequential data set. `RECFM=VB, LRECL=255` is suggested.

The following diagram describes the syntax of the `CCPROGSELECTDSN` command:




***file_name_pattern***
Specifies a naming pattern that determines the name of the data set that contains this file. Follow these guidelines when you create the naming pattern:

- Create a data set name that includes `&&USERID.` as one of the qualifiers. z/OS Debugger substitutes the user ID of the current user for this qualifier when it determines the name of the data set.
- Specify `NULLFILE` to indicate you do not want z/OS Debugger to process this file.

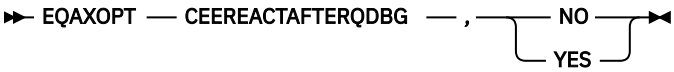
**LOUD**
Specifies that z/OS Debugger displays WTO messages, which helps you debug processing done by this command. z/OS Debugger normally does not display any messages if it does not find the data set or data set member. If you are trying to determine why z/OS Debugger is not processing this file, specify LOUD to see if it displays a message that it can not find the data set.

## CEEREACTAFTERQDBG

You can specify this command to restart z/OS Debugger with `CEETEST` after you use `QUIT DEBUG` to end a debug session. You can specify this command only in an EQAOPTS load module.

**Note:** You cannot use this command in standard mode for remote debugging.

The following diagram describes the syntax of the `CEEREACTAFTERQDBG` command:




The following list describes the parameters of the EQAOPTS `CEEREACTAFTERQDBG` command:

**NO**
Indicates that you do not want to restart z/OS Debugger if a `CEETEST` call is encountered after you use `QUIT DEBUG` to end a debug session. This parameter is the default setting.

**YES**
Indicates that you want to restart z/OS Debugger if a `CEETEST` call is encountered after you use `QUIT DEBUG` to end a debug session.

**Example**

```
EQAXOPT CEEREACTAFTERQDBG,NO
```

```
EQAXOPT CEEREACTAFTERQDBG,YES
```

# CODEPAGE

The default code page used by z/OS Debugger and the remote debuggers is 037. For any of the following situations, you need to use a different code page:

- Application programmers are debugging in remote debug mode and the source or compiler use a code page other than 037.

  If your C/C++ source contains square brackets or other special characters, you might need to specify an EQAOPTS CODEPAGE command to override the z/OS Debugger default code page (037). Check the code page specified when you compiled your source. The C/C++ compiler uses a default code page of 1047 if you do not explicitly specify one. If the code page used is 1047 or a code page other than 037, you need to specify an EQAOPTS CODEPAGE command specifying that code page.
- Application programmers are debugging in full screen mode and encounter one of the following situations:
  - They use the STORAGE command to update COBOL NATIONAL variables.
  - The source is coded in a code page other than 037.
- Application programmers use the XML(CODEPAGE(ccsid)) parameter on a LIST CONTAINER or LIST STORAGE command to specify an alternate code page.

z/OS Debugger uses the z/OS Unicode Services to process characters that need code page conversion.

The following diagram describes the syntax of the CODEPAGE command:


***nnnn***
A positive integer indicating the code page to use.

After implementing the EQAOPTS CODEPAGE command, if application programmers using full-screen mode still cannot display some characters correctly, have them verify that their emulator's code page matches the code page of the characters they need to display.

You might need to create your own conversion images as described in “Creating a conversion image for z/OS Debugger” on page 299.

**Example**

```
EQAXOPT CODEPAGE,121
```

## Creating a conversion image for z/OS Debugger

You might need to create a conversion image so that z/OS Debugger can properly transmit characters in a code page other than 037 between the remote debugger and the host. A conversion image contains the following information:

- The conversion table that specifies the source CCSID (Coded Character Set Identifiers) and target CCSID. For z/OS Debugger, specify a pair of conversion images between the host code page and Unicode code page (UTF-8). You can specify the host code page in the VADSCP*nnnnn* suboption of TEST runtime option or with the CODEPAGE command in the EQAOPTS data set. If you specify both the VADSCP*nnnnn* suboption and the CODEPAGE command, z/OS Debugger uses only the CODEPAGE command. The following table shows the images required for CCSIDs 930, 939 (Japanese EBCDIC), 933 (Korean EBCDIC), 1141 (Germany EBCDIC), and 1047 (Latin 1/Open Systems, EBCDIC). See *IBM z/OS Debugger Reference and Messages* for a detailed description of the suboption VADSCP*nnnnn*.

| *Table 12. Source and target CCSID to specify, depending on the code page command used* | | |
|---|---|---|
| **VADSCP*nnnn* suboption or CODEPAGE command** | **Source CCSID** | **Target CCSID** |
| VADSCP930 or CODEPAGE,930 | 1390[1] | 1208 (UTF-8) |
| | 1208 | 1390[1] |
| VADSCP939 or CODEPAGE,939 | 1399[1] | 1208 (UTF-8) |
| | 1208 | 1399[1] |
| VADSCP933 or CODEPAGE,933 | 933 | 1208 (UTF-8) |
| | 1208 | 933 |
| VADSCP1141 or CODEPAGE,1141 | 1141 | 1208 (UTF-8) |
| | 1208 | 1141 |
| VADSCP1047 or CODEPAGE,1047 | 1047 | 1208 (UTF-8) |
| | 1208 | 1047 |
| **Note:**<br>1. For compatibility with earlier versions, 1390 and 1399 are used. | | |

For each suboption, a pair of conversion images are needed for bidirectional conversion.

- The conversion technique, also called the technique search order. z/OS Debugger uses the technique search order RECLM, which means roundtrip, enforced subset, customized, Language Environment-behavior, and modified language. RECLM is the default technique search order, so you do not have to specify the technique search order in the JCL.

You might need to create a conversion image so that users debugging COBOL programs in full screen or batch mode can modify NATIONAL variables with the STORAGE command or to properly display C/C++ variables that contain characters in a code page other than 037. To create the conversion image, you need to do the following steps:

1. Ask your system programmer for the host's CCSID.
2. Submit a JCL job that specifies the conversion image between the host CCSID, which you obtained in step “1” on page 300, and CCSID 1200 (UTF-16).

“Example: JCL for generating conversion images” on page 300 describes how one JCL creates the conversion images for both situations.

## Example: JCL for generating conversion images

The following JCL generates the conversions images required for z/OS Debugger.

This JCL is a variation of the JCL located at *hlq*.SCUNJCL(CUNJIUTL), which is provided by the Unicode conversion services package.

```
//CUNMIUTL EXEC PGM=CUNMIUTL
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIMG DD DSN=UNI.IMAGES(CUNIMG01),DISP=SHR
//TABIN DD DSN=UNI.SCUNTBL,DISP=SHR
//SYSIN DD *
  /*****************************************************************/
  /* Conversion image input for z/OS Debugger in Remote            */
  /* debug mode                                                    */
  /*****************************************************************/
  CONVERSION 1390,1208;   /* IBM-930 to UTF-8,RECLM */
  CONVERSION 1208,1390;   /* UTF-8 to IBM-930,RECLM */
  CONVERSION 1399,1208;   /* IBM-939 to UTF-8,RECLM */
  CONVERSION 1208,1399;   /* UTF-8 to IBM-939,RECLM */
```

```
  CONVERSION  933,1208;   /* IBM-933 to UTF-8,RECLM */
  CONVERSION 1208,933;    /* UTF-8 to IBM-933,RECLM */
  CONVERSION 1141,1208;   /* IBM-1141 to UTF-8,RECLM */
  CONVERSION 1208,1141;   /* UTF-8 to IBM-1141,RECLM */
  CONVERSION 1047,1208;   /* IBM-1047 to UTF-8,RECLM */
  CONVERSION 1208,1141;   /* UTF-8 to IBM-1141,RECLM */
  /*****************************************************************/
  /* Conversion image input for z/OS Debugger to modify COBOL NATIONAL   */
  /* variables with the STORAGE command while in full screen mode     */
  /*****************************************************************/
  CONVERSION 0037,1200;   /*IBM-37 to UTF-16,RECLM */
/*
```

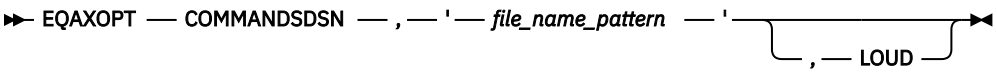
z/OS Debugger uses the character conversion services but not the case conversion or the normalization services of Unicode conversion services. You do not need to include CASE or NORMALIZE control statements unless other applications require them.

## COMMANDSDSN

Indicates that you want z/OS Debugger to read a user's commands file (with the name of the data set containing the commands file determined by the specified naming pattern) each time it starts. This works in the following situation:

- You do not specify a data set name or DD name for the user's commands file using any other method; for example, the TEST runtime option.
- You or your site specifies the EQAOPTS COMMANDSDSN command.
- The data set specified by the EQAOPTS COMMANDSDSN command exists and contains a member whose name matches the initial load module name in the first enclave.

The following diagram describes the syntax of the COMMANDSDSN command:




***file_name_pattern***
Specifies a naming pattern that determines the name of the data set that contains the user's commands file. Follow these guidelines when you create the naming pattern:

- Create a data set name that includes &&USERID. as one of the qualifiers. z/OS Debugger substitutes the user ID of the current user for this qualifier when it determines the name of the data set.
- Specify NULLFILE to indicate you do not want z/OS Debugger to process a commands file.

**LOUD**
Specifies that z/OS Debugger display WTO messages, which helps you debug processing done by this command. z/OS Debugger normally does not display any messages if it does not find the data set or data set member. If you are trying to determine why z/OS Debugger is not processing a user's commands file, specify LOUD to see if it displays a message that it cannot find the data set or the member.

If you choose to implement this option, users who want to use this function must create the commands file as a PDS or PDSE with the allocation parameters that are described in "Data sets used by z/OS Debugger" in the *IBM z/OS Debugger User's Guide*. Then, users create a member for each program that they want to debug, with the name of the member matching the initial load module name in the first enclave.

**Example**

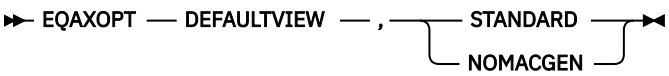
```
EQAOPT  COMMANDSDSN,'&&USERID.DBGTOOL.COMMANDS'
```

If you log in with user ID jsmith, z/OS Debugger determines the name of the data set to be JSMITH.DBGTOOL.COMMMANDS.

## DEFAULTVIEW

A user can control whether to display the statements of an assembler macro in the Source window by entering the SET DEFAULT VIEW command. Every time a LOADDEBUGDATA command is run for an assembler compile unit, z/OS Debugger uses the setting of this command to determine whether to display the macro-generated statements. You can control the initial default for this setting by using the EQAOPTS DEFAULTVIEW command.

The following diagram describes the syntax of the DEFAULTVIEW command:




Each of these fields corresponds to the similar field in the SET DEFAULT VIEW command. If you do not code the EQAOPTS DEFAULTVIEW command, the initial setting for DEFAULTVIEW is STANDARD.

**Example**

```
EQAXOPT DEFAULTVIEW,NOMACGEN
```

## DISABLERLIM

You can specify this command only in the EQAOPTS load module. It cannot be specified at run time.

You can specify this command to control whether or not z/OS Debugger disables Omegamon Resource Limiting (RLIM) during debug sessions in a CICS region.

The following diagram describes the syntax of the DLAYDBG command:


The following list describes the parameters of the DISABLERLIM command:

**YES**
Indicates that you want z/OS Debugger to disable RLIM processing during debug sessions; this is the default value.

**NO**
Indicates that you do not want z/OS Debugger to disable RLIM processing.

**Example**

```
EQAXOPT DISABLERLIM,YES
EQAXOPT DISABLERLIM,NO
```

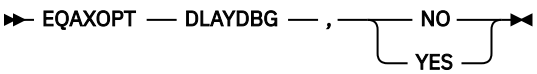
## DLAYDBG

You can specify this command only in the EQAOPTS load module. The only exception to this rule is a transaction executing in a private message region under the IMS Transaction Isolation Facility, which accepts this command in EQAOPTS DD as well.

You can specify this command to enable z/OS Debugger to delay the starting of a debug session until z/OS Debugger recognizes a certain program name or C function name (compile unit) (along with an optional load module name).

This command can be used only in the non-CICS environments.

The following diagram describes the syntax of the DLAYDBG command:



The following list describes the parameters of the DLAYDBG command:

**NO**
Indicates that you do not want delay debug enabled; this is the default value.

**YES**
Indicates that you want delay debug enabled.

If you choose to implement this option, users who want to use this option must create the delay debug profile as a physical sequential data set by using the option B of the IBM z/OS Debugger Utilities: Delay Debug Profile.

**Usage notes**

- This command does not support 64-bit programs.

**Example**

```
EQAXOPT DLAYDBG,NO
EQAXOPT DLAYDBG,YES
```

## DLAYDBGCND

You can specify this command only in the EQAOPTS load module. It cannot be specified at run time.

You can use this command to indicate whether you want z/OS Debugger to monitor condition events in the delay debug mode.

This command can be used only in the non-CICS environments.

The following diagram describes the syntax of the DLAYDBGCND command:


The following list describes the parameters of the DLAYDBGCND command:

**ALL**
Indicates that you want z/OS Debugger to monitor all condition events. This is the default option.

**NONE**
Indicates that you do not want z/OS Debugger to monitor condition events.

**Usage notes**

- This command does not support 64-bit programs.

**Example**

```
EQAXOPT DLAYDBGCND,NONE
```

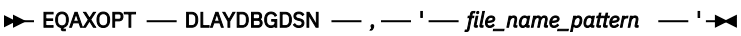
## DLAYDBGDSN

You can specify this command only in the EQAOPTS load module. The only exception to this rule is a transaction executing in a private message region under the IMS Transaction Isolation Facility, which accepts this command in EQAOPTS DD as well.

You can specify this command to indicate that you want z/OS Debugger to use the specified naming pattern when it constructs the delay debug profile data set name.

This command can be used only in the non-CICS environments.

The following diagram describes the syntax of the DLAYDBGDSN command:



***file_name_pattern***
Specifies a naming pattern that determines the name of the data set that contains the delay debug profile. Follow this guideline when you create the naming pattern:

- Create a data set name that includes `&&USERID.` as one of the qualifiers. z/OS Debugger substitutes the user ID of the current user for this qualifier when it determines the name of the data set.

The default naming pattern is `&&USERID.DLAYDBG.EQAUOPTS`.

**Usage notes**

- This command does not support 64-bit programs.

**Example**

```
EQAXOPT DLAYDBGDSN,'&&USERID.DLAYDBG.EQAUOPTS';
```

## DLAYDBGTRC

You can specify this command only in the EQAOPTS load module. The only exception to this rule is a transaction executing in a private message region under the IMS Transaction Isolation Facility, which accepts this command in EQAOPTS DD as well.

You can specify this command to indicate that you want z/OS Debugger to generate trace during the pattern match process in the delay debug mode. z/OS Debugger uses the WTO (write to operator) command to output the trace.

This command can be used only in the non-CICS environments.

The following diagram describes the syntax of the DLAYDBGTRC command:


***trace_level***
Specifies a trace level that determines the level of traces that z/OS Debugger generates. Valid levels are:

- 0 - no trace message; this is the default value.
- 1 - error and warning messages
- 2 - error, warning, and diagnostic messages
- 3 - error, warning, diagnostic, and internal diagnostic messages

**Usage notes**

- This command does not support 64-bit programs.

**Example**

```
EQAXOPT DLAYDBGTRC,2
```

## DLAYDBGXRF

You can specify this command only in the EQAOPTS load module. The only exception to this rule is a transaction executing in a private message region under the IMS Transaction Isolation Facility, which accepts this command in EQAOPTS DD as well.

In this section, the term generic ID refers to a user ID that executes a task but is not the ID that debugs the task. Common examples include the user ID associated with a WebSphere® MQ for z/OS trigger monitor, or the user ID that runs a web services-initiated program.

You can use the DLAYDBGXRF command to map a generic ID to a user ID that is obtained from one of the following sources:

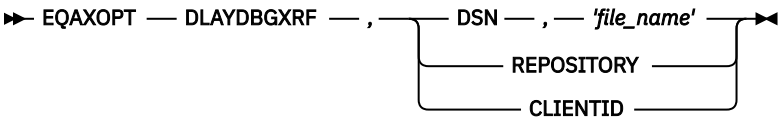
- The Delay Debug cross reference file
- Users logged on to Terminal Interface Manager

The user ID that is obtained by this method is used in place of the current user ID when the delay debug profile data set name is constructed.

The generic ID-to-user ID mapping can occur in the following environments:

- In the IMS environment when an IMS transaction is started with a generic ID.
- In the DB/2 stored procedures environment. This environment is supported by the REPOSITORY and CLIENTID options.

The following diagram describes the syntax of the DLAYDBGXRF command:




**DSN**
Specifies that the generic ID cross reference file *'file_name'* should be used to map the generic ID to the user ID of the z/OS Debugger user.

**REPOSITORY**
Specifies that z/OS Debugger communicates with Terminal Interface Manager (TIM) to determine whether a user has logged on to TIM and requested to debug the current IMS transaction or DB/2 stored procedure.

The REPOSITORY option requires that the debugging user ID be granted RACF authority to debug tasks initiated by the generic ID. This is done via the EQADTOOL.GENERICID.*generic_user_ID* facility. To set this up, use the following RACF commands:

```
RDEFINE EQADTOOL.GENERICID.generic_user_ID CLASS(FACILITY) UACC(NONE)
PERMIT EQADTOOL.GENERICID.generic_user_ID ID(user) ACC(READ)
```

The *generic_user_ID* can be a pattern.

The REPOSITORY option also requires that you start the Terminal Interface Manager started task with the REPOSITORY option. See "Starting the Terminal Interface Manager" in IBM z/OS Debugger Customization Guide for more information.

**CLIENTID**
Specifies that z/OS Debugger uses the DB/2 client user ID for a stored procedure call to determine whether a remote debug user has requested to debug DB/2 stored procedures that execute with that client user ID. If such a user exists, their user ID will be used to locate the delay debug profile data set.

***'file_name'***
Specifies an MVS sequential data set with FB LRECL 80 characteristics.

**Usage notes**

- This command does not support 64-bit programs.

**Example**

```
EQAXOPT DLAYDBGXRF,DSN,'EQAW.TRNUSRID.XREF'
EQAXOPT DLAYDBGXRF,REPOSITORY
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
*Debugging tasks running under a generic user ID* in the *IBM z/OS Debugger User's Guide*

## DOPTACBDSN

You can specify this command only in the EQAOPTS load module. It cannot be specified at run time.

The DOPTACBDSN command identifies the data set that will contain DOPT PSBs that are created by z/OS Debugger for the IMS Transaction Isolation Facility. This data set will be used to store ACBs generated for the EQATcccn DOPT PSBs.

There is no default value for this command. If your site will use the IMS Transaction Isolation Facility, this command must be specified.

The following diagram describes the syntax of the DOPTACBDSN command:


***data_set_name***
Fully-qualified name of the data set that will contain the EQAT*cccn* DOPT PSBs.

## DTCNDELETEDEADPROF

You can specify this command only in the EQAOPTS load module. It cannot be specified at run time.

This command controls the deletion of DTCN profiles. A dead profile is a profile whose owner has logged off or disconnected from the CICS region.

DTCN scans its repository for dead profiles from time to time, and on demand when EQADCDEL is called. When a dead profile is found, it deactivates the profile by default. You can specify DTCNDELETEDEADPROF to delete the profile instead. The dead profile is deactivated or deleted by the INACTIVATE or DELETE DTCN function.

The following diagram describes the syntax of the DTCNDELETEDEADPROF command:


**NO**
Indicates that you want DTCN not to delete the profile. NO is the default option.

**YES**
Indicates that you want DTCN to delete the profile.

**Example**

```
EQAXOPT DTCNDELETEDEADPROF,YES
```

## DTCNFORCE*xxxx*

You can specify these commands only in the EQAOPTS load module. You cannot specify them at run time.

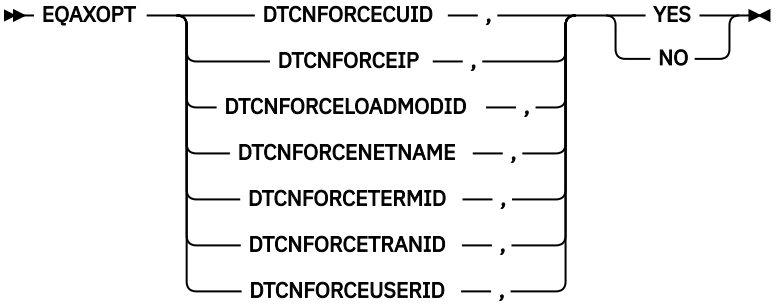
If your users create debugging profiles with DTCN, you can use the DTCNFORCE*xxxx* commands to require that certain DTCN fields are not left blank. The following list describes each resource type you can require each user to specify:

- DTCNFORCECUID or DTCNFORCEPROGID, which requires the user to specify the name of a compile unit or compile units.
- DTCNFORCEIP, which requires the user to specify the IP name or address.
- DTCNFORCELOADMODID, which requires the user to specify the name of a load module or load modules.
- DTCNFORCENETNAME, which requires the user to specify the four character name of a CICS terminal or a CICS system.
- DTCNFORCETERMID, which requires the user to specify the CICS terminal.

- DTCNFORCETRANID, which requires the user to specify a transaction ID.
- DTCNFORCEUSERID, which requires the user to specify a user ID.

If any of the statements are not included, the statement defaults to NO.

The following diagram describes the syntax of the DTCNFORCE*xxxx* command:




**YES**
Indicates that the specified field is required.

**NO**
Indicates that the specified field is not required.

**Examples**

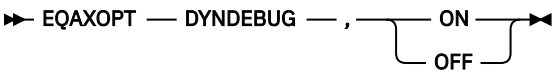
```
EQAXOPT DTCNFORCEUSERID,YES
EQAXOPT DTCNFORCETRANID,NO
```

## DYNDEBUG

z/OS Debugger Reference and Messages describes how you use the SET DYNDEBUG command to enable or disable dynamic debug mode in z/OS Debugger.

The initial default setting is DYNDEBUG ON. If you want to change the initial default setting, use the EQAOPTS DYNDEBUG command.

The following diagram describes the syntax of the DYNDEBUG command:




**ON**
Sets the initial default to DYNDEBUG ON.

**OFF**
Sets the initial default to DYNDEBUG OFF.

**Usage notes**

- This command does not support 64-bit programs.

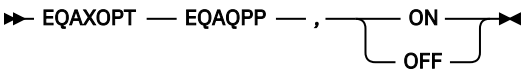
**Example**

```
EQAXOPT DYNDEBUG,OFF
```

## EQAQPP

You must specify this command to enable z/OS Debugger to debug MasterCraft Q++ programs, provided by Tata Consultancy Services Ltd. For more information about how to enable z/OS Debugger to support MasterCraft Q++, contact Tata Consultancy Services Ltd.

The following diagram describes the syntax of the EQAQPP command:




**ON**
Indicates z/OS Debugger supports Q++ debugging.

**OFF**
Indicates z/OS Debugger does not support Q++ debugging. If you do not specify the EQAQPP command, OFF is the default.

**Usage notes**

- This command does not support 64-bit programs.

**Example**

```
EQAXOPT EQAQPP,ON
```

## EXPLICITDEBUG

The *IBM z/OS Debugger Reference and Messages* describes how you use the SET EXPLICITDEBUG command to enable explicit debug mode in z/OS Debugger. However, before you can enter the SET EXPLICITDEBUG command, z/OS Debugger has already processed the initial load module and loaded the debug data for the compile units it contains. If you want to enable explicit debug mode prior to processing the initial load module, use the EQAOPTS EXPLICITDEBUG command.

The following diagram describes the syntax of the EXPLICITDEBUG command:


**ON**
Enables explicit debug mode.

**OFF**
Disables explicit debug mode. This is the default.

**Example**

```
EQAXOPT EXPLICITDEBUG,ON
```

## GPFDSN

You can create a *global preferences file* that runs a set of z/OS Debugger commands at the start of all z/OS Debugger sessions. For example, a global preferences file can have a command that sets PF keys to specific values. If your site uses the PF6 key as the program exit key, you can specify the SET PF6 "EXIT" = QUIT; command, which assigns the z/OS Debugger QUIT command to the PF6 key, in the global preferences file. (See "Customizing your full-screen session" in the IBM z/OS Debugger User's Guide for a description of the interface features you can change.)

Whenever a user starts z/OS Debugger, z/OS Debugger processes the commands in the global preferences file first. The user can also create his or her own preferences file and a commands file. In this situation, z/OS Debugger processes the files in the following order:

1. Global preferences file
2. User preferences file
3. Commands file

To create a global preferences file, do the following steps:

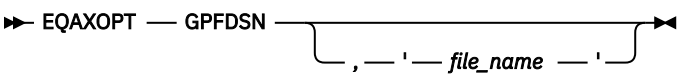
1. Create a preferences file that is stored as a sequential file or a PDS member. Refer to *IBM z/OS Debugger User's Guide* for a description of preferences files.

   The rules for the preferences file are dependant on the programming language of the first program z/OS Debugger encounters. Because you might not know what programming language z/OS Debugger will encounter first, use the following rules when you create the preferences file:

   - Put the commands in columns 8 - 72.
   - Do not put line numbers in the file.
   - Use `COMMENT` or `/* */` to delimit comments.
2. Specify the GPFDSN command to indicate the name of the global preferences file.

   For *'file_name'*, specify the name of the data set where the global preferences file will be stored.

The following diagram describes the syntax of the GPFDSN command:




***'file_name'***
The name of the data set where you stored the global preferences file.

**Examples**

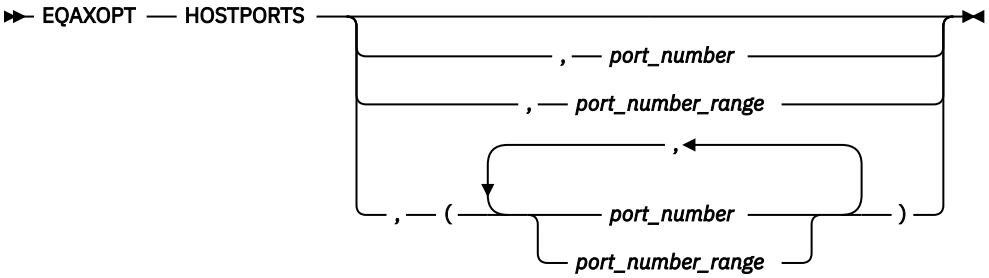
```
EQAXOPT GPFDSN,'GROUP1.COMMON.DTOOL.PREFS'
EQAXOPT GPFDSN
```

## HOSTPORTS

You can specify this command only in the EQAOPTS load module. It cannot be specified at run time. To use this command in the CICS environment, you must be using the TCP/IP Socket Interface for CICS. For instructions on activating the TCP/IP Socket Interface for CICS, see the z/OS Communications Server *IP CICS Sockets Guide*.

You can use this command to specify a host port or range of ports for a TCP/IP connection from the host to a workstation when using remote debug mode.

The following diagram describes the syntax of the HOSTPORTS command:




***port_number***
A positive integer (1-32767) identifying a TCP/IP port number.

***port_number_range***
The first and last *port_number* identifying a range of port numbers, separated by a hyphen (-).

**Examples**

```
EQAXOPT  HOSTPORTS,29500-30499
EQAXOPT  HOSTPORTS,(29500-30499,31500-32499)
```

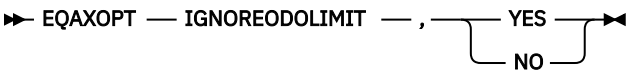
## IGNOREODOLIMIT

This command tells z/OS Debugger to display COBOL table data items even when an ODO value is out of range, and to suppress the following messages:

**MFI and batch:**
EQA1471E Incorrect value for ODO variable data item

**Remote:**
EQA2377E Invalid data.

The following diagram describes the syntax of the IGNOREODOLIMIT command:




**YES**
Indicates that z/OS Debugger should display the requested table data item even when an ODO value is out of range, and to suppress issuing EQA1471E or EQA2377E.

**NO**
Indicates that z/OS Debugger should not display the requested table data item when an ODO value is out of range, and to issue EQA1471E or EQA2377E.

**Examples**

```
EQAXOPT IGNOREODOLIMIT,YES
EQAXOPT IGNOREODOLIMIT,NO
```

**Notes:**

- The default setting is NO.
- IGNOREODOLIMIT only affects the behaviour of z/OS Debugger if the compilation unit was compiled with one of the following compilers:
  - COBOL for OS/390 & VM Version 2 (5648-A25)
  - Enterprise COBOL for z/OS and OS/390 Version 3 (5655-G53)
  - Enterprise COBOL for z/OS Version 4 (5655-S71)
- IGNOREODOLIMIT is ignored for LangX COBOL.

## IMSISOORIGPSB

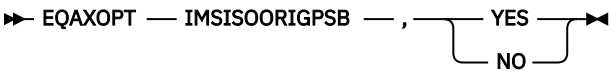
**Note:** This command is deprecated. It is accepted for compatibility but has no effect. The original PSB is always preserved.

You can specify this command only in the EQAOPTS load module. It cannot be specified at run time. For the command to take effect, the EQAOPTS load module that contains it must be in the search path of the IMS control region.

This command instructs the IMS Transaction Isolation Facility to preserve the original PSB of the transaction when a message is routed to a private message processing region.

For example, if you register to debug conversational transaction IVTCB, and the private message class 121 is assigned to you, a message routed to your private message processing region will be sent to transaction EQAC1211. Ordinarily, the PSB name for EQAC1211 is EQAT1211. With IMSISOORIGPSB in effect, EQAC1211 is changed to associate the PSB name with IVTCB, DFSIVP34.

The following diagram describes the syntax of the IMSISOORIGPSB command:



**YES**
Indicates that you want IMS Isolation to preserve the original PSB of the transaction when a message is routed to a private message. IMS Transaction Isolation bypasses its normal operations for bringing up the transaction under the control of the debugger. Therefore, the debugger starts only for Language Environment-enabled programs.

**NO**
Indicates that you want IMS Isolation not to preserve the original PSB of the transaction when a message is routed to a private message. **NO** is the default setting.

## LOGDSN

By default, z/OS Debugger handles the log file data set in one of the following ways:

- In a non-CICS environment, when z/OS Debugger starts in batch mode or full-screen mode, and you allocate the INSPLOG DD name, z/OS Debugger runs the command `SET LOG ON FILE INSPLOG OLD` and starts writing the log to INSPLOG.
- In a CICS environment, when z/OS Debugger starts in full-screen mode, it runs the command `SET LOG OFF`. If you want a log file, you run the command `SET LONG ON FILE` *`fileid`* `OLD` and z/OS Debugger starts writing the log to *fileid*.

`LOGDSN` allows a site or a user to specify the default data set name for the log file. If you specify the `LOGDSN` command, z/OS Debugger handles the log file in the following way:

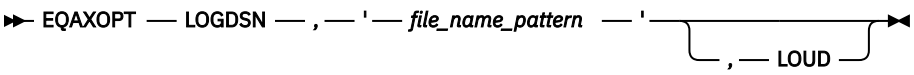
- In a non-CICS environment, when z/OS Debugger starts in batch mode or full-screen mode, and you allocate the INSPLOG DD name, z/OS Debugger runs the command `SET LOG ON FILE INSPLOG OLD` and starts writing the log to INSPLOG . This behavior remains the same.
- In a non-CICS environment, when z/OS Debugger starts in full-screen mode, if you do not allocate the INSPLOG DD name, z/OS Debugger runs the command `SET LOG ON FILE` *`fileid`* `OLD` and starts writing the log to the data set specified in the `LOGDSN` command.
- In CICS, when z/OS Debugger starts in full-screen mode, it runs the command `SET LOG ON FILE` *`fileid`* `OLD` and starts writing the log to the data set specified in the `LOGDSN` command.

  This allows a user to always write the log file to a data set, whether in CICS or not, and without having to pre-allocate the log file data set.

  For instructions on how to specify the allocation parameters for automatically creating the data set, see "LOGDSNALLOC" on page 312. Use the EQAOPTS `LOGDSN` and `LOGDSNALLOC` commands to help a new z/OS Debugger user automatically create and write to the log file.

If you are an existing z/OS Debugger user that uses a SAVESETS data set, and you or your site specify the EQAOPTS commands `LOGDSN` and `LOGDSNALLOC`, then the SAVESETS data set contains a `SET LOG` command that overrides the EQAOPTS command `LOGDSN`.

The following diagram describes the syntax of the `LOGDSN` command:




***file_name_pattern***
Specifies a naming pattern that determines the name of the data set that contains the log file. Follow these guidelines when you create the naming pattern:

- Create a data set name that includes `&&USERID.` as one of the qualifiers. z/OS Debugger substitutes the user ID of the current user for this qualifier when it determines the name of the data set.
- Specify `NULLFILE` to indicate you do not want z/OS Debugger to write to a log file.

**LOUD**
Specifies that z/OS Debugger display WTO messages, which helps you debug processing done by this command. z/OS Debugger normally does not display any messages if it does not find the data set. If

you are trying to determine why z/OS Debugger is not writing to this log file, specify LOUD to see if it displays any messages.

If you choose to implement this option, users who want to use the EQAOPTS LOGDSN command must create a log file in one of the following ways:

- Instruct z/OS Debugger to create the log file by specifying the EQAOPTS LOGDSNALLOC command, as described in “LOGDSNALLOC” on page 312.
- Create the log file manually with the allocation parameters that are described in "Data sets used by z/OS Debugger" in the *IBM z/OS Debugger User's Guide* .

**Example**

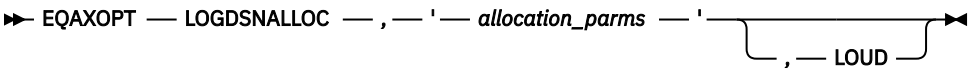
```
EQAXOPT  LOGDSN,'&&USERID.DBGTOOL.LOG'
```

If you log in with user ID `jsmith`, z/OS Debugger determines the name of the data set to be `JSMITH.DBGTOOL.LOG`.

## LOGDSNALLOC

Indicates that you want z/OS Debugger to create the log file data set specified by EQAOPTS LOGDSN command if it does not exist. You specify the EQAOPTS LOGDSNALLOC command with the corresponding allocation parameters for the data set, which z/OS Debugger uses when it creates the data set.

The following diagram describes the syntax of the LOGDSNALLOC command:




***allocation_parms***
Specifies the allocation parameters you want z/OS Debugger to use when it creates the data set. You can specify only the keys in the following list:

- BLKSIZE
- BLOCK
- CYL
- DATACLAS
- DIR
- DSNTYPE
- DSORG
- LRECL
- MGMTCLAS
- RECFM
- SPACE
- STORCLAS
- TRACKS
- UNIT
- VOL

Separate the keys by one or more blanks. z/OS Debugger does not provide defaults for any of the keys.

For information on the format of the keys, see the chapter "BPXWDYN: a text interface to dynamic allocation and dynamic output" in the *z/OS Using REXX and z/OS UNIX System Services* manual. Specify that the data set be sequential. To learn about other formatting rules for the log file, see "Data sets used by z/OS Debugger" of the IBM z/OS Debugger User's Guide.

**LOUD**
Specifies that z/OS Debugger display WTO messages, which helps you debug processing done by this command. z/OS Debugger normally does not display any messages when it creates this data set. If you are trying to determine why the log file was not created, specify LOUD to view any messages.

**Example**

```
EQAXOPT  LOGDSNALLOC,'MGMTCLAS(STANDARD) STORCLAS(DEFAULT)    +
         LRECL(72) BLKSIZE(0) RECFM(F,B) DSORG(PS) SPACE(2,2) +

CYL'
```

# MAXTRANUSER

You can specify this command only in the EQAOPTS load module. It cannot be specified at run time.

The MAXTRANUSER command defines the maximum number of IMS transactions that a single user can register to debug using the IBM Transaction Isolation Facility. If this command is not specified, the default value of 15 will be used.

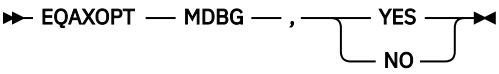
The following diagram describes the syntax of the MAXTRANUSER command:


***max_trans***
An integer value between 1 and 15 to designate the maximum number of transactions a user can register to debug.

# MDBG

If you are using z/OS XL C/C++, Version 1.10 or later, you can indicate that z/OS Debugger always searches for .mdbg files to retrieve the source and debug information by using the MDBG command.

The following diagram describes the syntax of the MDBG command:




**YES**
Indicates that z/OS Debugger searches for .mdbg files.

**NO**
Indicates that z/OS Debugger does not search for .mdbg files.

When you set MDBG to YES, z/OS Debugger retrieves the debug information from an .mdbg file and does not try to find the debug information from the following sources, even if they exist:

- a .dbg file
- if the program was compiled with the ISD compiler option, the object

If you do not specify MDBG or set it to NO, z/OS Debugger retrieves the debug information from either the .dbg file or, if the program was compiled with the ISD compiler option, the object.

**Example**

```
EQAXOPT MDBG,YES
```

# MULTIPROCESS

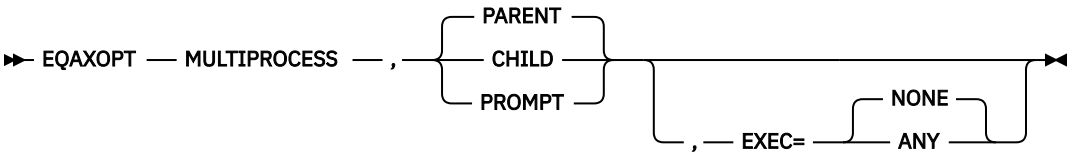
Controls the behavior of z/OS Debugger when a new POSIX process is created by a fork() or exec() function in the application.

With the MULTIPROCESS command, you can instruct z/OS Debugger to perform any of the following tasks when a new POSIX process is created:

- Continue debugging the current process. The current process is also referred to as the PARENT process.
- Stop debugging the current process and start debugging the newly created process. The newly created process is also referred to as the CHILD process.
- Prompt you to decide whether to follow the PARENT or CHILD process.

**Note:** The MULTIPROCESS command applies only to remote debug mode.

The following diagram describes the syntax of the MULTIPROCESS command:




The following list describes the parameters of the MULTIPROCESS command:

**PARENT**
Indicates that z/OS Debugger continues with the current debug session; that is, z/OS Debugger follows the PARENT process.

**CHILD**
Indicates that z/OS Debugger stops debugging the current process and starts debugging the newly created process; that is, z/OS Debugger follows the CHILD process.

**PROMPT**
Indicates that the remote debug GUI prompts you to decide whether to follow the PARENT or CHILD process.

**EXEC=ANY**
Indicates that z/OS Debugger debugs any process that is reinitialized by the exec() function.

**EXEC=NONE**
Indicates that z/OS Debugger does not debug a process that is reinitialized by the exec() function. If you do not specify the EXEC option, the default setting is EXEC=NONE.

**Examples**

- Specify that z/OS Debugger follows the PARENT process and debugs the new process that is created by the exec() function.

```
EQAXOPT MULTIPROCESS,PARENT,EXEC=ANY
```

- Specify that z/OS Debugger follows the CHILD process and does not debug the new process that is created by the exec() function.

```
EQAXOPT MULTIPROCESS,CHILD,EXEC=NONE
```

- Specify that you are prompted to choose whether to follow the PARENT or CHILD process and z/OS Debugger does not debug the new process that is created by the exec() function.

```
EQAXOPT MULTIPROCESS,PROMPT
```

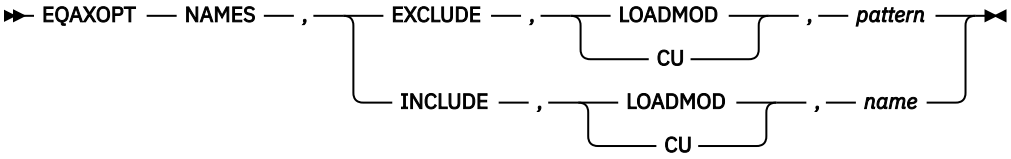
# NAMES

The topic "Solving Problems in Complex Applications" in the IBM z/OS Debugger User's Guide in the describes how the NAMES command can be used to perform several specific functions dealing with load module and compile unit names recognized by z/OS Debugger. However, the NAMES command cannot be used to alter the behavior of load module or compile unit names that have already been seen by z/OS Debugger at the time the NAMES command is processed.

If it becomes necessary to perform these functions on the initial load module processed by z/OS Debugger or on any of the compile units contained in that load module, you must provide the information (that would otherwise have been specified using the NAMES command) through the EQAOPTS NAMES command.

One or more invocations of the EQAOPTS NAMES command can be used for this purpose.

The following diagram describes the syntax of the NAMES command:




Each of these fields corresponds to the similar parameter in the z/OS Debugger NAMES command. If you use an asterisk (*) in *pattern* to indicate a wildcard, you must enclose *pattern* in apostrophes.

**Examples**

```
EQAXOPT NAMES,EXCLUDE,LOADMOD,'ABC1*'
EQAXOPT NAMES,EXCLUDE,CU,MYCU22
EQAXOPT NAMES,EXCLUDE,CU,'MYCU*'
EQAXOPT NAMES,INCLUDE,LOADMOD,CEEMYMOD
EQAXOPT NAMES,INCLUDE,CU,EQATESTP
```

## NODISPLAY

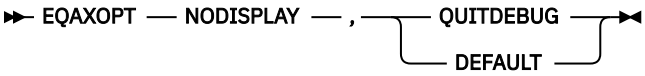
In the following two situations, in which a user can request a specific user interface, that interface might not be available:

- Full-screen mode using the Terminal Interface Manager. If the terminal is not available, the program being debugged terminates with a U4038 abend.
- Remote debugger. If the remote debugger is not available, z/OS Debugger will use full-screen mode if the user is running under TSO. If the user is not running under TSO, z/OS Debugger will use batch mode.

In both cases, Write To Operator (WTO) messages also appear.

You can modify these behaviors by specifying the EQAOPTS NODISPLAY command so that z/OS Debugger continues processing as if the user immediately entered a QUIT DEBUG command. This modification prevents any forced abend or prevents the debugger from starting.

The following diagram describes the syntax of the NODISPLAY command:




**DEFAULT**
z/OS Debugger follows the default behavior.

**QUITDEBUG**
z/OS Debugger displays a message that indicates that z/OS Debugger will quit, and that the user interface could not be used. z/OS Debugger processing continues as if the user entered a QUIT DEBUG command.

**Example**

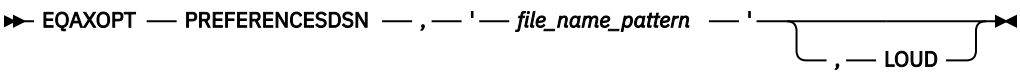
```
EQAXOPT NODISPLAY,QUITDEBUG
```

## PREFERENCESDSN

Indicates that you want z/OS Debugger to read a user's preferences file (with the name of the data set containing the preferences file determined by the specified naming pattern) each time it starts. This works in the following situation:

- You do not specify a data set name or DD name for the user's preferences file using any other method; for example, the TEST runtime option.
- In a non-CICS environment, you do not allocate ISPPPREF DD.
- You or your site specifies the PREFERENCESDSN command.
- The data set specified by the PREFERENCESDSN command exists.

The following diagram describes the syntax of the PREFERENCESDSN command:




***file_name_pattern***
Specifies a naming pattern that determines the name of the data set that contains the preferences file. Follow these guidelines when you create the naming pattern:

- Create a data set name that includes &&USERID. as one of the qualifiers. z/OS Debugger substitutes the user ID of the current user for this qualifier when it determines the name of the data set.
- Specify NULLFILE to indicate you do not want z/OS Debugger to process a preferences file.

**LOUD**
Specifies that z/OS Debugger display WTO messages, which helps you debug processing done by this command. z/OS Debugger normally does not display any messages if it does not find the data set. If you are trying to determine why z/OS Debugger is not processing your preferences file, specify LOUD to see if it displays any messages about not finding the data set.

If you choose to implement this option, users who want to use this function must create the preferences file as a sequential data set with the allocation parameters that are described in "Data sets used by z/OS Debugger" in the *IBM z/OS Debugger User's Guide*.

**Example**

```
EQAXOPT  PREFERENCESDSN,'&&USERID.DBGTOOL.PREFS'
```

If you log in with user ID `jsmith`, z/OS Debugger determines the name of the data set to be `JSMITH.DBGTOOL.PREFS`.

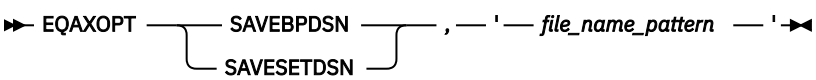
## SAVEBPDSN, SAVESETDSN

You can modify the default names of the data sets used to save and restore settings and breakpoints, monitor values, and LOADDEBUGDATA (LDD) specifications. The following list describes the initial default names:

- For settings: *userid*.DBGTOOL.SAVESETS
- For breakpoints, monitor values, and LOADDEBUGDATA (LDD) specifications: *userid*.DBGTOOL.SAVEBPS

To change the default name for either or both of these data sets, you need to specify the EQAOPTS SAVESETDSN and SAVEBPDSN commands, along with a corresponding naming pattern for the data set.

The following diagram describes the syntax of the SAVESETDSN and SAVEBPDSN commands:



***file_name_pattern***
Specifies a naming pattern for the data set that stores this information.

In most environments, you should choose one of the following rules for the naming pattern:

- Any data set name that includes &&USERID. as one of the qualifiers. z/OS Debugger substitutes the user ID of the current user for this qualifier when it creates the data set.
- A DD name (Reminder: DD names are not supported under CICS)
- The string NULLFILE to indicate that saving and restoring this information is not supported

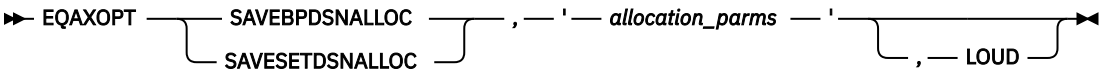
**Examples**

```
EQAXOPT SAVESETDSN,'CICS.DTDATA.&&USERID.SAVSET'
EQAXOPT SAVEBPDSN,'&&USERID.USERDATA.DTOOL.SAVBPMON';
```

## SAVESETDSNALLOC, SAVEBPDSNALLOC

Indicates that you want z/OS Debugger to create the data sets for SAVESETS, SAVEBPS, or both (specified by EQAOPTS SAVESETDSN or SAVEBPDSN commands) if they do not exist. You specify the EQAOPTS SAVESETDSNALLOC and SAVEBPDSNALLOC commands with the corresponding allocation parameters for the data sets, which z/OS Debugger uses when it creates the data sets. After creating each data set, z/OS Debugger runs commands that save the information (settings, breakpoints, monitors, preferences, and LDD specifications) in the corresponding data set.

The following diagram describes the syntax of the SAVEBPDSNALLOC and SAVESETDSNALLOC commands:




***allocation_parms***
Specifies the allocation parameters you want z/OS Debugger to use when it creates the data set. You can specify only the keys in the following list:

- BLKSIZE
- BLOCK
- CYL
- DATACLAS
- DIR
- DSNTYPE
- DSORG
- LRECL
- MGMTCLAS
- RECFM
- SPACE
- STORCLAS
- TRACKS
- UNIT
- VOL

Separate the keys by one or more blanks. z/OS Debugger does not provide defaults for any of the keys.

For information on the format of the keys, see the chapter "BPXWDYN: a text interface to dynamic allocation and dynamic output" in the *z/OS Using REXX and z/OS UNIX System Services* manual. Specify that the data set be sequential for SAVESETS; a PDS or PDSE for SAVEBPS. To learn about

other formatting rules for these files, see "Data sets used by z/OS Debugger" of the IBM z/OS Debugger User's Guide.

**LOUD**
Specifies that z/OS Debugger display WTO messages, which helps you debug processing done by this command. z/OS Debugger normally does not display any messages when it creates these data sets. If you are trying to determine why the data sets were not created, specify LOUD to view any messages.

z/OS Debugger does the following tasks when you specify these commands:

1. If you specified the SAVESETDSNALLOC command, it creates the SAVESETS data set.
2. If it creates the SAVESETS data set successfully, it runs the following commands:

```
SET SAVE SETTINGS AUTO;
SET RESTORES SETTINGS AUTO;
```

   If it did not create the SAVESETS data set successfully, it skips the rest of these steps and does the next processing task.
3. If you specified the SAVEBPDSNALLOC command, it creates the SAVEBPS data set.
4. If it creates the SAVEBPS data set successfully, it runs the following commands:

```
SET SAVE BPS AUTO;
SET SAVE MONITORS AUTO;
SET RESTORE BPS AUTO;
SET RESTORE MONITORS AUTO;
```

In a CICS environment, review the performance implications discussed in the "Performance considerations in multi-enclave environments" section of the "Using full-screen mode: overview" topic in the *IBM z/OS Debugger User's Guide* before choosing to implement the SAVEBPDSNALLOC command. If you think the performance implications might adversely affect your site, do not implement the SAVEBPDSNALLOC command in the EQAOPTS for CICS.

**Example**

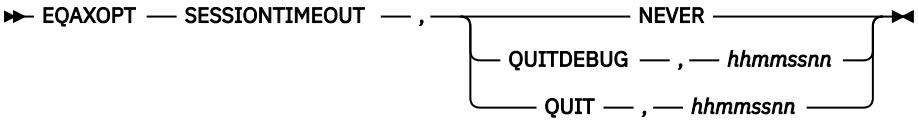
```
      EQAXOPT  SAVESETDSNALLOC,'MGMTCLAS(STANDARD) STORCLAS(DEFAULT)  +
               LRECL(3204) BLKSIZE(0) RECFM(V,B) DSORG(PS) SPACE(2,2) +
               TRACKS'
      EQAXOPT  SAVEBPDSNALLOC,'MGMTCLAS(STANDARD) STORCLAS(DEFAULT)   +
               LRECL(3204) BLKSIZE(0) RECFM(V,B) DSORG(PO)            +
               DSNTYPE(LIBRARY) SPACE(1,3) CYL'
```

# SESSIONTIMEOUT

You can specify this command only in the EQAOPTS load module. It cannot be specified at run time.

You can establish a timeout for idle z/OS Debugger sessions that use the Terminal Interface Manager. Timed out sessions are canceled after a specified period of no user activity.

The following diagram describes the syntax of the SESSIONTIMEOUT command:




**NEVER**
No timeout is enforced. Unattended sessions will not be canceled.

**QUITDEBUG,*hhmmssnn***
Sessions left unattended for the specified time interval will be canceled, and the process being debugged will proceed as though the QUIT DEBUG command had been entered.

The time interval is expressed as *hhmmssnn*, where

- *hh* is the number of hours,

- *mm* is the number of minutes,
- *ss* is the number of seconds,
- *nn* is the number of hundredths of seconds.

**QUIT,*hhmmssnn***
Sessions left unattended for the specified time interval will be canceled, and the process being debugged will be terminated with a U4038 abend, as though the `QUIT ABEND` command had been entered.

The time interval is expressed as *hhmmssnn*, where

- *hh* is the number of hours,
- *mm* is the number of minutes,
- *ss* is the number of seconds,
- *nn* is the number of hundredths of seconds.

If the command is not specified in the EQAOPTS load module, the default behavior is "NEVER", which means that any full-screen mode session using Terminal Interface Manager left unattended will not be canceled.

**Example**

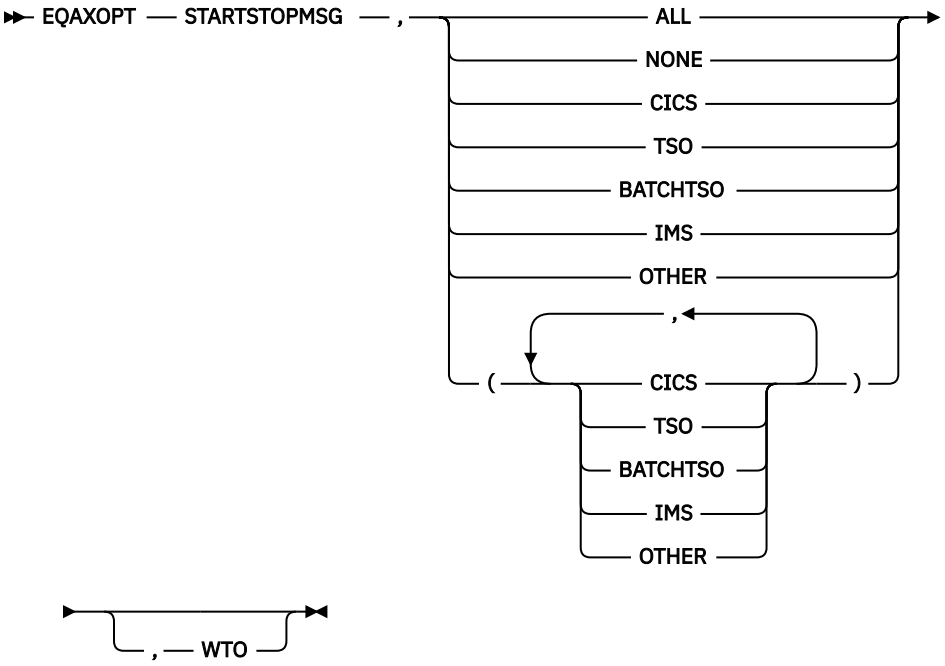
To specify a timeout interval of 1 hour and allow the debugged process to proceed after the debug session is canceled, enter the following EQAOPTS command:

```
EQAXOPT SESSIONTIMEOUT,QUITDEBUG,01000000
```

## STARTSTOPMSG

This command controls whether to issue a message when each debugging session is initiated or terminated. By default, these messages are not issued.

The following diagram describes the syntax of the STARTSTOPMSG command:




The following list describes the parameters of the STARTSTOPMSG command:

**ALL**
Indicates that the start/stop messages should be written in all environments.

**NONE**
Indicates that no start/stop messages should be written. If you do not specify EQAOPTS STARTSTOPMSG, the default option is NONE.

**CICS**
Indicates that the start/stop messages should be written in the CICS environment.

**TSO**
Indicates that the start/stop messages should be written in the TSO environment.

**BATCHTSO**
Indicates that the start/stop messages should be written in the BATCH TSO environment.

**IMS**
Indicates that the start/stop messages should be written when running under IMS. In addition, an informational message that contains the IMS system ID, region ID, and transaction ID is written.

**OTHER**
Indicates that the start/stop messages should be written in all other environments, such as MVS batch.

**WTO**
Indicates that the start/stop messages should be written to the system log using a WTO.

**Examples**

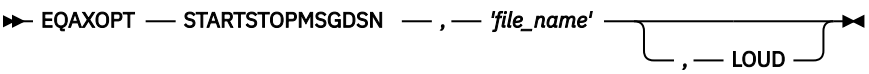
```
EQAXOPT STARTSTOPMSG,ALL,WTO
EQAXOPT STARTSTOPMSG,(TSO,OTHER),WTO
```

## STARTSTOPMSGDSN

You can specify this command only in the EQAOPTS load module. It cannot be specified at run time.

You can use this command to indicate that you want z/OS Debugger to write a message in the message file when the debug session is started and stopped.

The following diagram describes the syntax of the STARTSTOPMSGDSN command:




***'file_name'***
Specifies an MVS sequential data set with FB LRECL 80 characteristics. It is recommended that you allocate sufficient space for the file and perform regular maintenance by removing outdated messages. In a CICS environment, the region owner must have the update authority to the data set.

**LOUD**
Specifies that z/OS Debugger displays WTO messages, which help you debug processing done by this command. z/OS Debugger normally does not display any messages when it operates on the data set. If you try to determine problems related to the data set, specify LOUD to view any related messages.

**Example**

```
EQAXOPT STARTSTOPMSGDSN,'EQAW.SSMSG.LOG'
```

# SUBSYS

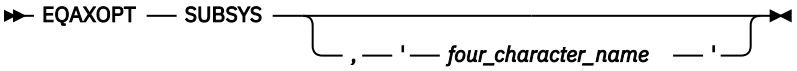
If both of the following conditions apply at your site, you need to use the EQAOPTS SUBSYS command:

- The source code is managed by a library system that requires that you specify the SUBSYS=*library_subsystem_name* allocation parameter when you allocate a data set.
- Your users are debugging C or C++ programs without using the EQAOPTS MDBG command or debugging Enterprise PL/I programs compiled without the SEPARATE suboption of the TEST compiler option.

In this case, you must run z/OS Debugger and the specified subsystem on the same system.

You cannot use SUBSYS to debug programs that run under CICS.

The following diagram describes the syntax of the SUBSYS command:




***four_character_name***
Specifies the subsystem name to be used.

**Examples**

```
EQAXOPT SUBSYS
EQAXOPT SUBSYS,'SBSX'
```

## SVCSCREEN

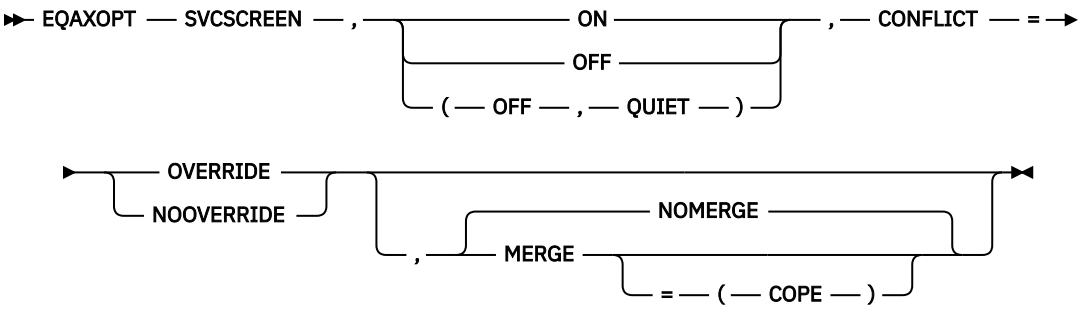
In a non-CICS environment, z/OS Debugger requires SVC screening for the following situations:

- Invoking z/OS Debugger by using EQANMDBG to debug programs that start outside Language Environment including non-Language Environment COBOL programs.
- Debugging programs that do not run in Language Environment and are started by programs that begin in Language Environment.
- Debugging LangX COBOL programs.
- Detecting services such as MVS LINK, LOAD, DELETE and ATTACH.

If you need to run z/OS Debugger in any of the following situations, you must specify the actions that z/OS Debugger must take regarding SVC screening:

- Start z/OS Debugger by using EQANMDBG in an environment that already uses SVC screening.
- Run z/OS Debugger when debugging programs that do not run in Language Environment and are started by programs that begin in Language Environment.
- Run z/OS Debugger when debugging LangX COBOL programs.
- Run z/OS Debugger when you need to detect services such as MVS LINK, LOAD and DELETE.
- Run z/OS Debugger when you debug subtasks within a multi-tasked application, where subtasks are started by using the ATTACH assembler macro.
- Run z/OS Debugger in a situation that requires SVC screening and SVC screening is already in use by a program with which z/OS Debugger supports MERGE SVC screening as described by the MERGE operand that follows.

The following diagram describes the syntax of the SVCSCREEN command:




**ON**
Indicates that you want z/OS Debugger to use SVC screening in order to support MVS LOAD, DELETE, and LINK SVCs.

**OFF**
Indicates that you want z/OS Debugger to not use SVC screening. z/OS Debugger will not know about programs started through MVS LOAD, DELETE, and LINK SVCs. If you start z/OS Debugger by using the EQANMDBG program, the OFF setting is ignored.

**QUIET**
Suppresses message EQA2458I, which is written to the z/OS Debugger log when SVC screening is disabled by default.

**CONFLICT=**
Specifies what you want z/OS Debugger to do when ON is specified or defaulted and SVC screening is already used by another program.

**OVERRIDE**
Indicates that you want z/OS Debugger to override the current SVC screening and take control of SVC screening.

**NOOVERRIDE**
Indicates that if SVC screening is already in use, z/OS Debugger does not initiate SVC screening and proceeds as if OFF were specified.

**NOMERGE**
Indicates that SVC screening is not to be merged with SVC screening used by any other product. NOMERGE is the default.

**MERGE**
Indicates that when SVC screening is already being used by another program when z/OS Debugger starts, z/OS Debugger saves the current SVC screening environment, then enables SVC screening for both z/OS Debugger and the other program. When z/OS Debugger terminates, it restores the original SVC screening environment.

Currently, z/OS Debugger supports the MERGE command with only one other program: COPE.

If you specify the MERGE command and z/OS Debugger does not recognize the program that is using the SVC screening, the MERGE command is ignored and z/OS Debugger starts based on the value of the CONFLICT option.

**MERGE=(COPE)**
If COPE is active, z/OS Debugger saves the current SVC screening environment, then enables SVC screening for both z/OS Debugger and COPE. When z/OS Debugger terminates, it restores COPE's SVC screening environment.

If COPE is not active, z/OS Debugger starts based on the value of the CONFLICT option.

The default parameters for the EQAOPTS SVCSCREEN command is one of the following situations:

- If z/OS Debugger is started by using the EQANMDBG program:
  SVCSCREEN,ON,CONFLICT=NOOVERRIDE,NOMERGE
- If z/OS Debugger is started by any other method:
  SVCSCREEN,OFF,CONFLICT=NOOVERRIDE,NOMERGE

Use Table 13 on page 323 as a guide to select the appropriate suboptions.

**Usage notes**

- This command does not support 64-bit programs.

**Example**

```
EQAXOPT SVCSCREEN,ON,CONFLICT=OVERRIDE,NOMERGE
EQAXOPT SVCSCREEN,OFF,CONFLICT=NOOVERRIDE,NOMERGE
```

## Combinations of suboptions for the EQAOPTS SVCSCREEN command

The following table shows examples of combinations of suboptions for the EQAOPTS SVCSCREEN command:

*Table 13. Combination of SVSCREEN options and their effects*

| SVCSCREEN options | Type of z/OS Debugger session | Action |
|---|---|---|
| `OFF,CONFLICT=NOOVERRIDE` (default) | z/OS Debugger started by using EQANMDBG | Same as for `ON,CONFLICT=NOOVERRIDE`. |
| | z/OS Debugger started by any other method | • z/OS Debugger does not enable its SVC screening.<br>• You cannot debug programs that do not run in Language Environment which were started by programs that do run in Language Environment.<br>• z/OS Debugger does not detect the MVS services LINK, LOAD and DELETE.<br>• The CONFLICT setting is ignored when the OFF setting is specified. |
| `OFF,CONFLICT=OVERRIDE` | z/OS Debugger started by using EQANMDBG | Same as for `ON,CONFLICT=OVERRIDE`. |
| | z/OS Debugger started by any other method | Same as for `OFF,CONFLICT=NOOVERRIDE`.<br>The CONFLICT setting is ignored when the OFF setting is specified. |
| `ON,CONFLICT=NOOVERRIDE` | z/OS Debugger started by using EQANMDBG | If SVC screening is active, z/OS Debugger terminates. If SVC screening is not active, z/OS Debugger enables its SVC screening, runs the debugging session, and disables its SVC screening after the debugging session ends. |
| | z/OS Debugger started by any other method | If SVC screening is active, z/OS Debugger does not enable its SVC screening. You cannot debug programs that do not run in Language Environment which were started by programs that do run in Language Environment. z/OS Debugger does not detect the MVS services LINK, LOAD and DELETE.<br>If SVC screening is not active, z/OS Debugger enables its SVC screening, runs the debugging session, and disables its SVC screening after the debugging session ends. |

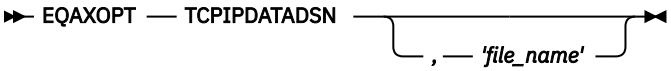
| *Table 13. Combination of SVSCREEN options and their effects (continued)* | | |
|---|---|---|
| **SVCSCREEN options** | **Type of z/OS Debugger session** | **Action** |
| `ON,CONFLICT=OVERRIDE` | z/OS Debugger started by using EQANMDBG | If any SVC screening is active and the `NOMERGE` option is in effect, z/OS Debugger overrides the existing SVC screening. This is also the default behavior. z/OS Debugger enables its SVC screening, runs the debugging session, and disables its SVC screening after the debugging session ends. If any SVC screening was active, z/OS Debugger restores the previous SVC screening. If you specify the `MERGE` option, see the following information about `MERGE`. |
| | z/OS Debugger started by any other method | |

## TCPIPDATADSN

You can specify this command only in the EQAOPTS load module. It cannot be specified at run time.

You can use this command to instruct z/OS Debugger to dynamically allocate the specified *'file_name'* to DDNAME SYSTCPD (if SYSTCPD is not already allocated). This provides the data set name for TCPIP.DATA when no GLOBALTCPIPDATA statement is configured in the system TCP/IP options.

The following diagram describes the syntax of the `TCPIPDATADSN` command:

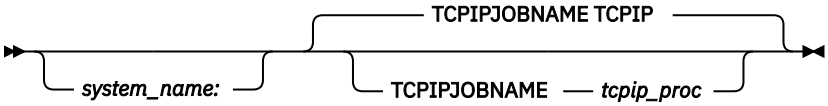



**Example**

```
EQAXOPT TCPIPDATADSN,'SYS2.TCPIP.DATA'
```

```
EQAXOPT TCPIPDATADSN,'SYS2.TCPIP.PARMLIB(TCPDATA)'
```

If you want to use an alternative TCP/IP stack, you can add an entry to the specified `TCPIP.DATA` dataset with a `TCPIPJOBNAME` statement.




**Example**

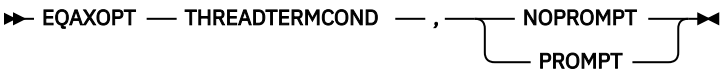
To specify TCPIPA as the name of the procedure that was used to start the TCP/IP address space, add the following code to the specified `TCPIP.DATA` dataset:

```
TCPIPJOBNAME TCPIPA
```

## THREADTERMCOND

You can indicate that z/OS Debugger should not prompt the user when a FINISH, CEE066, or CEE067 thread termination condition is raised by Language Environment, regardless of the suboptions used in the TEST runtime option. These conditions are raised by statements like STOP RUN, GOBACK, or EXEC CICS RETURN, which can occur frequently in an application program. Suppressing the display of these prompts can reduce the number of times your users are interrupted by this prompt during a debugging session.

The following diagram describes the syntax of the THREADTERMCOND command:




**NOPROMPT**
Suppress the display of termination prompts.

**PROMPT**
Prompts the user at termination. If you do nto specify the THREADTERMCOND command, the default PROMPT is used.

**Example**

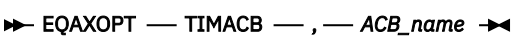
```
EQAXOPT THREADTERMCOND,NOPROMPT
```

# TIMACB

You can include this command only in the EQAOPTS load module. You cannot specify it at run time.

TIMACB identifies the name of an ACB, other than EQASESSM, that z/OS Debugger uses to make full-screen mode using the Terminal Interface Manager work in an environment where you want to run the Terminal Interface Manager on more than one LPAR in the same VTAM network. You specify TIMACB as the last step in "Example: Defining the VTAM EQAMVnnn and Terminal Interface Manager APPL definition statements when z/OS Debugger runs on four LPARs" in the IBM z/OS Debugger Customization Guide.

The following diagram describes the syntax of the TIMACB command:




***ACB_name***
The new ACB name you want used. The default name is EQASESSM.

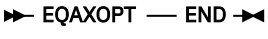
**Example**

```
EQAXOPT TIMACB,EQASESS2
```

# END

The END command identifies the last EQAOPTS command. You must always specify it and it must be the last command in the input stream.

The following diagram describes the syntax of the END command:




**Example**

```
EQAXOPT END
```

# Chapter 7. z/OS Debugger built-in functions

z/OS Debugger provides you with several built-in functions which allow you to manipulate variables. All z/OS Debugger built-in function names begin with a percent sign (%).

The table below summarizes the z/OS Debugger built-in functions. Unless otherwise indicated, the functions can be used with all supported languages.

| z/OS Debugger built-in function | Returns |
|---|---|
| "%CHAR (assembler, disassembly, and LangX COBOL)" on page 327 | Character value of an operand. |
| "%DEC (assembler, disassembly, and LangX COBOL)" on page 327 | Decimal value of an operand. |
| "%GENERATION (PL/I)" on page 328 | A specific generation of a controlled variable |
| "%HEX" on page 328 | Hexadecimal value of an operand |
| "%INSTANCES (C, C++, and PL/I)" on page 329 | Maximum value of %RECURSION for a block |
| "%RECURSION (C, C++, and PL/I)" on page 330 | An automatic variable or a parameter in a specific instance of a recursive procedure |
| "%WHERE (assembler, disassembly, and LangX COBOL)" on page 331 | A string indicating the address of the operand. |

## %CHAR (assembler, disassembly, and LangX COBOL)

Returns the EBCDIC character value of an operand.


***expression***
A valid assembler, disassembly, or LangX COBOL expression.

**Examples**

Assuming that R6 contains the address of a character string of length five, enter the following command:

```
LIST %CHAR(_STORAGE(%GPR6+1::5))
```

The Log window displays the corresponding character string.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"LIST expression command" on page 149

## %DEC (assembler, disassembly, and LangX COBOL)

Returns the decimal value of an operand.


***expression***
A valid assembler, disassembly, or LangX 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:

```
LIST %DEC(R1+2);
```

For LangX COBOL, the command is:

```
LIST %DEC('R1+2')
```

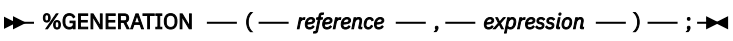
The Log window displays the value 16.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"LIST expression command" on page 149

## %GENERATION (PL/I)

Returns a specific generation of a controlled variable in your program.




***reference***
A controlled variable.

***expression***
The generation number *n* of a controlled variable *x*, where:

```
1 ≤ n ≤ ALLOCATION(x)
```

To return the oldest instance of *x*, specify:

```
%GENERATION(x,1)
```

To return the most recent instance of *x*, specify:

```
%GENERATION(x,ALLOCATION(x))
```

### Usage notes

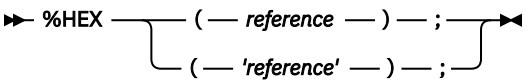
- You cannot use the %GENERATION built-in function while you replay recorded steps.
- If you want to use the %GENERATION built-in function while debugging Enterprise PL/I programs, you must apply the Language Environment PTF for APAR PK16316, if you are running on z/OS Version 1 Release 6 or Version 1 Release 7.

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
*IBM z/OS Debugger User's Guide*

## %HEX

Returns the hexadecimal value of an operand.




***reference***
One of the following:

- COBOL or PL/I reference
- C or C++ lvalue
- assembler, or disassembly

***'reference'***

Use the syntax of *'reference'* enclosed in apostrophes (') only for LangX COBOL programs. It is not supported in any other programming language.

**Examples**

C and C++: To display the internal representation of the packed decimal variable `zvar1` whose external representation is 235, enter the following command.

```
LIST %HEX(zvar1);
```

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 topics for more information related to the material discussed in this topic.

**Related references**

"LIST expression command" on page 149

# %INSTANCES (C, C++, and PL/I)

Returns the maximum value of %RECURSION (the most recent recursion number) for a given block.




***reference***

An automatic variable or a subroutine parameter. If necessary, you can use qualification to specify the variable.

%INSTANCES can be used like a z/OS Debugger variable.

**Usage notes**

z/OS Debugger 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.

  ```
  int RecFn(unsigned int i) {
    if (i == 0) {
      __ctest("");
  ```

  At this point, the `__ctest()` call gives control to z/OS Debugger, 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.

```
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.

```
int result;
result = remove("mayfile.dat");
if (result != 0)
  perror("could not delete file");
```

- z/OS Debugger 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.

```
double sqrtval;
sqrtval = sqrt(2);
```

- Nested function calls can be performed, as shown in the example below.

```
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.

```
fprintf(stdout, "value of errno is %d\n", errno);
```

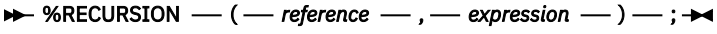
Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

"%RECURSION (C, C++, and PL/I)" on page 330

# %RECURSION (C, C++, and PL/I)

Returns a specific instance of an automatic variable or a parameter in a recursive procedure.




***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:

```
%RECURSION(x,1)
```

To return the most recent recursion of *x*, specify:

```
%RECURSION(x,%INSTANCES(x))
```

### Usage notes

- The higher the value of the expression, the more recent the generation of the variable z/OS Debugger references.
- %RECURSION can be used like a z/OS Debugger variable.
- You cannot use the %RECURSION built-in function while you replay recorded steps.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"%INSTANCES (C, C++, and PL/I)" on page 329

## %WHERE (assembler, disassembly, and LangX 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.




***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 LangX 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:

```
LIST %WHERE(X'14B0')||'ABC'
```

### Examples

- Assuming that CSECT ROUTINE1 is located at address X'1BC0400', the following command returns "ROUTINE1+X'2A'":

  ```
  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:

  ```
        USING   DATA1,R3
        …
        SLR   R0,R0
        …
  DATA1 DSECT ,
  …
  ```

  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 8. z/OS Debugger variables

z/OS Debugger reserves several variables for its own information. These z/OS Debugger variable names begin with a percent sign (%), to distinguish them from program variables. You can access z/OS Debugger variables while testing programs in any supported language.

You can use all z/OS Debugger 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.

```
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 z/OS Debugger variable, use the LIST command, as shown in the example below.

```
LIST %GPR15
```

The table below summarizes the z/OS Debugger variables.

| z/OS Debugger variable | Value |
|---|---|
| "%ADDRESS" on page 335 | Address of the location where your program was interrupted |
| "%AMODE" on page 335 | Current® AMODE of the suspended program |
| "%BLOCK" on page 335 | Name of the current block |
| "%CAAADDRESS" on page 336 | Address of the CAA control block associated with the suspended program |
| "%CC (assembler and disassembly only)" on page 336 | (Assembler and disassembly only) Condition code from current PSW |
| "%CONDITION" on page 336 | Name or number of the condition when z/OS Debugger is entered because of an AT OCCURRENCE |
| "%COUNTRY" on page 336 | Current country code |
| "%CU" on page 336 | Name of the primary entry point of the current compile unit |
| "%EPA" on page 336 | Address of the primary entry point in the current compile unit |
| "%EPRn or %EPRHn (%EPRHn assembler and disassembly only)" on page 336 | (Assembler, disassembly, C and C++, and PL/I only) Extended-precision floating-point registers |
| "%EPRBn (assembler and disassembly only)" on page 337 | (Assembler and disassembly only) Extended-precision floating-point registers in binary format |
| "%EPRDn (assembler and disassembly only)" on page 337 | (Assembler and Disassembly only) Extended-precision floating-point registers in decimal format |
| "%FPRn or %FPRHn (%FPRHn assembler and disassembly only)" on page 337 | Single-precision floating-point registers in hexadecimal format |

| z/OS Debugger variable | Value |
|---|---|
| "%FPRBn (assembler and disassembly only)" on page 338 | (Assembler and Disassembly only) Single-precision floating-point registers in binary format |
| "%FPRDn (assembler and disassembly only)" on page 338 | (Assembler and Disassembly only) Single-precision floating-point registers in decimal format |
| "%GPRn" on page 338 | 32-bit base General Purpose Registers at the point of interruption in a program |
| "%GPRGn" on page 339 | 64-bit General Purpose Registers at the point of interruption in a program |
| "%GPRHn" on page 339 | 32-bit high General Purpose Registers at the point of interruption in a program |
| "%HARDWARE" on page 340 | Type of hardware where the application is running |
| "%LINE or %STATEMENT" on page 340 | Current source line number |
| "%LOAD" on page 340 | Name of the load module of the current program, or an asterisk (*) |
| "%LPRn or %LPRHn (%LPRHn assembler and disassembly only)" on page 340 | Double-precision floating-point registers in hexadecimal format |
| "%LPRBn (assembler and disassembly)" on page 341 | (Assembler and Disassembly only) Double-precision floating-point registers in binary format |
| "%LPRDn (assembler and disassembly)" on page 341 | (Assembler and Disassembly only) Double-precision floating-point registers in decimal format |
| "%NLANGUAGE" on page 341 | National language currently in use |
| "%PATHCODE" on page 341 | Integer identifying the type of change occurring when the program flow reaches a point of discontinuity, and the path condition is raised |
| "%PLANGUAGE" on page 341 | Current programming language |
| "%PROGMASK (assembler and disassembly only)" on page 342 | (Assembler and disassembly only) Program mask from current PSW |
| %PROGRAM | Equivalent to %CU |
| "%PSW (assembler and disassembly only)" on page 342 | (Assembler and disassembly only) Current Program Status Word |
| "%RC" on page 342 | Return code from the most recent z/OS Debugger command |
| "%RSTDSETS" on page 342 | A value of 1 if user settings have been restored and 0 otherwise |
| "%RUNMODE" on page 342 | String identifying the presentation mode of z/OS Debugger |

| z/OS Debugger variable | Value |
|---|---|
| "%Rn" on page 342 | 32-bit base General Purpose Registers for the currently qualified assembler or disassembly CU |
| %STATEMENT | Equivalent to %LINE |
| "%SUBSYSTEM" on page 343 | Name of the underlying subsystem, if any, where the program is running |
| "%SYSTEM" on page 343 | Name of the operating system supporting the program |

You can access z/OS Debugger variables even when they have no intrinsic meaning in your operating system or language.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"Attributes of z/OS Debugger variables in different languages" on page 343

## %ADDRESS

Contains the address of the location where the program has been interrupted.

**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 z/OS Debugger 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, 31, or 64.

## %BLOCK

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.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"DESCRIBE command" on page 103
"LIST expression command" on page 149
"SET QUALIFY command" on page 250

## %CAAADDRESS

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.

## %CC (assembler and disassembly only)

Contains the condition code portion of the current PSW.

## %CONDITION

Contains the name or number of the condition when z/OS Debugger is entered because of an AT OCCURRENCE.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

"AT OCCURRENCE command" on page 65

## %COUNTRY

Contains the current country code.

## %CU

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.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

"SET QUALIFY command" on page 250

## %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.

**Usage note**

The value of %EPA is valid only in programs that adhere to standard linkage conventions for R13, R14, and R15.

## %EPRn or %EPRH*n* (%EPRH*n* assembler and disassembly only)

(%EPR0 , %EPR1, %EPR4, %EPR5, %EPR8, %EPR9, %EPR12, and %EPR13. %EPRH0 , %EPRH1, %EPRH4, %EPRH5, %EPRH8, %EPRH9, %EPRH12, and %EPRH13.)

Represent the extended-precision floating-point registers in hexadecimal format.

To modify one of these registers, assign a value to the associated %EPR*n* or %EPRH*n* variable.

%EPR*n* and %EPRH*n* cannot be used as the target of an assignment while debugging Enterprise PL/I programs.

You cannot use the %EPR*n* or %EPRH*n* variable while you are replaying recorded statements.

**Usage note**

For assembler and disassembly, the LIST %EPR*n* command displays values in hexadecimal but the LIST %EPRH*n* command displays values as hexadecimal floating point.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"Expression command (C and C++)" on page 116
"Assignment command (PL/I)" on page 36
"Assignment command (assembler and disassembly)" on page 33

## %EPRB*n* (assembler and disassembly only)

(%EPRB0 , %EPRB1, %EPRB4, %EPRB5, %EPRB8, %EPRB9, %EPRB12, and %EPRB13.)

Represent the extended-precision floating-point registers in binary format.

To modify one of these registers, assign a value to the associated %EPRB*n* variable.

If 64-bit hardware is not present, these variables are not supported. Any reference to them in such an environment will result in an "undefined symbol" message.

You cannot use the %EPRB*n* variable while you are replaying recorded statements.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"Assignment command (assembler and disassembly)" on page 33

## %EPRD*n* (assembler and disassembly only)

(%EPRD0 , %EPRD1, %EPRD4, %EPRD5, %EPRD8, %EPRD9, %EPRD12, and %EPRD13.)

Represent the extended-precision floating-point registers in decimal format.

To modify one of these registers, assign a value to the associated %EPRD*n* variable.

If both Decimal Floating Point and 64-bit hardware are not present, these variables are not supported. Any reference to them in such an environment will result in an "undefined symbol" message.

You cannot use the %EPRD*n* variable while you are replaying recorded statements.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"Assignment command (assembler and disassembly)" on page 33

## %FPR*n* or %FPRH*n* (%FPRH*n* assembler and disassembly only)

(%FPR0, %FPR1, %FPR2, %FPR3, %FPR4, %FPR5, %FPR6, %FPR7, %FPR8, %FPR9, %FPR10, %FPR11, %FPR12, %FPR13, %FPR14, and %FPR15. %FPRH0, %FPRH1, %FPRH2, %FPRH3, %FPRH4, %FPRH5, %FPRH6, %FPRH7, %FPRH8, %FPRH9, %FPRH10, %FPRH11, %FPRH12, %FPRH13, %FPRH14, and %FPRH15.)

Represent single-precision floating-point registers in hexadecimal format.

To modify one of these registers, assign a value to the associated %FPR*n* or %FPRH*n* variable.

%FPR*n* and %FPRH*n* cannot be used as the target of an assignment while debugging Enterprise PL/I programs.

**Usage note**

For assembler and disassembly, the LIST %FPR*n* command displays values in hexadecimal, but for the LIST %FPRH*n* command, values are listed as hexadecimal floating point.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"Expression command (C and C++)" on page 116
"MOVE command (COBOL)" on page 175
"Assignment command (PL/I)" on page 36
"Assignment command (assembler and disassembly)" on page 33

## %FPRB*n* (assembler and disassembly only)

(%FPRB0, %FPRB1, %FPRB2, %FPRB3, %FPRB4, %FPRB5, %FPRB6, %FPRB7, %FPRB8, %FPRB9, %FPRB10, %FPRB11, %FPRB12, %FPRB13, %FPRB14, and %FPRB5.)

Represent single-precision floating-point registers in binary format.

To modify one of these registers, assign a value to the associated %FPRB*n* variable.

If 64-bit hardware is not present, these variables are not supported. Any reference to them in such an environment will result in an "undefined symbol" message.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"Assignment command (assembler and disassembly)" on page 33

## %FPRD*n* (assembler and disassembly only)

(%FPRD0, %FPRD1, %FPRD2, %FPRD3, %FPRD4, %FPRD5, %FPRD6, %FPRD7, %FPRD8, %FPRD9, %FPRD10, %FPRD11, %FPRD12, %FPRD13, %FPRD14, and %FPRD15.)

Represent single-precision floating-point registers in decimal format.

To modify one of these registers, assign a value to the associated %FPRD*n* variable.

If both Decimal Floating Point and 64-bit hardware are not present, these variables are not supported. Any reference to them in such an environment will result in an "undefined symbol" message

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"Assignment command (assembler and disassembly)" on page 33

## %GPRn

(%GPR0 to %GPR15.)

Represent 32-bit 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.

**Usage notes**

- If you modify a %GPRn register, the change is reflected when you resume program execution.
- Do not modify base registers.
- %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++:

```
%GPR15=name_table;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“Expression command (C and C++)” on page 116
“MOVE command (COBOL)” on page 175
“Assignment command (assembler and disassembly)” on page 33
“Assignment command (PL/I)” on page 36
“Assignment command (LangX COBOL)” on page 35

## %GPRGn

(%GPRG0 to %GPRG15.)

Represent 64-bit General Purpose Registers at the point of interruption in a program.

To modify one of these registers, assign a value to the associated %GPRG*n* variable.

**Usage notes**

- If you modify a %GPRG*n* register, the change is reflected when you resume program execution.
- In disassembly and assembler, you can replace GR*n* with %GPR*n*.
- If your program is running on hardware that does not support 64-bit instructions or your program is suspended at a point where the high-half of the General Purpose Registers are not available, these variables are treated as undefined symbols.

**Examples**

Assembler:

```
LIST %GPRG0;
%GPRG0 = 12;
```

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
“Assignment command (assembler and disassembly)” on page 33
“Assignment command (LangX COBOL)” on page 35

## %GPRHn

(%GPRH0 to %GPRH15.)

Represent 32-bit high General Purpose Registers at the point of interruption in a program.

To modify one of these registers, assign a value to the associated %GPRH*n* variable.

**Usage note**

If you modify a %GPRH*n* register, the change is reflected when you resume program execution.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

"Expression command (C and C++)" on page 116
"MOVE command (COBOL)" on page 175
"Assignment command (assembler and disassembly)" on page 33
"Assignment command (PL/I)" on page 36
"Assignment command (LangX COBOL)" on page 35

## %HARDWARE

Identifies the type of hardware where the application program is running. A possible value is: 370/ESA.

## %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.

## %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 (*).

z/OS Debugger 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[3], z/OS Debugger does not recognize load modules that have been loaded by the MVS `LOAD` service.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

"SET QUALIFY command" on page 250

## %LPR*n* or %LPRH*n* (%LPRH*n* assembler and disassembly only)

(%LPR0, %LPR1, %LPR2, %LPR3, %LPR4, %LPR5, %LPR6, %LPR7, %LPR8, %LPR9, %LPR10, %LPR11, %LPR12, %LPR13, %LPR14, and %LPR15. %LPRH0, %LPRH1, %LPRH2, %LPRH3, %LPRH4, %LPRH5, %LPRH6, %LPRH7, %LPRH8, %LPRH9, %LPRH10, %LPRH11, %LPRH12, %LPRH13, %LPRH14, and %LPRH15.)

Represent the double-precision floating-point registers in hexadecimal format.

To modify one of these registers, assign a value to the associated %LPR*n* or %LPRH*n* variable.

%LPR*n* cannot be used as the target of an assignment while debugging Enterprise PL/I programs.

[3] See the *IBM z/OS Debugger Customization Guide* for instructions on how to control SVC screening.

**Usage note**

For assembler and disassembly, the `LIST %LPR`*n* command displays values in hexadecimal, but for the `LIST %LPRH`*n* command, values are listed as hexadecimal floating point.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

"Expression command (C and C++)" on page 116
"MOVE command (COBOL)" on page 175
"Assignment command (PL/I)" on page 36
"Assignment command (assembler and disassembly)" on page 33

## %LPRB*n* (assembler and disassembly)

(%LPRB0, %LPRB1, %LPRB2, %LPRB3, %LPRB4, %LPRB5, %LPRB6, %LPRB7, %LPRB8, %LPRB9, %LPRB10, %LPRB11, %LPRB12, %LPRB13, %LPRB14, and %LPRB15.)

Represent the double-precision floating-point registers in binary format.

To modify one of these registers, assign a value to the associated %LPRB*n* variable.

If 64-bit hardware is not present, these variables are not supported. Any reference to them in such an environment will result in an "undefined symbol" message.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

"Assignment command (assembler and disassembly)" on page 33

## %LPRD*n* (assembler and disassembly)

(%LPRD0, %LPRD1, %LPRD2, %LPRD3, %LPRD4, %LPRD5, %LPRD6, %LPRD7, %LPRD8, %LPRD9, %LPRD10, %LPRD11, %LPRD12, %LPRD13, %LPRD14, and %LPRD15.)

Represent the double-precision floating-point registers in decimal format.

To modify one of these registers, assign a value to the associated %LPRD*n* variable.

If both Decimal Floating Point and 64-bit hardware are not present, these variables are not supported. Any reference to them in such an environment will result in an "undefined symbol" message.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

"Assignment command (assembler and disassembly)" on page 33

## %NLANGUAGE

Indicates the national language currently in use: `ENGLISH`, `UENGLISH`, `JAPANESE`, or `KOREAN`.

## %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.

## %PLANGUAGE

Indicates the programming language currently in use.

%PLANGUAGE returns C for both C and C++.

## %PROGMASK (assembler and disassembly only)

Contains the program mask portion of the current PSW.

## %PROGRAM

Contains the name of the primary entry point of the current program.

%PROGRAM is equivalent to %CU. See "%CU" on page 336 for more information.

## %PSW (assembler and disassembly only)

Contains the current Program Status Word.

## %RC

Contains a return code whenever a z/OS Debugger 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 z/OS Debugger variable. It is not related to the return code that can be found in Register 15.

## %RSTDSETS

Contains a value of 1 if the user settings have been restored as a result of the `SET RESTORE SETTINGS AUTO` command or the `RESTORE SETTINGS` command or a value of 0 otherwise.

**Usage note**

You can use this variable as part of an %IF statement in a preferences or commands file to avoid modifying SET values that have been restored.

## %RUNMODE

Contains a string identifying the presentation mode of z/OS Debugger. The possible values are listed below.

```
LINE
SCREEN
BATCH
```

## %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
```

## %SYSTEM

Contains the name of the operating system supporting the program. The only possible value is MVS.

## Attributes of z/OS Debugger variables in different languages

The table below shows the attributes for z/OS Debugger variables when used with different programming languages.

| z/OS Debugger variable | C and C++ attributes | COBOL attributes | PL/I attributes | Assembler/ disassembly attributes |
|---|---|---|---|---|
| %ADDRESS | void * | USAGE POINTER | POINTER | A |
| %AMODE | signed short int | PICTURE S9(4) USAGE COMP | FIXED BINARY(15,0) | H |
| %BLOCK | unsigned char[] | PICTURE X(j) | CHARACTER(j) | CLj |
| %CAAADDRESS | void * | USAGE POINTER | POINTER | A |
| %CC | n/a | n/a | n/a | H |
| %CONDITION | unsigned char[] | PICTURE X(j) | CHARACTER(j) | CLj |
| %COUNTRY | unsigned char[] | PICTURE X(j) | CHARACTER(j) | CLj |
| %CU | unsigned char[] | PICTURE X(j) | CHARACTER(j) | CLj |
| %EPA | void * | USAGE POINTER | POINTER | A |
| %EPRn | long double | n/a | FLOAT DECIMAL(33) | L |

| z/OS Debugger variable | C and C++ attributes | COBOL attributes | PL/I attributes | Assembler/ disassembly attributes |
|---|---|---|---|---|
| %EPRBn | n/a | n/a | n/a | LB |
| %EPRDn | n/a | n/a | n/a | LD |
| %EPRHn | n/a | n/a | n/a | LH |
| %FPRn | float | USAGE COMP-1 | FLOAT DECIMAL(6) | E |
| %FPRBn | n/a | n/a | n/a | EB |
| %FPRDn | n/a | n/a | n/a | ED |
| %FPRHn | n/a | n/a | n/a | EH |
| %GPRn | signed int | PICTURE S9(9) | FIXED BINARY(31,0) | F |
| %GPRGn | n/a | n/a | n/a | FD |
| %GPRHn | signed int | PICTURE S9(9) | FIXED BINARY(31,0) | F |
| %HARDWARE | unsigned char[] | PICTURE X(j) | CHARACTER(j) | CLj |
| %LINE or %STATEMENT | unsigned char[] | PICTURE X(j) | CHARACTER(j) | CLj |
| %LOAD | unsigned char[] | PICTURE X(j) | CHARACTER(j) | CLj |
| %LPRn | double | USAGE COMP-2 | FLOAT DECIMAL(16) | D |
| %LPRBn | n/a | n/a | n/a | DB |
| %LPRDn | n/a | n/a | n/a | DD |
| %LPRHn | n/a | n/a | n/a | DH |
| %NLANGUAGE | unsigned char[] | PICTURE X(j) | CHARACTER(j) | CLj |
| %PATHCODE | signed short int | PICTURE S9(4) USAGE COMP | FIXED BINARY(15,0) | H |
| %PLANGUAGE | unsigned char[] | PICTURE X(j) | CHARACTER(j) | CLj |
| %PROGMASK | n/a | n/a | n/a | H |
| %PROGRAM | unsigned char[] | PICTURE X(j) | CHARACTER(j) | CLj |
| %PSW | n/a | n/a | n/a | CL8 |
| %RC | signed short int | PICTURE S9(4) USAGE COMP | FIXED BINARY(15,0) | H |
| %RSTDSETS | signed int | PICTURE S9(9) USAGE COMP | FIXED BINARY(31,0) | F |
| %RUNMODE | unsigned char[] | PICTURE X(j) | CHARACTER(j) | CLj |
| %R*n* | n/a | n/a | n/a | F |
| %SUBSYSTEM | unsigned char[] | PICTURE X(j) | CHARACTER(j) | CLj |
| %SYSTEM | unsigned char[] | PICTURE X(j) | CHARACTER(j) | CLj |

# Chapter 9. z/OS Debugger messages

All messages that 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 EQAnnnnx, where EQA indicates that the message is an z/OS Debugger message, nnnn 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 z/OS Debugger attempted to fix.

**E**
An ***error*** message describes an error that z/OS Debugger 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 z/OS Debugger from continuing.

**Symbols in messages**

Many of the z/OS Debugger 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 z/OS Debugger located on the host notifies you of errors associated with debugging functions carried out by the host.

Refer to the following topics for more information related to the material discussed in this topic.

**Related tasks**
*z/OS Language Environment Programming Guide*

**Related references**
"Allowable comparisons for the IF command (COBOL)" on page 132
"Allowable moves for the MOVE command (COBOL)" on page 177

**EQA1000I TEST (*cu_name* initialization):**

## Explanation

z/OS Debugger is ready to accept a command from the terminal. This message is used in line mode when an initial prompt occurs after z/OS Debugger 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, qualify, 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

Asking for confirmation to terminate debug session. Enter Y for YES or N for NO.

**EQA1012I Enter Y for YES and N for NO**

**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.

**EQA1014I Current command is incomplete, enter more input below.**

## Explanation

This informational message is displayed while entering a command continuation character or a block of commands. Complete the command in the Command pop-up window.

**EQA1015I Source window is closed; FINDBP not performed.**

## Explanation

The Source window must be open when a FINDBP command is issued.

**EQA1016I The following errors were detected processing the run-time EQAOPTS specifications**

## Explanation

Errors were detected processing the run-time EQAOPTS commands. Each command containing an error will be listed, followed by the applicable error message.

**EQA1017I *EqaOpts-Cmd* An EqaOpts command.**

## Explanation

Errors were detected processing the run-time EQAOPTS commands. Each command containing an error will be listed, followed by the applicable error message.

**EQA1018I The following EQAOPTS were specified at run-time.**

## Explanation

The EQAOPTS specifications shown following this message were specified at run-time.

**EQA1019I The following EQAOPTS were obtained from the EQAOPTS load module.**

## Explanation

The EQAOPTS specifications shown following this message were obtained from the EQAOPTS load module.

**EQA1020E** **A close-quote in operand *column-number* was not followed by a blank or a comma.**

## Explanation

A close-quote indicates the end of an operand. Therefore, it must be followed by a comma to indicate that another operand is present or by a blank to indicate that this is the last operand.

**EQA1021E** **A continuation was not found.**

## Explanation

The last line contained a non-blank character in column 72 indicating a continued statement but no continuation line was found.

**EQA1022E** **The last operand ended in a comma but column 72 is blank.**

## Explanation

The last operand on the current line ended with a comma but a non-blank was not present in column 72 to indicate a continuation.

**EQA1023E** **The specified numeric value of operand *operand-number* is not within the expected range. It is either too small or too large.**

## Explanation

The specified number is outside of the range allowed for the indicated operand.

**EQA1024E** **Not enough operands were specified.**

## Explanation

Some of the operands required for this EQAOPTS function were not specified.

**EQA1025E** **More operands were specified than are allowed for this EQAOPTS function.**

## Explanation

Some of the operands required for this EQAOPTS function were not specified.

**EQA1026E** **EQAXOPT opcode was not found.**

## Explanation

All EQAOPTS functions must contain the EQAXOPT opcode.

**EQA1027E** **A value is not allowed for operand *operand-number*.**

## Explanation

The indicated operand keyword was followed by an '=' and an operand. However, this operand does not allow the specification of a value.

**EQA1028E** **No value was specified for operand *operand-number*.**

## Explanation

The indicated operand requires a value to be specified but none was found.

**EQA1029E** **Internal error (invalid operand type) in operand *operand-number*.**

## Explanation

An unexpected situation was encountered. Contact z/OS Debugger support.

**EQA1030I** **PENDING:**

## Explanation

z/OS Debugger 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** **The partially parsed command is:**

## Explanation

The explanation of a command was requested or a command was determined to be in error.

## Programmer response

Determine the cause of the error and reenter the command.

**EQA1032I** **The next word can be one of:**

## Explanation

This title line will be followed by message 1015.

**EQA1033I** ***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.

**EQA1034E** **The MONITOR LIMIT must be greater than or equal to 1000.**

## Explanation

The MONITOR LIMIT specified in the `SET MONITOR LIMIT` *n* command must be greater than or equal to 1000.

**EQA1035E** **A non-decimal digit was found in the numeric value for operand *operand-number*.**

## Explanation

The indicated operand expects a numeric value. However, a character other than a decimal digit was found in the value.

**EQA1036E** **The first operand is not a valid EQAOPTS function.**

## Explanation

The first operand must be a valid EQAOPTS function name.

**EQA1037E** **EQAXOPT END is not the last command.**

## Explanation

The 'EQAXOPT END' command must be the last command entered.

**EQA1038E** **Incorrect use of quotes in operand *operand-number*.**

## Explanation

Quotes were incorrectly used in the indicated operand. If you want to include a quote or an ampersand within a quoted string, a pair of quotes or ampersands must be specified.

**EQA1039E** **Column 1 to 15 of a continuation line are non-blank.**

## Explanation

Continuation lines must begin in column 16. Make sure that you did not begin the continuation line before column 15.

**EQA1040E** **Operand *operand-number* is too long.**

## Explanation

The indicated operand is longer than the maximum length allowed.

**EQA1041E** **Internal error (more than one value for a function) in operand *operand-number*.**

## Explanation

An unexpected situation was encountered. Contact z/OS Debugger support.

**EQA1042E** **Unmatched parenthesis were found in operand *operand-number*.**

## Explanation

A left parenthesis was found without a corresponding right parenthesis or vice-versa.

**EQA1043E** **An asterisk was specified in operand *opernand-number* but is not allowed in that operand.**

## Explanation

An asterisk was detected in an operand that does not allow an asterisk to be specified.

**EQA1044E** **An unrecognized keyword was specified in operand *operand-number*.**

## Explanation

An unrecognized keyword was specified for the indicated operand.

**EQA1045E** **Return code *RC* and reason code *Reason* were encountered while obtaining the run-time EQAOPTS specifications. The run-time EQAOPTS were not processed.**

## Explanation

The return code value indicates the function being processed as follows: 1 = Allocating memory, 2 = Obtaining data set attributes, 3 = Opening data set, 4 = Reading data set, and 5 = Closing data set. The specified reason code indicates the specific error encountered by this function.

**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.

**EQA1047I** **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** **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 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 z/OS Debugger. 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.

**EQA1062E** **The specified load module does not contain CU *CU_name*.**

## Explanation

The specified load module is known to z/OS Debugger but it does not contain a CU by the specified name.

**EQA1063E** **A load module name was not specified for CU *CU_name*. Both load module and CU names must be specified in the form LOADMOD::>CU.**

## Explanation

When an LDD is entered in Explicit Debug Mode, both load module and CU names must be specified.

**EQA1064I** **Previous value in memory:**

## Explanation

Shows the previous content in memory for area being watched.

**EQA1065I** **Current value in memory:**

## Explanation

Shows the current content in memory for area being watched.

**EQA1066I** **The current setting for the Level 88 variable is True.**

## Explanation

Shows the current setting for a Level 88 variable.

**EQA1067I** **The current setting for the Level 88 variable is False.**

## Explanation

Shows the current setting for a Level 88 variable.

**EQA1076I** ***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.

**EQA1077I** ***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.

**EQA1078I** ***Direction Place*** **in** ***PLANG CU***

## Explanation

CU name of the call chain. The message is issued as a result of the LIST CALLS command.

**EQA1079I** ***Direction*** **address** ***Address*** **in** ***PLANG CU***

## Explanation

The compile unit was compiled without the TEST option and is in the z/OS Debugger list of CUs.

**EQA1080E** **This EQAXOPT command must be entered in the EQAOPTS load module.**

## Explanation

The indicated command cannot be specified at run-time.

**EQA1081E** **The EQAOPTS data set, "*DSName*", cannot be opened or cannot be read.**

## Explanation

An error occurred trying to OPEN or READ the EQAOPTS data set.

**EQA1086I** **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.

**EQA1090I** **The compiler data for program** ***cu_name*** **is**

## Explanation

This is the title line for the DESCRIBE PROGRAM command.

**EQA1091I** **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

Indicate 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.)

**EQA1114I** **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.

**EQA1115I** **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.

**EQA1116I** **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.

**EQA1117I** **PL/I Math library is being used**

## Explanation

This is the response for the DESCRIBE ENVIRONMENT command when the PL/I math library is being used.

**EQA1118I** ***compile option* (*compile suboption*, *compile suboption*, *compile suboption*)**

## Explanation

Used to display a compile option with three parameters, for example TEST(ALL,SYM,SEPARATE)

**EQA1119I** **Current allocations:**

## Explanation

Heading line for DESCRIBE ALLOCATIONS output.

**EQA1120I** **VOLUME CAT DISP OPEN DDNAME DSNAME**

## Explanation

Header for DESCRIBE ALLOCATIONS output.

**EQA1121I** **------ --- ---------- ---- -------- ------------------------------ --------------**

## Explanation

Header for DESCRIBE ALLOCATIONS output.

**EQA1122I** ***allocation description***

## Explanation

Description of the current allocation.

**EQA1123I** **Insufficient storage is available to process command.**

## Explanation

There was not enough main memory available to process the command. This may occur when attempting to load a source listing, debug data, etc.

**EQA1124I** **Return code *return code* /reason code *reason code* from *macro name* macro invocation.**

## Explanation

During the processing of the command, the indicated macro invocation failed with the indicated return and reason codes.

**EQA1125I** **ALLOCATE / FREE failed. Return code *return code* /reason code *reason code* from dynamic allocation.**

## Explanation

The dynamic allocation failed with the indicated return and reason codes.

**EQA1126I** **ALLOCATE / FREE failed. Dataset *dsname* was not found.**

## Explanation

The indicated data set was not cataloged or was not found on the volume on which it was cataloged.

**EQA1127I** **ALLOCATE / FREE failed. The FILE specified was already in use (ALLOCATE) or was not allocated (FREE).**

## Explanation

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.

**EQA1128I** **ALLOCATE / FREE failed. Dataset *dsame* is already allocated to another JOB or USER.**

## Explanation

The specified data set is already allocated in such a way that it cannot be allocated with the specified disposition.

**EQA1129I** **Command not supported on the current platform.**

## Explanation

The command is not supported in the current environment (such as CICS)

**EQA1130I** **The EQALANGX debug data also contains data for the following CUs:**

## Explanation

This is the header used to display the additional CSECT's included in the EQALANGX data for the current CU.

**EQA1131I** ***CU name CU language***

## Explanation

Used to display CSECT's also included in the EQALANGX data for the current CU.

**EQA1132I** **EQALANGX version for this CU: *EQALANGX_version***

## Explanation

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.

**EQA1133I** **Current EQALANGX version: *EQALANGX_version***

## Explanation

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 z/OS Debugger.

**EQA1134I** **The INCLUDE files in this program are indexed as follows: *INCLUDE_file_names*.**

## Explanation

The names of the INCLUDE files contained by the program are displayed under this title line.

**EQA1139I** ******* PREVIOUS STATEMENT *******

## Explanation

Automonitor Previous Statement area in the Monitor window.

**EQA1140I** ***character***

## Explanation

This message is used to produce output for `LIST ( ... )`.

**EQA1141I** ***expression name* = *expression value***

## Explanation

This message is used to produce output for `LIST TITLED ( ... )` when an expression is a scalar.

**EQA1142I** ***expression element***

## Explanation

This insert is used for naming the expression for expression element.

**EQA1143I** **>>> EXPRESSION ANALYSIS <<<**

## Explanation

First line of output from the `ANALYZE EXPRESSION` command.

**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.

**EQA1166I** **There are no known session procedures.**

## Explanation

A LIST NAMES PROCEDURES was issued but no session procedures exist.

**EQA1167I** ***register name* = *register value***

## Explanation

Used when listing registers.

**EQA1168I** **No LIST STORAGE data is available for the requested reference or address.**

## Explanation

The given reference or address is invalid.

**EQA1169I** **No frequency data is available**

## Explanation

This message is issued upon failure to find frequency information.

**EQA1170I** **Frequency of verb executions in *cu_name***

## Explanation

This is the header produced by the LIST FREQUENCY command.

**EQA1171I** ***character string* = *count***

## Explanation

This is the frequency count produced by the LIST FREQUENCY command.

**EQA1172I** **TOTAL VERBS= *total statements*, TOTAL VERBS EXECUTED= *total statements executed*, PERCENT EXECUTED= *percent executed***

## Explanation

This is the trailer produced by the LIST FREQUENCY command.

**EQA1173I** **(*history number*) ENTRY hook for *cu_name***

## Explanation

This is a LIST HISTORY message.

**EQA1174I** **(*history number*) ENTRY hook for block *block name* in *cu_name***

## Explanation

This is a LIST HISTORY message.

**EQA1175I** **(*history number*) EXIT hook for *cu_name***

## Explanation

This is a LIST HISTORY message.

**EQA1176I** **(*history number*) EXIT hook for block *block name* in *cu_name***

## Explanation

This is a LIST HISTORY message.

**EQA1177I** **(*history number*) STATEMENT hook at statement *cu_name* :> *statement_id***

## Explanation

This is a LIST HISTORY message.

**EQA1178I** **(*history number*) PATH hook at statement *cu_name* :> *statement_id***

## Explanation

This is a LIST HISTORY message.

**EQA1179I** **(*history number*) Before CALL hook at statement *cu_name* :> *statement_id***

## Explanation

This is a LIST HISTORY message.

**EQA1180I** **(*history number*) CALL CEETEST at statement *cu_name* :> *statement_id***

## Explanation

This is a LIST HISTORY message.

**EQA1181I** **(*history number*) Waiting for program input from *ddname***

## Explanation

This is a LIST HISTORY message.

**EQA1182I** **(*history number*) LOAD occurred at statement *cu_name* :> *statement_id***

## Explanation

This is a LIST HISTORY message.

**EQA1183I** **(*history number*) DELETE occurred at statement *cu_name* :> *statement_id***

## Explanation

This is a LIST HISTORY message.

**EQA1184I** **(*history number*) *condition name* raised at statement *cu_name* :> *statement_id***

## Explanation

This is a LIST HISTORY message.

**EQA1185I** **(*history number*) LABEL hook at statement *cu_name* :> *statement_id***

## Explanation

This is a LIST HISTORY message.

**EQA1186I** **Unable to display value of *variable name*. Use LIST (*variable name*) for further details**

## Explanation

This is used to inform the user that for some reason one of the variables cannot be displayed for LIST TITLED.

**EQA1187I** **There are no data members in the requested object.**

## Explanation

The requested object does not contain any data members. It contains only methods.

**EQA1188I** **(*history number*) DATE hook at statement *cu_name* :> *statement_id***

## Explanation

This is a LIST HISTORY message.

**EQA1189I** **There are no CUs compiled with debug data. To see the CU names, issue SET DISASSEMBLY ON before LIST command.**

## Explanation

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

**EQA1190I** **Unable to update the requested location.**

## Explanation

The given reference or address is invalid.

**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.

**EQA1195I** **Its linkage is Pre-Language Environment OS/Standard**

## Explanation

This compile unit uses OS/Standard linkage and was generated by a pre-Language Environment compiler.

**EQA1196I** **Its linkage is Language Environment FastLink**

## Explanation

This compile unit uses Language Environment FastLink linkage.

**EQA1197I** **Its linkage is Language Environment OS/Standard**

## Explanation

This compile unit uses Language Environment OS/Standard Linkage.

**EQA1198I** **Its linkage is OS/Standard**

## Explanation

This non-Language Environment compile unit uses OS/Standard linkage.

**EQA1199I** **Its linkage is Language Environment XPLINK precision-subtype**

## Explanation

This compile unit uses Language Environment 31 or 64-bit XPLink linkage. The -1, -2, etc. indicates the type of Language Environment header found.

**EQA1200I** **Code Coverage: *status* in *CU-name*.**

## Explanation

This is the header produced by the `LIST CC` command.

**EQA1201I** ***statement_id status***

## Explanation

This shows the execution *status* (x for executed, blank for not-executed) for *statement_id*.

**EQA1202I** **No code coverage data is available.**

## Explanation

This message is issued upon failure to find code coverage information.

**EQA1203I** **The Source/Listing is not available because this program is not in the current enclave.**

## Explanation

The source or listing is not available for this program because it is not in the current enclave.

**EQA1204I** **The source/listing for this program is not available.**

## Explanation

Source or Listing data is not available, or the CU was not compiled with the correct compile options.

**EQA1210I** **The `MONITOR` command has already been established. The duplicate is ignored.**

## Explanation

The `MONITOR` command entered is a duplicate of a monitor command that was previously entered.

**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 executing 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** ***LanguageCode* 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* at address *CU-address*.**

## 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 SET.

**EQA1246I** ***PFKEY string command***

## Explanation

Used to display PFKEYS as part of QUERY PKFEYS.

**EQA1247I** **COLOR *color hilight intensity colored area***

## Explanation

Used to display SCREEN as part of QUERY SCREEN.

**EQA1248I** **You were prompted because STEP ended.**

## Explanation

Shows the bearings in an interrupted program.

**EQA1249I** ***character string* - The QUERY source setting file name is not available.**

## Explanation

The source listing name is not available. The source listing was not required or set before this command.

**EQA1250I** **SET INTERCEPT is already set *on or off* for FILE *filename*.**

## Explanation

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.

**EQA1251I** **You were prompted because RUNTO ended.**

## Explanation

The program has stopped because RUNTO cursor/ statement command reached the cursor location or pointed statement number.

**EQA1252I** **********  AUTOMONITOR  **********

## Explanation

Header for the automonitor area in the monitor window.

**EQA1253I** **You were prompted because Playback replay mode ended.**

## Explanation

A PLAYBACK STOP command was processed, which terminated Playback replay mode.

**EQA1254I** **The LOADDEBUGDATA command was not processed.**

## Explanation

An error occurred so the LLD command was not processed.

**EQA1255E** **The CU specified for the LOADDEBUGDATA command is not a disassembly CU.**

## Explanation

Only a disassembly CU can be identified as assembler CU.

**EQA1256E** **An error occurred while attempting to load the debug (EQALANGX) file for a specified CU.**

## 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.

**EQA1257E** **The MONITOR parameter HEX or DEFAULT is valid only for LIST reference command.**

## Explanation

MONITOR n HEX or MONITOR n DEFAULT are valid only if n represents LIST reference command.

**EQA1258E** **There is no MONITOR LIST command with the referenced integer.**

## Explanation

MONITOR n HEX or MONITOR n DEFAULT are valid only if n represents valid LIST command.

**EQA1259I** **The LOADDEBUGDATA command for *CU_name* has been deferred until the CU appears.**

## Explanation

The indicated CU is not currently known to z/OS Debugger. The LOADDEBUGDATA will be executed when the CU appears in a loaded module.

**EQA1260I** **The CU specified for the LOADDEBUGDATA command is already an assembler or LangX COBOL CU.**

## Explanation

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.

**EQA1261I** **The requested view is already active.**

## Explanation

A CHANGEVIEW command requested a view that is already the active view for the currently qualified CU.

**EQA1262I** **The requested view is not supported by the currently qualified CU.**

## Explanation

A CHANGEVIEW command requested a view that not supported by the programming language of the currently qualified CU.

**EQA1263W** **Breakpoints have been set on statements that would be suppressed in the requested view. The view is not changed.**

## Explanation

z/OS Debugger does not currently support changing to a view that would suppress statements that currently contain breakpoints.

**EQA1264I** **The current view is *Current_View*.**

## Explanation

This message is displayed in response to the `QUERY CURRENT VIEW` command.

**EQA1265E** **Command failed due to an internal communications error.**

## Explanation

The previous command could not be completed because of an internal communications error.

**EQA1266I** **You were prompted because JUMPTO ended.**

## Explanation

The program has stopped because JUMPTO command reached the cursor location or pointed statement number.

**EQA1267I** ******* Previous Statement *CU-name* :> *statement id* *******

## Explanation

Automonitor Previous Statement area in the monitor window.

**EQA1268I** **The previous statement is out of scope. No variables can be displayed.**

## Explanation

The data from the statement in which z/OS Debugger last had control cannot be resolved because the statement is no longer in scope.

**EQA1269I** **The previous location is *CU-name* :> *statement id* .**

## Explanation

Shows the place in the program where the program was previously interrupted.

**EQA1270I** ******* AUTOMONITOR - PREVIOUS *CU-name* :> *statement id* *******

## Explanation

Header for the automonitor area in the Monitor window when PREVIOUS is specified.

**EQA1271I** **The program previously entered block *block name*.**

## Explanation

Shows the location where z/OS Debugger previously had control.

**EQA1272I** **The program previously exited block *block name*.**

## Explanation

Shows the bearings in an interrupted program.

**EQA1273I** **The program previously executed prolog code for *block name*.**

## Explanation

Shows the bearings in an interrupted program.

**EQA1274I** ******* AUTOMONITOR *CU-name* :> *statement id* *******

## Explanation

Header with location for the automonitor area in the monitor window.

**EQA1275I** ************ AUTOMONITOR - PREVIOUS ************

## Explanation

Header without location information for the automonitor area in the monitor window when PREVIOUS is specified. Used when no location information is available.

**EQA1276I** **TEST:**

## Explanation

z/OS Debugger is ready to accept a command from the terminal.

## 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.

**EQA1277I** **The USE file is empty.**

## Explanation

z/OS Debugger tried to read commands from an empty USE file. If unintentional, this could be because of an incorrect file specification.

## Programmer response

Correct the file specification and retry.

**EQA1278I** ***alignment spaces command part***

## 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 z/OS Debugger.

- 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.

**EQA1279I** **TEST (*cu_name*:> *statement_id*):**

## Explanation

z/OS Debugger 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.

**EQA1280I** **TEST (*cu_name* ENTRY):**

## Explanation

z/OS Debugger 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.

**EQA1281I** **TEST (*cu_name*:> *block name* ENTRY):**

## Explanation

z/OS Debugger 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.

**EQA1282I** **TEST (*cu_name* EXIT):**

## Explanation

z/OS Debugger 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.

**EQA1283I** **TEST (*cu_name*:> *block name* EXIT):**

## Explanation

z/OS Debugger 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.

**EQA1284I** **TEST (Application program has terminated):**

## Explanation

z/OS Debugger is ready to accept a command from the terminal. This message is used in line mode when an initial prompt occurs at the termination of the application program.

## 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.

**EQA1285I** **TEST (*label-name* LABEL);**

## Explanation

z/OS Debugger 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.

**EQA1286I** **(Application program has terminated)**

## Explanation

z/OS Debugger 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.

**EQA1287I** **Unknown**

## Explanation

z/OS Debugger 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.

**EQA1288I** **initialization**

## Explanation

z/OS Debugger is ready to accept a command from the terminal. This message is used in full-screen mode when an initial prompt occurs after z/OS Debugger initialization and before any program hooks are reached.

**EQA1289I** ***ddname*: *program output***

## Explanation

Displays program output with the ddname preceding the output.

**EQA1290I** **The program is waiting for input from *ddname***

## Explanation

z/OS Debugger has gained control because the program is waiting for input.

**EQA1291I** **Use the INPUT command to enter *recsize* characters for the intercepted fixed-format file.**

## Explanation

Prompts you for intercepted input of fixed-format file.

**EQA1292I** **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

z/OS Debugger 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 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.

**EQA1294I** **The suspended LOCAL MONITOR commands are:**

## Explanation

This is the header produced by `LIST MONITOR` when suspended local monitors are present.

**EQA1295I** **A System 0C1 Abend occurred at an address that appears to be the target of a BALR/BASR 14,15. A CALL to an invalid address is assumed. The current location is positioned to the CALL statement.**

## Explanation

A System 0C1 Abend occurred at an address that is not in any known Compile Unit. This address matches the address in GPR 15 and GPR14 points to the instruction immediately following a BALR 14,15 or a BASR 14,15. The current location is assumed to be the BALR or BASR instruction.

**EQA1301I** **A *ABEND code* ABEND occurred.**

## Explanation

A system or user ABEND has been detected by z/OS Debugger.

**EQA1306I** **You were prompted because the *CONDITION name* condition was raised in your program.**

## Programmer response

The program has stopped running due to the occurrence of the named condition.

**EQA1307I** **You were prompted because an ATTENTION interrupt occurred.**

## Explanation

The attention request from the terminal was recognized and the z/OS Debugger 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.

**EQA1320E** **A trigraph was found when the FIND column specifications are not 1 to *.**

## Explanation

When searching through C or C++ source code, you can only specify FIND columns (explicitly or through SET FIND BOUNDS) other than 1 to * if no trigraphs exist in the source code.

**EQA1321E** **The SET FIND BOUNDS column specifications are invalid.**

## Explanation

The left column must be less than or equal to the right column.

**EQA1322E** **The FIND column specification(s) is invalid.**

## Explanation

The left column must be less than or equal to the right source margin. If left and right columns are specified, the left column must be less than or equal to the right column.

**EQA1323E** **The FIND argument will not fit between the specific columns.**

## Explanation

If left and right columns are specified or defaulted for FIND, the argument must fit within the specified columns. If only a left column is specified, the argument must fit within the left column and the right source margin.

**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***

## 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 z/OS Debugger 15.0.*m time stamp* 5724-T07: Copyright IBM Corp. 1992, 2020**

## Explanation

This message is used to place the z/OS Debugger logo, a timestamp, and copyright at the beginning of the log.

This message is also issued in response to the CALL %VER command. See "CALL %VER command" on page 83 for further details on additional information about the time stamp when CALL %VER is used.

**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** **PLAYBACK statement sequence error. PLAYBACK *found stmt* was found before PLAYBACK *req stmt*. Statement ignored.**

## Explanation

Playback was not is the proper state to process the specified statement. The statement is ignored.

**EQA1352I** **PLAYBACK statement sequence error. PLAYBACK *found stmt* was found after PLAYBACK *req stmt*. Statement ignored.**

## Explanation

Playback was not is the proper state to process the specified statement. The statement is ignored.

**EQA1353I** **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 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 z/OS Debugger 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. PLAYBACK START command rejected.**

## 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 FORWARD 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.

**EQA1366I** **PLAYBACK START is not allowed when AUTOMONITOR options BOTH or PREVIOUS are in effect.**

## Explanation

Playback cannot be started while AUTOMONITOR with option BOTH or AUTOMONITOR with option PREVIOUS are in effect.

**EQA1367I** **AUTOMONITOR options BOTH or PREVIOUS are not allowed while *PLAYBACK* is started.**

## Explanation

AUTOMONITOR with option PREVIOUS or AUTOMONITOR with option BOTH is not allowed while PLAYBACK is started.

**EQA1368I** **Previous location is now out of scope. No variables can be displayed.**

## Explanation

The variables from the previous statement cannot be resolved because the data is no longer available from the runtime at the time of the request.

**EQA1369I** **The previous statement is out of scope. No variables can be updated.**

## Explanation

The data from the statement in which z/OS Debugger last had control cannot be updated because the statement is no longer in scope.

**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 "*".

**EQA1376I** ***Channel_Tag Channel_Name***

## Explanation

This message contains the output from the DESCRIBE CHANNEL command.

**EQA1377I** **A CICS Storage Violation has not been detected.**

## Explanation

z/OS Debugger did not detect any storage violation.

**EQA1378W** **A CICS Storage Violation has been detected. The leading/trailing check zone associated with the storage that starts at *Hex_Str StgV_Address Quote_Str* for a length of *StgV_Length* has been damaged.**

## Explanation

z/OS Debugger detected a storage violation.

**EQA1379I** **No MEMORY data is available for the requested reference or address.**

## Explanation

The given reference or address is invalid.

**EQA1380I** **The following LABELS are known in *Program name*:**

## Explanation

This title line precedes a list of label names for a program in a LIST NAMES LABELS command.

**EQA1381I** **The following LABELS with pattern *pattern* are known in *PROGRAM name*:**

## Explanation

This title line precedes a list of label names for a program that satisfy the pattern in a LIST NAMES LABELS command.

**EQA1382I** **There are no LABELS with pattern *pattern* in *PROGRAM NAME*.**

## Explanation

This line appears when no LABELS satisfied the pattern in a LIST NAMES LABELS command for the currently qualified program.

**EQA1383I** **This command is not supported for currently qualified program *PROGRAM NAME*.**

## Explanation

This line appears when the language of the currently qualified program is PL/I, C, or C++.

**EQA1384I** **There are no LABELS in currently qualified program *PROGRAM NAME*.**

## Explanation

This line appears when no labels are found after issuing the LIST NAMES LABELS command for the currently qualified program.

**EQA1387I** **DTCN Pattern-match breakpoint disabled for:**

## Explanation

This is the title line for the LIST DTCN command.

**EQA1388I** **CADP Pattern-match breakpoint disabled for:**

## Explanation

This is the title line for the LIST CADP command.

**EQA1389I** **Load module = *load_module_Name* CU = *compile_unit_name***

## Explanation

This message lists the load module and compile unit names output by the LIST DTCN command.

**EQA1390I** **Program = *program_name* CU = *compile_unit_name***

## Explanation

This message lists the program and compile unit names output by the LIST CADP command.

**EQA1391I** **This program and/or compile unit is not in the pattern-match breakpoint list.**

## Explanation

The ENABLE CADP command is not allowed since this particular program, compile unit, or both are not in the pattern-match breakpoint list.

**EQA1392I** **The pattern-match breakpoint list is empty.**

## Explanation

There are no entries in the pattern-match breakpoint list.

**EQA1393I** **This load module and/or compile unit is not in the pattern-match breakpoint list.**

## Explanation

The ENABLE DTCN command is not allowed for this particular program and/or compile unit.

**EQA1394I** **This compile unit was optimized by ABO.**

## Explanation

After this COBOL program was compiled, the module was optimized by Automatic Binary Optimizer for z/OS.

For programs complied with Enterprise COBOL for z/OS Version 5 or later, you can debug them as the programs that are not optimized by ABO.

For programs complied with Enterprise COBOL for z/OS Version 4 or earlier, you can use the LangX COBOL support in z/OS Debugger to debug (with restrictions) a load module or program object generated by the ABO. For more information, see "Debugging a program processed by the Automatic Binary Optimizer for z/OS" in IBM z/OS Debugger User's Guide.

**EQA1396I** **Bottom of data reached.**

## Explanation

FINDBP has reached the bottom of the data without finding a breakpoint of the specified (or defaulted) status.

**EQA1397I** **Top of data reached.**

## Explanation

FINDBP has reached the top of the data without finding a breakpoint of the specified (or defaulted) status.

**EQA1398I** **An ENABLED breakpoint was found.**

## Explanation

FINDBP has found an ENABLED breakpoint. The cursor is placed in the prefix area of the line containing the breakpoint.

**EQA1399I** **An DISABLED breakpoint was found.**

## Explanation

FINDBP has found an DISABLED breakpoint. The cursor is placed in the prefix area of the line containing the breakpoint.

**EQA1400E** **The value entered is invalid.**

## Explanation

The user entered an invalid value.

**EQA1401E** **The command entered is not a valid panel sub-command.**

## Explanation

The user entered a command not recognized by panel processor.

**EQA1402E** **Each window must have unique letters of L, M, S, and E.**

## Explanation

Look at the Window Layout Select Panel, verify that each window has an L, M, S, or E and that each letter is used only once. For example, you cannot have two windows with the letter L.

**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, z/OS Debugger 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** **No line, statement or offset breakpoints were found.**

## Explanation

No line, statement or offset breakpoints were found of the specified (or defaulted) status.

**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.

**EQA1411E** **Invalid operand number was entered.**

## Explanation

The user entered a number corresponding to an invalid operand.

**EQA1412E** **Prefix command not supported for current programming language.**

## Explanation

Prefix command not supported for current programming language.

**EQA1413E** **There are no operands in the statement to display.**

## Explanation

The specified prefix command is on a statement with no operands.

**EQA1414E** **Prefix command was entered on invalid statement.**

## Explanation

The user entered prefix command on an invalid line.

**EQA1415E** **Specified operand number is too big.**

## Explanation

The user entered an operand number that exceeds the number of operands in that statement.

**EQA1416E** **Multiple invalid prefix commands. For details, reenter commands individually.**

## Explanation

More then one invalid prefix commands. To see details you must reenter the invalid command one at a time.

**EQA1417E** **Invalid line for prefix command. Line must be in the active block.**

## Explanation

The user entered a prefix command on a line that is contained in a block that is not currently active.

**EQA1418E** **One or more of multiple updates were not processed.**

## Explanation

One or more invalid updates. The commands were not processed. For details, please see messages in the LOG.

**EQA1419E** **Prefix command entered inside CC block command.**

## Explanation

The user entered a prefix command inside a CC block command.

**EQA1420E** **Block command incomplete.**

## Explanation

The user entered only one CC in the Monitor prefix area. A second CC is expected to finish the block command.

**EQA1421E** **The command is not supported in the List Pop-up window. Results are in the LOG.**

## Explanation

The user entered a LIST command that is not supported in the List Pop-up window. Results are in the LOG.

**EQA1422E** **Invalid SCROLL command for List pop-up window. Only SCROLL DOWN/UP are allowed.**

## Explanation

The user entered an invalid SCROLL command when the List pop-up window is shown. Only SCROLL DOWN/UP are allowed.

**EQA1423E** **List pop-up window shows first 1000 lines only. Complete results are in the LOG.**

## Explanation

The output for LIST command is bigger than 1000 lines. The List pop-up window shows only the first 1000 lines. Complete results are in the LOG.

**EQA1424W** **One or more variables undefined in current CU. MONITOR command or commands not defined.**

## Explanation

LOCAL was specified, limiting the monitor command to the current compile unit.

## Programmer response

Use the GLOBAL keyword to monitor variables in noncurrent compile units.

**EQA1425W** **One or more variables undefined in current CU. MONITOR command or commands not defined.**

## Explanation

The named variable could not be located or is undefined.

**EQA1426W** **An invalid qualifier in a qualified reference. MONITOR command or commands not defined.**

## Explanation

A qualified reference is invalid. One or more qualifiers might be undefined, or they are not in the same structure as the desired data item.

**EQA1427W** **Expression error. MONITOR not defined. Use LIST command to see the details in LOG.**

## Explanation

Error occurred during processing. Use the LIST command to see detailed message in the LOG.

**EQA1428W** **Multiple errors. MONITOR or MONITORS not defined. Use LIST commands to see details in LOG.**

## Explanation

Multiple expression errors occurred during processing. Use the LIST command to see detailed messages in the LOG.

**EQA1429E** **The label *label-name* has been removed due to optimization. Breakpoint not set.**

## Explanation

The label was removed by the compiler because of optimization.

**EQA1431W** **There are no EQUATE definitions in effect.**

## Explanation

`CLEAR EQUATE` or `QUERY EQUATES` was issued but there are no EQUATE definitions.

**EQA1432E** ***function* is not supported.**

## Explanation

Language/Country is not supported.

## Programmer response

Set National Language and Country.

**EQA1433E** **Switching to the programming language *language-name* is invalid because there are no *language-name* compilation units in the initial load module.**

## Explanation

A `SET PROGRAMMING LANGUAGE` command was issued, but the initial load module contains no compilation units compiled in the language specified (or implied).

**EQA1434E** **Error in setting debug *name* to *?????????*.**

## Explanation

Refer to the maximum number of CUs allowed for debugging.

**EQA1435E** **Error in setting *name*.**

## Explanation

This is a generic message for SET command errors.

**EQA1436W** **SET EXECUTE is OFF -- command will not be executed.**

## Explanation

The command was parsed but not executed.

**EQA1437W** **SET DYNDEBUG cannot 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.**

## Explanation

The Dynamic Debug facility setting cannot be changed to ON in the middle of a debugging session.

**EQA1438W** **SET DYNDEBUG cannot 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.**

## Explanation

The Dynamic Debug facility setting cannot be changed to OFF in the middle of a debugging session.

**EQA1439E** **This CU is not AUTOMONITOR capable for expressions.**

## Explanation

The CU is not AUTOMONITOR capable.

## Programmer response

Refer to the description of the SET AUTOMONITOR command in the *IBM z/OS Debugger Reference and Messages* document to determine the requirements this CU must fulfill in order to use the SET AUTOMONITOR command.

**EQA1440E** **SET AUTOMONITOR ON is not valid for this CU. Use the SET AUTOMONITOR ON LOG command to activate the statement trace function.**

## Explanation

The current CU is not AUTOMONITOR capable. SET AUTOMONITOR ON LOG will activate the statement trace.

## Programmer response

Refer to the description of the SET AUTOMONITOR command in the *IBM z/OS Debugger Reference and Messages* document to determine the requirements this CU must fulfill in order to use the SET AUTOMONITOR command.

**EQA1441I** **The statement trace is now active. Use the SET AUTOMONITOR OFF command to deactivate the statement trace.**

## Explanation

The statement trace is active for a CU that is not AUTOMONITOR capable.

**EQA1442E** **DYNDEBUG cannot be SET OFF when running without the Language Environment run-time.**

## Explanation

The Dynamic Debug facility cannot be deactivated while running without the Language Environment run-time.

**EQA1443I** **There are no INTERCEPT specifications in effect.**

## Explanation

QUERY INTERCEPT was issued but there are no INTERCEPT specifications active.

**EQA1444I** **For this command to be effective, SETTINGS must be saved.**

## Explanation

In order for SET RESTORE BPS AUTO or SET RESTORE MONITORS AUTO to be effective, the settings must be saved. You entered one of these commands but the current setting is SET SAVE SETTINGS NOAUTO.

**EQA1445E** ***this_cmd* conflicts with *State*, *default_cmd* is used.**

## Explanation

The specified command (*this_cmd*) conflicts with an existing setting (*State*). The altered command (*default_cmd*) is used.

**EQA1446E** **The label *label_name* has been removed due to optimization. No GOTO or JUMPTO are performed.**

## Explanation

The label was removed by the compiler because of optimization.

**EQA1447E** **The attempt to set this breakpoint failed. The statement might be invalid or was removed by the compiler due to optimization.**

## Explanation

The program is compiled with OPT. Compiler might have removed this state or the statement entered is not a valid statement. The breakpoint is not set.

**EQA1448W** **The MONITOR LIST command has already been established.**

## Explanation

The MONITOR LIST command that was entered is a duplicate of an existing MONITOR LIST command. The new command is ignored.

**EQA1449E** **The command is not supported with PL/I. Use PL/I built-in function HEX to obtain hexadecimal values.**

## Explanation

%HEX and MONITOR LIST %HEX are not valid for PL/I. It is recommended that the PL/I built-in function HEX be used instead. For example: LIST HEX(expr) or MONITOR LIST HEX(expr).

**EQA1450E** **Unable to display the result from expression evaluation**

## Explanation

The entire result from the expression evaluation cannot be displayed; for example, the array is too large.

**EQA1451E** ***operand*** **contains incompatible data type.**

## Explanation

Comparison or assignment involves incompatible data types, or incompatible or unsupported date fields. If you are using COBOL, see "Allowable comparisons for the IF command (COBOL)" on page 132 for allowable comparisons for the z/OS Debugger IF command, and "Allowable moves for the MOVE command (COBOL)" on page 177 for allowable moves for the z/OS Debugger MOVE command.

**EQA1452E** ***argument name*** **is not a valid argument.**

## Explanation

The specified argument is not valid.

**EQA1453E** **The number of arguments is not correct.**

## Explanation

There are either too many or too few arguments specified.

**EQA1454E** ***operand name*** **is not a valid operand.**

## Explanation

The specified operand is undefined or is an invalid literal.

**EQA1455E** **An unsupported operator/operand is specified.**

## Explanation

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

**EQA1456S** **The variable** ***variable name*** **is undefined or is incorrectly qualified.**

## Explanation

The named variable could not be located or undefined.

## Programmer response

You need to qualify to a different block in order to locate the variable.

**EQA1457E** **The exponent** ***exponent*** **contains a decimal point.**

## Explanation

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.**

## Explanation

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.**

## Explanation

The combination of characters specified for the literal is not a valid literal.

**EQA1460E** **Operand** ***operand name*** **should be numeric.**

## Explanation

A nonnumeric operand was found where a numeric operand was expected.

**EQA1461E** **Invalid data for** ***data item*** **is found.**

## Explanation

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.**

## Explanation

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.**

## Explanation

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.**

## Explanation

The target of an assignment is not valid.

**EQA1465E** **The TGT for a program is not available.**

## Explanation

The program might have been deleted or canceled.

**EQA1466E** ***data item*** **is not a valid subscript or index.**

## Explanation

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***

## Explanation

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***

## Explanation

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***

## Explanation

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***

## Explanation

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***

## Explanation

The ODO variable might not have been initialized, or the current value is out of range.

**EQA1472E** **Invalid specification of reference modification.**

## Explanation

The specification of the reference modification is not consonant with the length field.

**EQA1473E** **Invalid zero value for *data item***

## Explanation

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.**

## Explanation

Invalid name is specified for a data item.

**EQA1475E** ***data item*** **is an invalid qualifier in a qualified reference.**

## Explanation

A qualified reference is invalid. One or more qualifiers might be undefined or not in the same structure as the desired data item.

**EQA1476E** **Too many qualifiers in a qualified reference.**

## Explanation

The qualified reference contains more than the legal number of qualifiers.

**EQA1477E** **DATA DIVISION does not contain any entries.**

## Explanation

There is no data to display for a LIST * request because the DATA DIVISION does not contain any entries.

**EQA1478E** **No status available for sort file *sort file***

## Explanation

Status was requested for a sort file. There is never a status available for a sort file.

**EQA1479E** **Unable to locate any TGT. An attempt to locate any TGT failed.**

## Explanation

No COBOL program exists in TEST mode.

**EQA1480E** ***operand name* is an invalid operand for SET command.**

## Explanation

The operands for a SET command are incorrect. At least one of the operands must be index name.

**EQA1481E** **Too many digits for the exponent of floating point literal *data item***

## Explanation

The exponent specified for a floating-point literal contains too many digits.

**EQA1482E** ***command name* command is terminated due to an error in processing.**

## Explanation

The command is terminated unsuccessfully because an error occurred during processing.

**EQA1483E** ***reference* could not be formatted for display.**

## Explanation

The requested data item could not be displayed due to an error in locating or formatting the data item.

**EQA1484E** **Resources (for example, heap storage) are not available for processing and the command is terminated unsuccessfully.**

## Explanation

z/OS Debugger has an internal heap limit of 250M for processing expressions. The command could not be processed because this limit was exceeded.

## Programmer response

Reduce the number of structure or array elements being processed by explicitly listing only those of interest.

**EQA1485E** **The command is not supported because the CU is compiled with incorrect compiler options.**

## Explanation

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.

**EQA1486E** ***variable name* is presently not in accessible storage.**

## Explanation

The variable might be CONTROLLED or AUTOMATIC and does not yet exist.

**EQA1487S** **The number of dimensions for *variable name* is *number* -- but *number* have been specified.**

## Explanation

The wrong number of subscripts were specified with the variable reference.

**EQA1488E** **The indices in *variable name* are invalid. Use the DESCRIBE ATTRIBUTES command (without any indices specified) to see the valid indices.**

## Explanation

The subscripts with the variable reference do not properly relate to the variable's characteristics.

**EQA1489S** ***variable name* is not a based variable but a locator has been supplied for it.**

## Explanation

A pointer cannot be used unless the variable is BASED.

## Programmer response

Use additional qualification to get to the desired variable.

**EQA1490S** ***variable name*** **cannot be used as a locator variable.**

## Explanation

Only variables whose data type is POINTER or OFFSET can be used to locator with other variables.

**EQA1491S** **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.**

## Explanation

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.

**EQA1492S** **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.**

## Explanation

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.

**EQA1493E** **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.**

## Explanation

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 z/OS Debugger, pseudo-variables cannot be the target of assignments.

**EQA1494S** ***variable name*** **is an aggregate. It cannot be used as a locator reference.**

## Explanation

The variable that is being as a locator is not the correct data type.

**EQA1495S** **The name *variable name* is ambiguous and cannot be resolved.**

## 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.

## Programmer response

Retry the command with enough qualification so that the name is unambiguous.

**EQA1496S** **The name *variable name* refers to a structure, but structures are not supported within this context.**

## Explanation

Given `DCL 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.

**EQA1497S** **An aggregate cannot be used as an index into an array.**

## 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.

## Programmer response

Use a scalar as the index.

**EQA1498S** **Generation and recursion numbers must be positive.**

## Explanation

In %GENERATION(x,y) and %RECURSION(x,y), y must be positive.

**EQA1499S** **Generation and recursion expressions cannot be aggregate expressions.**

## Explanation

In %GENERATION(x,y) and %RECURSION(x,y), y must be a scalar.

**EQA1500S** **%RECURSION can be applied only to parameters and automatic variables.**

## Explanation

In %RECURSION(x,y), x must be a parameter or an automatic variable.

**EQA1501S** **%RECURSION *number* of *procedure name* does not exist. The present number of recursions of the block *block name* is *number*.**

## Explanation

In %RECURSION(x,y), y must be no greater than the number of recursions of the block where x is declared.

**EQA1502S** **%Generation can be applied only to controlled variables.**

## Explanation

In %GENERATION(x,y), x must be controlled.

**EQA1503S** **%Generation *number* of *variable name* does not exist. The present number of allocations of *variable name* is *number*.**

## Explanation

In %GENERATION(x,y), y must be no greater than the number of allocations of the variable x.

**EQA1504S** **%Generation *number* of %RECURSION (*procedure name*, *number*) does not exist. The present number of allocations of %RECURSION (*procedure name*, *number*) is *number*.**

## Explanation

In %GENERATION(x,y), y must be no greater than the number of allocations of the variable x.

**EQA1505S** **The variable *variable name* belongs to a FETCHed procedure and is a CONTROLLED variable that is not a parameter. This violates the rules of PL/I.**

## Explanation

PL/I does not allow FETCHed procedures to contain CONTROLLED variable types.

## Programmer response

Correct the program.

**EQA1506S** **The variable *character string* cannot be used.**

## Explanation

The variable belongs to the class of variables, such as members of structures with REFER statements, which z/OS Debugger does not support.

**EQA1507E** **The expression in the QUIT command must be a scalar that can be converted to an integer value.**

## 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.

**EQA1508E** **An internal error occurred in C run time during expression processing.**

## Explanation

This message applies to C. An internal error occurred in the C run time and the command is terminated.

**EQA1509E** **The unary operator *operator name* requires a scalar operand.**

## Explanation

This message applies to the C unary operator ! (logical negation).

**EQA1510E** **The unary operator *operator name* requires a modifiable lvalue for its operand.**

## Explanation

This message applies to the C unary operators ++ and --.

**EQA1511E** **The unary operator *operator name* requires an integer operand.**

## Explanation

This message applies to the C unary operator ~ (bitwise complement).

**EQA1512E** **The unary operator *operator* requires an operand that is either arithmetic or a pointer to a type with defined size.**

## 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.

**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 +.

**EQA1536E** **The operand of the address operator must be a function designator or an lvalue that is not a bitfield.**

## Explanation

This message applies to the C unary operator & (address).

**EQA1537E** **Invalid constant for the C language.**

## Explanation

This message applies to C constants.

**EQA1538E** **Argument *argument* in function call *function* is incompatible with the function definition. Since Warning is on, the function call is not made.**

## Explanation

This message applies to C function calls. The argument must have a type that would be valid in an assignment to the parameter.

**EQA1539E** **The binary operator *operator* requires integer operands.**

## Explanation

This message applies to the C binary operator % (remainder), << (bitwise left shift), >> (bitwise right shift), & (bitwise and), ??¬' (bitwise exclusive or), |(bitwise inclusive or), and the corresponding assignment operators (for example, %=, and <<=).

**EQA1540E** **The binary operator *operator* requires a modifiable lvalue for its first operand.**

## Explanation

This message applies to the C binary assignment operators.

**EQA1541E** **The binary operator *operator* requires arithmetic operands.**

## Explanation

This message applies to the C binary operators * and /.

**EQA1542E** **Source in assignment to an enum is not a member of the enum. Since Warning is on, the operation is not performed.**

## Explanation

This message applies to C. You attempted to assign a value to enum, but the value is not legitimate for that enum.

**EQA1543E** **Invalid value for the shift operator *operator*. Since Warning is on, the operation will not be performed.**

## Explanation

This message applies to the C binary operators << (bitwise left shift) and >> (bitwise right shift). Shift values must be nonnegative and less than 33. These tests are made only when WARNING is on.

**EQA1544E** **Array subscript is negative. Since Warning is on, the operation is not performed.**

## Explanation

This message applies to the C subscripts.

**EQA1545E** **Array subscript exceeds maximum declared value. Since Warning is on, the operation is not performed.**

## Explanation

This message applies to the C subscripts.

**EQA1546E** **ZeroDivide would have occurred in performing a division operator. Since Warning is on, the operation is not performed.**

## Explanation

Divide by zero is detected by C run time.

**EQA1547E** ***variable* is presently not in accessible storage.**

## Explanation

This message applies to C. Use the `LIST NAMES` command to list all known variables.

**EQA1548E** **There is no variable named *variable***

## Explanation

This message applies to C. Use the `LIST NAMES` command to list all known variables.

**EQA1549E** **The function call *function* is not performed because the function linkages do not match.**

## Explanation

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.

**EQA1550E** **There is no typedef identifier named *name***

## Explanation

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.

**EQA1551E** ***name* is the name of a member of an enum type.**

## Explanation

This message applies to C.

**EQA1552E** **The name *name* is invalid.**

## Explanation

This message applies to C declarations.

**EQA1553E** **Linkage type for function call *function* is unknown.**

## Explanation

This message applies to C function calls.

**EQA1554E** **Function call *function* has linkage type PL/I, which is not supported.**

## Explanation

This message applies to C function calls.

**EQA1555E** **Function call *function* has linkage type FORTRAN which is not supported.**

## Explanation

This message applies to C function calls.

**EQA1556E** ***name* is a tag name. This cannot be listed since it has no storage associated with it.**

## Explanation

This message applies to C tag names.

**EQA1557E** ***name* is not an lvalue. This cannot be listed since it has no storage associated with it.**

## 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 `CALL %DUMP` for 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** **Reference is too large. Use LIST to display**

## Explanation

The requested data item cannot be displayed with automonitor because it is too large. Use MONITOR LIST to display the item in the monitor. This could cause a short of storage condition.

**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 "=" and "¬=" 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/I 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/I, 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/I constants.

**EQA1597E** **AREA condition would have been raised**

## Explanation

This message applies to PL/I 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** **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

z/OS Debugger does not support aggregates in this context.

**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.

**EQA1617W** **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.

**EQA1618W** **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.

**EQA1619W** **The third 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.

**EQA1620E** **If the CHAR built-in function is invoked with only one argument, that argument must not have the GRAPHIC attribute with length 16383.**

## Explanation

CHAR(x) is illegal if x is GRAPHIC with length 16383 since the resultant string would require 32768 characters.

**EQA1621E** ***built-in function* ( *variable name* ) is not defined since *variable name* is not connected.**

## Explanation

This applies to the PL/I CURRENTSTORAGE and STORAGE built-in functions.

**EQA1622E** ***built-in function* ( *variable name* ) is not defined since *variable name* is an unaligned fixed-length bit string.**

## Explanation

This applies to the PL/I CURRENTSTORAGE and STORAGE built-in functions.

**EQA1623E** ***built-in function* (x) is undefined if ABS(x) > 1.**

## Explanation

This applies to the PL/I ASIN and ACOS built-in functions.

**EQA1624E** **ATANH(z) is undefined if z is COMPLEX and z = +1 or z = -1.**

## Explanation

This applies to the PL/I ATANH built-in function.

**EQA1625E** **ATAN(z) is undefined if z is COMPLEX and z = +1i or z = -1i.**

## Explanation

This applies to the PL/I ATAN built-in function.

**EQA1626E** **Built-in function not defined since the argument is real and less than or equal to zero**

## Explanation

This message applies to PL/I constants.

**EQA1627E** **SQRT(x) is undefined if x is REAL and x < 0.**

## Explanation

This applies to the PL/I SQRT built-in function.

**EQA1628E** ***built-in function* (x,y) is undefined if x or y is COMPLEX.**

## Explanation

This applies to the PL/I ATAN and ATAND built-in functions.

**EQA1629E** **Built-in function(X,Y) 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

**TAN**
ABS(X) <= (2**18)*pi if x is real and ABS(REAL(X)) <= (2**17)*pi if x is complex

**TANH**
ABS(IMAG(X)) <= (2**17)*pi if x is complex

**COSH, EXP and SINH**
ABS(IMAG(X)) <= (2**18)*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

**TAN**
ABS(X) <= (2**50)*pi if x is real and ABS(REAL(X)) <= (2**49)*pi if x is complex

**TANH**
ABS(IMAG(X)) <= (2**49)*pi if x is complex

**COSH, EXP and SINH**
ABS(IMAG(X)) <= (2**50)*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

**TAN**
ABS(X) <= (2**106)*pi if x is real and ABS(REAL(X)) <= (2**105)*pi if x is complex

**TANH**
ABS(IMAG(X)) <= (2**105)*pi if x is complex

**COSH, EXP and SINH**
ABS(IMAG(X)) <= (2**106)*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 `ALLOCATE`able.

## 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 `DCL 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. z/OS Debugger 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, ERF, 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 z/OS Debugger 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 F. Syntax of the TEST Compiler option in the *IBM z/OS Debugger 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 z/OS Debugger tried to set this breakpoint, an error occurred. This breakpoint cannot be set.

**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 z/OS Debugger 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 a nd 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, TRAN SMIT and UNDEFINEDFILE. This would also apply to the corresponding TRIGGER commands.

**EQA1673W** **An attempt to automatically restore an AT *breakpoint type* breakpoint failed.**

## Explanation

z/OS Debugger 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 z/OS Debugger attempts to reestablish that breakpoint, it will fail with this message.

**EQA1674W** **An attempt to automatically disable an AT *breakpoint type* breakpoint failed.**

## Explanation

z/OS Debugger 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

z/OS Debugger 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.

**EQA1684E** **Since the program for the statement *statement-number* has DYNAMIC DEBUG turned off, AT commands are rejected for all statements in the program.**

## Explanation

A compile unit must have been compiled with TEST(STMT) or TEST(ALL) or set DYNDEBUG ON for at statements.

**EQA1685E** **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.

**EQA1686E** **The command/option *Keyword* is not supported for Disassembly View.**

## Explanation

The command or option is not supported for a DISASSEMBLY compile unit. See the IBM z/OS Debugger Reference and Messages document for information about the restrictions on the use of this command.

**EQA1687E** **The command LIST*Keyword* is not supported in the Compile Unit *Cu_name*.**

## Explanation

The command is not supported for a DISASSEMBLY compile unit. See the IBM z/OS Debugger Reference and Messages document for information about the restrictions on the use of this command.

**EQA1688E** **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.

**EQA1689E** **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.

**EQA1690I** **The current programming language does not return information for DESCRIBE ENVIRONMENT.**

## Explanation

The current programming language does not support the DESCRIBE ENVIRONMENT command.

**EQA1691I** **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 Enterprise PL/I.

**EQA1692W** **Restoring of assembler breakpoints is not currently supported.**

## Explanation

Breakpoints in assembler compile units are not restored.

**EQA1693E** **The command AT *Keyword* is not supported in the Compile Unit *Cu_name* because it is PLI compiled with the NOTEST option and does not have the hook necessary to set the breakpoint.**

## Explanation

The command is not supported for a Compile Unit compiled with a High Level Language compiler with the NOTEST option since the compile unit does not have the hook necessary to set the breakpoint.

**EQA1694I** **The *command_name* command is not supported in the current program.**

## Explanation

The command, *command_name*, is not supported in programs that are compiled with Enterprise PL/I.

**EQA1695E** **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.

**EQA1696E** **Conditional Expression *Conditional_Logic_Expression* in WHEN clause cannot 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.

**EQA1697E** **Conditional Expression *Conditional_Logic_Expression* in WHEN clause cannot 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.

**EQA1698E** **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.

**EQA1699E** ***Address_Length_Info Flags Name***

## Explanation

This message contains the output from the DESCRIBE LOADMODS command.

**EQA1700E** **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.

**EQA1708I** **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 cannot 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 z/OS Debugger 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

```
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.

**EQA1739I** **Some or all of the** ***save_restore_cmd*** **could not be restored from** ***dsname***

## Explanation

An attempt was made to restore breakpoints and/or monitor settings from the specified data set. However, the specified data set contained invalid data so some or all of the breakpoints and/or monitor settings could not be restored from the specified data set.

**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 z/OS Debugger 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, an invalid RECFM, 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** **An error occurred in "switch" command processing. The command is terminated.**

## Explanation

This message applies to C. The `switch` command is terminated because an error occurred during processing.

**EQA1755S** **Comparison with the** ***keyword-name*** **keyword in** ***command-name*** **command was invalid. The command was ignored.**

## Explanation

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
```

**EQA1756W** **EQALANGX file not available for CSECT *CMD_NAME*.**

## Explanation

A debug data file is not found for the supplied CSECT name.

**EQA1757W** **Cannot *save_restore* breakpoints and/or monitors because an OPEN exit is active.**

## Explanation

Breakpoints and monitors cannot be saved or restored when an OPEN exit is active because MVS does not support using dynamic allocation in this situation.

**EQA1758E** **Operation is not permitted in Browse Mode.**

## Explanation

When Browse Mode is active, operations that modify storage or registers are not permitted. In addition, other operations such as clearing of the log are also not permitted.

**EQA1759E** **Operations that alter the control flow are not permitted in Browse Mode.**

## Explanation

When Browse Mode is active, operations that modify the flow of control are not allowed.

**EQA1760I** **Use QUIT DEBUG or QUIT ABEND to exit z/OS Debugger in Browse Mode.**

## Explanation

When Browse Mode is active, you cannot use QQUIT, QUIT; or QUIT (*expression*) because this alters the control flow. You must use either QUIT DEBUG or QUIT ABEND.

**EQA1763E** ***save_restore_cmd* failed because *dsname* is not a partitioned data set.**

## Explanation

The specified data set must be a partitioned data set.

**EQA1764E** ***save_restore_cmd* could not locate data set *dsname*.**

## Explanation

The specified data set could not be located.

**EQA1765E** ***save_restore_cmd* error *rc-reason* allocating *dsname*.**

## Explanation

The specified data set could not be allocated. The return code and reason code are shown as *ddd-xxxxyyyy*. In most cases, *xxxx* is the S99Error code from dynamic allocation. You can use this code to determine more information about the source of the error. For more information about the S99Error codes, see *z/OS MVS Programming: Authorized Assembler Services Guide*. You should also inspect the MVS console log for other messages associated with this error.

**EQA1766E** **The target of the GOTO command is in an inactive block.**

## Explanation

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 allow the procedure to run).

**EQA1767S** **No offset was found for label "*label*".**

## Explanation

No offset associated with the label was found; the code associated with the label might have been removed by optimization.

**EQA1768S** **The label "*label*" is not known.**

## Explanation

The label is not known.

**EQA1769S** **The label "*label*" is ambiguous - multiple labels of this name exist.**

## Explanation

The label is ambiguous; multiple labels of this name exist.

**EQA1770S** **The GOTO is not permitted, perhaps because of optimization and SET WARNING is ON.**

## Explanation

The GOTO command is not recommended. For COBOL, this might be caused by optimization, or because register contents other than the code base cannot be guaranteed for the target. If SET WARNING is OFF, z/OS Debugger executes the command, but the results are unpredictable.

**EQA1771S** **The GOTO is not permitted due to language rules.**

## Explanation

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.

**EQA1772S** **The GOTO was not successful.**

## Explanation

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.

**EQA1773E** **GOTO is invalid when the target statement number is in a C function.**

## Explanation

The target statement number in a GOTO command must belong to an active procedure.

**EQA1776E** **The target of the JUMPTO command is in an inactive block.**

## Explanation

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.

**EQA1777E** ***variable_name*** **is not a LABEL variable or constant. No JUMPTO commands can be issued against it.**

## Explanation

You are trying to JUMPTO a variable name that cannot be associated with a label in the program.

**EQA1778S** **The JUMPTO is not allowed, perhaps because of optimization and SET WARNING is ON.**

## Explanation

The JUMPTO command is not recommended. For COBOL, this might be caused by optimization, or because register contents other than the code base cannot be guaranteed for the target. If SET WARNING is OFF, z/OS Debugger executes the command, but the results are unpredictable.

**EQA1779S** **The JUMPTO is not permitted due to language rules.**

## Explanation

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.

**EQA1780S** **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.

**EQA1781E** **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.

**EQA1782I** **EQALANGX data from *LANGX_File_ID* will be used for deferred LDD *cu_spec*.**

## Explanation

When the deferred LDD for the specified CU is executed, the EQALANGX data will be loaded from the specified data set.

**EQA1786W** **There are no entries in the HISTORY table.**

## Explanation

z/OS Debugger has not yet encountered any of the situations that cause entries to be put into the HISTORY table; so it is empty.

**EQA1787W** **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. z/OS Debugger was not invoked for any STATEMENT hooks.

**EQA1788W** **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. z/OS Debugger was not invoked for any PATH hooks.

**EQA1789W** **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).

**EQA1790W** **There are no active blocks.**

## Explanation

The LIST CALLS command was issued before any STEP or GO.

**EQA1791E** **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.)

**EQA1792S** **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.

**EQA1793S** **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.

**EQA1794S** **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.

**EQA1795W** **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.

**EQA1806E** **The command element *character* is invalid.**

## Explanation

The command entered could not be parsed because the specified element is invalid.

**EQA1807E** **The command element *character* is ambiguous.**

## Explanation

The command entered could not be parsed because the specified element is ambiguous.

**EQA1808E** **The hyphen cannot appear as the last character in an identifier.**

## Explanation

COBOL identifiers cannot end in a hyphen.

**EQA1809E** **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.

**EQA1810E 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.

**EQA1811E 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 A decimal exponent is required.**

## Explanation

In COBOL, an E in a float constant must be followed by at least one decimal digit (optionally preceded by a sign). In C, if a + or - sign is specified after an E in a float constant, it must followed by at least one decimal digit.

**EQA1813E Error reading DBCS character codes.**

## Explanation

An unmatched or nested shift code was found.

**EQA1814E Identifier is too long.**

## Explanation

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.

**EQA1815E Invalid character code within DBCS name, literal or DBCS portion of mixed literal.**

## Explanation

A character code point was encountered that was not within the defined code values for the first or second byte of a DBCS character.

**EQA1816E An error was found at line *line-number* in the current input file.**

## Explanation

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.

**EQA1817E Invalid hexadecimal integer constant specified.**

## Explanation

A hexadecimal digit must follow 0x.

**EQA1818E Invalid octal integer constant specified.**

## Explanation

Only an octal digit can follow a digit-0.

**EQA1819E A COBOL DBCS name must contain at least one nonalphanumeric double byte character.**

## Explanation

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'.

**EQA1820E Invalid double byte alphanumeric character found in a COBOL DBCS**

**name. Valid COBOL double byte alphanumeric characters are: A-Z, a-z, 0-9.**

## Explanation

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.

**EQA1821E** **The DBCS representation of the hyphen was the first or last character in a DBCS name.**

## Explanation

COBOL DBCS names cannot have a leading or trailing DBCS hyphen.

**EQA1822E** **A DBCS Name, DBCS literal or mixed SBCS/DBCS literal may not be continued.**

## Explanation

Continuation rules do not apply to DBCS names, DBCS literals or mixed SBCS/DBCS literals. These items must appear on a single line.

**EQA1823E** **An end of line was encountered before the end of a DBCS name or DBCS literal.**

## Explanation

An end of line was encountered before finding a closing shift-in control code. This message is for the System/370 environment.

**EQA1824E** **A DBCS literal or DBCS name contains no DBCS characters.**

## Explanation

A shift-out shift-in pair of control characters were found with no intervening DBCS characters. This message is for the System/370 environment.

**EQA1825E** **End-of-source was encountered while processing a DBCS name or DBCS literal.**

## Explanation

No closing Shift-In control code was found before end of file. This message is for the System/370 environment.

**EQA1826E** **A DBCS literal was not delimited by a trailing quote or apostrophe.**

## Explanation

No closing quotation mark

**EQA1827E** **Invalid separator character found following a DBCS name.**

**EQA1828E** **Fixed binary constants are limited to 31 digits.**

## Explanation

A fixed binary constant must be between -2**31 and 2**31 exclusive.

**EQA1829E** **Fixed decimal constants are limited to 20 digits.**

## Explanation

A fixed decimal constant must be between -10**20 and 10**20 exclusive.

**EQA1830E** **Float binary constants are limited to 109 digits.**

## Explanation

This limit applies to all PL/I FLOAT BINARY constants.

**EQA1831E** **Float decimal constants are limited to 33 digits.**

## Explanation

This limit applies to all PL/I FLOAT DECIMAL constants.

**EQA1832E** **Floating-point exponents are limited to 3 digits.**

## Explanation

This limit applies to all C float constants and to all PL/I FLOAT BINARY constants.

**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.

**EQA1836E** **Float constants are limited to 35 digits.**

## Explanation

This limit applies to all C `float` constants.

**EQA1837E** **Float constants must be bigger than 5.39760534693402789086646991425024 96E -79 and less than 7.23700557733226221397318656304299 29E +75.**

## Explanation

This is the range of values allowed by C.

**EQA1838E** **The DBCS representation of the underscore was the first character in a DBCS name.**

## Explanation

COBOL DBCS names cannot have a leading DBCS underscore.

**EQA1856S** **The module "*module*" cannot be loaded - it was NOT found.**

## Explanation

The `LOAD MODULE` request failed, the module was not found.

**EQA1857S** **The module "*module*" cannot be loaded - there is NOT enough storage to do the load.**

## Explanation

The `LOAD MODULE` request failed, there is not enough storage.

**EQA1858S** **The module "*module*" cannot be loaded.**

## Explanation

The LOAD MODULE request was unsuccessful.

**EQA1860S** **The module "*module*" was NOT loaded by the z/OS Debugger and therefore CANNOT be deleted.**

## Explanation

Only modules loaded by the z/OS Debugger may be deleted.

**EQA1861S** **The module "*module*" cannot be loaded because it was already loaded by z/OS Debugger.**

## Explanation

The `LOAD MODULE` request was unsuccessful.

**EQA1862E** **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.

**EQA1863I** **%FA complete. See your Fault Analyzer history file.**

## Explanation

%FA was complete.

## Programmer response

%FA was complete. See your Fault Analyzer history file.

**EQA1864S** **LOAD is not supported during initialization. STEP or GO, then retry the LOAD command.**

## Explanation

You cannot issue a LOAD request until z/OS Debugger initialization is complete.

**EQA1865I** **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.

**EQA1866I** **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.

**EQA1867W** **A load module contains LE CUs but LE is not active. The LE CUs will not be created.**

## Explanation

A LOAD command was used in a non-Language Environment environment to load a load module that contained one or more Language Environment compile units or an attempt was made to QUALIFY to an Language Environment compile unit in a previously loaded load module. These compile units cannot be created until Language Environment is active. Compile units will be created only for non-Language Environment compile units.

**EQA1868I** **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.

**EQA1869I** **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.

**EQA1870S** **The CU *CU_name* cannot be destroyed.**

## Explanation

The attempt to destroy the specified CU was unsuccessful.

**EQA1872E** **An error occurred while opening file: *file name*. The file may not exist, or is not accessible.**

## Explanation

An error during the initial processing (OPEN) of the file occurred.

**EQA1873E** **An error occurred during an input or output operation.**

## Explanation

An error occurred performing an input or output operation.

**EQA1874I** **The command *command name* can be used only in full screen mode.**

## Explanation

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.

**EQA1875I** **Insufficient storage available.**

## Explanation

This message is issued when not enough storage is available to process the last command issued or to handle the last invocation.

**EQA1876E** **Not enough storage to display results.**

## Explanation

Increase size of virtual storage.

**EQA1877E** **An error occurred in writing messages to the dump file.**

## Explanation

This could be caused by a bad file name specified with the call dump FNAME option.

**EQA1878E** **The cursor is not positioned at a variable name.**

## Explanation

A command, such as `LIST`, `LIST TITLED`, `LIST STORAGE`, or `DESCRIBE ATTRIBUTES`, which takes input from the Source window was entered with the cursor in the Source window, but the cursor was not positioned at a variable name.

## Programmer response

Reposition the cursor and reenter.

**EQA1879E** **The listing file name given is too long.**

## Explanation

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).

**EQA1880E** **You may not resume execution when the program is waiting for input.**

## Explanation

The user attempted to issue a GO/RUN or STEP request when the program was waiting for input. The input must be entered to resume execution.

**EQA1881E** **The INPUT command is only valid when the program is waiting for input.**

## Explanation

The user attempted to enter the INPUT command when the program was not waiting for any input.

**EQA1882E** **The logical record length for *filename* is out of bounds. It will be set to the default.**

## Explanation

The logical record length is less than 32 bytes or greater than 256 bytes.

**EQA1883E** **Error closing previous log file; Return code = *rc***

## Explanation

The user attempted to open a new log file and the old one could not be closed; the new log file is used, however.

**EQA1884E** **An error occurred when processing the source listing. Check return code *return code* in the Using the z/OS Debugger manual for more detail.**

## Explanation

An error occurred during processing of the list lines command. Possible return codes:

2 - The listing file could not be found or allocated.

5 - The CU was not compiled with the correct compile option.

7 - Failed due to inadequate resources.

**EQA1885I** **Attempt to open INSPPREF failed. User did not specify the Preferences File TEST option and/or did not allocate INSPPREF.**

## Explanation

If the Preferences File TEST option is not specified, the default is INSPPREF. The user did not specify the Preferences File TEST option, so z/OS Debugger assumes INSPPREF and tries to open it. If INSPPREF is not allocated, this open fails.

**EQA1886I** ***** Global preferences file commands follow *****

## Explanation

Start of commands in the global preferences file.

**EQA1887I** ***** User preferences file commands follow *****

## Explanation

Start of commands in the user preferences file.

**EQA1888I** ***** Commands file commands follow *****

## Explanation

Start of commands in the commands file.

**EQA1889I** ***** Global preferences file commands end *****

## Explanation

End of commands in the global preferences file.

**EQA1890I** ***** User preferences file commands end *****

## Explanation

End of commands in the user preferences file.

**EQA1891I** ***** Commands file commands end *****

## Explanation

End of commands in the commands file.

**EQA1892I** **Global Preferences file exists: *file name***

## Explanation

The global preferences file is opened successfully.

**EQA1893I** **Default User Preferences file exists: *file name*.**

## Explanation

The default user preferences file is opened successfully.

**EQA1894I** **Default User Commands file exists: *file name*.**

## Explanation

The default user commands file is opened successfully.

**EQA1902W** **The command has been terminated because of the attention request.**

## Explanation

The previously-executing command was terminated because of an attention request. Normal debugging can continue.

**EQA1903E** **An attention request has been issued. Enter QUIT to terminate z/OS Debugger or GO or RUN to resume execution.**

## Explanation

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.

**EQA1904E** **The STEP and GO/RUN commands are not allowed at termination.**

## Explanation

The STEP and GO/RUN commands are not allowed after the application program ends.

**EQA1905W** **You cannot trigger a condition in your program at this time.**

## Explanation

The environment is in a position that it would not be meaningful to trigger a condition. For example, you have control during environment initialization.

**EQA1906S** **The condition named *CONDITION name* is unknown.**

## Explanation

A condition name was expected, but the name entered is not the name of a known condition.

**EQA1907W** **The attempt to trigger this condition has failed.**

## Explanation

For some reason, when z/OS Debugger tried to trigger the specified condition, it failed and the condition was not signaled.

**EQA1918S** **The block name *block-qualification* :> *block_name* is ambiguous.**

## Explanation

There is another block that has the same name as this block.

## Programmer response

Provide further block name qualification: by load module name, by compile unit name, or by additional block names if a nested block.

**EQA1919E** **The present block is not nested. You cannot QUALIFY UP.**

## Explanation

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.

## Programmer response

You have either misinterpreted your current execution environment or you have to qualify to some block explicitly.

**EQA1920E** **The present block has no dynamic parent. You cannot QUALIFY RETURN.**

## Explanation

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.

### Programmer response

You have either misinterpreted your current run-time environment or you have to qualify to some block explicitly.

**EQA1921S** **There is no block named *block_name*.**

### Explanation

The block that you named could not be located by z/OS Debugger.

### Programmer response

Provide further block name qualification: by load module name, by compile unit name, or by additional block name(s) if a nested block.

**EQA1922S** **There is no block named *block_name* within block *block-qualification*.**

### Explanation

The qualification you are using (or the spelling of the block names) prevented z/OS Debugger from locating the target block.

### Programmer response

Verify that the named block should be within the current qualification.

**EQA1923S** **There is no compilation unit named *cu_name*.**

### Explanation

The compilation unit (program) that you named could not be located by z/OS Debugger.

**EQA1924S** **Statement *statement_id* is not valid.**

### Explanation

The statement number does not exist or cannot be used. Note that the statement number could exist but is unknown.

**EQA1925S** **There is no load module named *load module name*.**

### Explanation

Load module qualification is referring to a load module that cannot be located.

### Programmer response

The load module might be missing or it might have been loaded before z/OS Debugger was first used. On the System/370, z/OS Debugger is aware of additional load modules **only** if they were FETCHed after z/OS Debugger got control for the first time.

**EQA1926S** **There is no cu named *cu_name* within load module *load module name*.**

### Explanation

The compilation unit might be misspelled or missing.

**EQA1927S** **There are *number* CUs named *cu_name*, but neither belongs to the current load module.**

### Explanation

The compilation unit you named is not unique.

### Programmer response

Add further qualification so that the correct load module will be known.

**EQA1928S** **The block name *block_name* is ambiguous.**

### Explanation

There is another block that has the same name as this block.

### Programmer response

Provide further block name qualification: by load module name, by compile unit name, or by additional block names if a nested block.

**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 z/OS Debugger.

**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 z/OS Debugger.

## Programmer response

Provide further block name qualification--by load module name, by compile unit name, or by additional block name(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 z/OS Debugger commands are unpredictable.**

## Explanation

z/OS Debugger 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 z/OS Debugger. 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 z/OS Debugger to locate the block named.

**EQA1939W** ***CU-name*** **is an assembler CU name that is longer than 8 characters.**

## Explanation

Assembler CU names longer than 8 characters are not currently supported.

**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.

**EQA1946I** **The variable is too big to be displayed. Resources (for example, heap storage) are not available.**

## Explanation

The command could not be completed due to inadequate resources. The variable is too big.

**EQA1947E** **The specified address does not correspond to any known CU.**

## Explanation

The specified address is not within any known CU.

**EQA1948E** **The .dbg file for *CompileUnitName* created by the FORMAT(DWARF) suboption of the DEBUG compiler option cannot be found. Make sure the file is in the location specified during compile time or use the EQADBG DD statement, EQAUEDAT user exit, or set EQA_DBG_PATH environment variable before starting the debug session to indicate the alternate location of the file. If your debug session is already started then use the SET DEFAULT DBG command.**

## Explanation

The .dbg file containing the debugging tables cannot be found. Some of the possible conditions that could cause this are:

- The .dbg file was deleted.
- The .dbg file was moved to a new location.
- The user does not have RACF authorization to access the file.

## Programmer response

Make sure the file is in the location specified during compile time or use the EQADBG DD statement, EQAUEDAT user exit, or set EQA_DBG_PATH environment variable before starting the debug session to indicate the alternate location. If your debug session is already started, then use the SET DEFAULT DBG command.

**EQA1949E** **The EQALANGX file does not match the object for *Compile Unit name*. The EQALANGX file cannot be used.**

## 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.

**EQA1950E** **The MONITOR table is empty. If the first MONITOR command entered is numbered, it must have number 1.**

## Explanation

A `MONITOR n` command was issued when the MONITOR table is empty, but n is greater than 1.

**EQA1951E** **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*.**

## Explanation

A `MONITOR n` command was issued but n is greater than 1 plus the highest numbered `MONITOR` command.

**EQA1952E** **The MONITOR command table is full. No unnumbered MONITOR commands will be accepted.**

## Explanation

A `MONITOR` command was issued but the `MONITOR` table is full.

## Programmer response

The Monitor Window is used to display the value of a variable and supports up to 99 instances. Variables are added to the Monitor Window using the `MONITOR` or `AUTOMONITOR` primary commands or via the `M` line command in the Source window. Each time one of these commands is issued, the variable is added to the Monitor window followed by a distinct number (1, 2, ... 99). You can reduce the number of variables appearing in the Monitor window by entering one of the following commands:

- `CLEAR MONITOR n` (clears variable numbered n)
- `CLEAR MONITOR (n1, n2, ...)` (clears variables numbered n1, n2, etc.)
- `CLEAR MONITOR` (clears all variables)

You can selectively replace an existing variable in the Monitor window by using the distinct number of that variable in the command `MONITOR n LIST new-variable`. For example, `MONITOR 10 LIST VAR9` replaces the 10th variable in the Monitor window with the variable VAR9.

**EQA1953E** **No command has been set for MONITOR *monitor-number*.**

## Explanation

A `LIST MONITOR n` or `CLEAR MONITOR n` command was issued, but n is greater than the highest numbered `MONITOR` command.

**EQA1954E** **The command for MONITOR *monitor-number* has already been cleared.**

## Explanation

A `CLEAR MONITOR n` command was issued, but `MONITOR` has already been cleared.

**EQA1955E** **There are no MONITOR commands established.**

## Explanation

A `LIST MONITOR` or `CLEAR MONITOR` command was issued, but there are no `MONITOR` commands established.

**EQA1956E** **No previous FIND argument exists. FIND operation not performed.**

## Explanation

A `FIND` command must include a string to find when no previous `FIND` command has been issued.

**EQA1957E** **String could not be found.**

## Explanation

A `FIND` attempt failed to find the requested string.

**EQA1958E** **The requested SYSTEM command could not be run.**

## Explanation

A `SYSTEM` command was issued. The underlying operating system received it but did not process it successfully.

**EQA1959E** **The requested SYSTEM command was not recognized.**

## Explanation

The underlying operating system was passed a command that was not recognized. The system could not process the command.

**EQA1960S** **There is an error in the definition of variable *variable name*. Attribute information cannot be displayed.**

## Explanation

The specified variable has an error in its definition or length and address information is not currently available in the execution of the program.

**EQA1961E** **Automonitor cannot be removed or replaced. Use SET AUTOMONITOR OFF command.**

## Explanation

The Automonitor can only be set off with the SET AUTOMONITOR OFF command.

**EQA1962E** **Automonitor is already set off.**

## Explanation

The Automonitor function is already off.

**EQA1963S** **The *command* command is not supported on this platform.**

## Explanation

The given command is not supported on the current platform.

**EQA1964E** **Source or Listing data is not available, or the CU was not compiled with the correct compile options.**

## Explanation

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)

**EQA1965E** **Attributes of source of assignment statement conflict with target *variable name*. The assignment cannot be performed.**

## Explanation

The assignment contains incompatible data types; the assignment cannot be made.

**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 z/OS Debugger. Uppercase United States English (UEN) will be used instead.**

## Explanation

The national-language-specified conflicts with the supported national languages for this installation of z/OS Debugger. 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 quotation marks.

**EQA1975E** **SYSDEBUG/SEPARATE file cannot 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 cannot 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 cannot 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 being 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 cannot 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. Module= *module_name*, Location= *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.

**EQA1988I** **The environment variable QPPLISTFILES is not defined.**

## Explanation

For Q++ programs, z/OS Debugger requires that you specify the path where the list files are stored in the environment variable QPPLISTFILES.

## Programmer response

You can use the Language Environment runtime option ENVAR to specify the path where the list files are stored. For example, the following runtime option specifies that the list files are stored in the path /u/USER1/SAMPLE/list_files: `ENVAR("QPPLISTFILES=/u/USER1/SAMPLE/list_files/")` To learn more about specifying environment variables using Language Environment runtime options, see *Language Environment Customization*.

**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 z/OS Debugger 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 z/OS Debugger 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 z/OS Debugger manual.

**EQA1990E** **Invalid session parameter - *session_parameter***

## Explanation

Conversation initialization failed. A batch program, attempting to start an LU 6.2 debug session, has passed an invalid Session Parameter. For example, LU2 or MFI has been specified for session type or a session ID longer than eight characters has been specified. For a description of the Session Parameter and its contents, see the z/OS Debugger manual.

## Programmer response

Correct the Session Parameter and invoke the batch application again.

**EQA1991E** **CICS terminal TERM is not accessible**

## Explanation

The terminal id specified to receive z/OS Debugger screen was detected but not acquired.

## Programmer response

Correct the z/OS Debugger Term Id using DTCN Replace function or logon to already defined one.

**EQA1992E** **Missing workstation parameter**

## Explanation

Keywords APPC&, TCPIP&, VADAPPC&, and VADTCPIP& require a workstation ID to be entered.

## Programmer response

Correct or enter the workstation destination name.

**EQA1993E** **Invalid TCP/IP portid parameter**

## Explanation

Keywords VADTCPIP& or TCPIP& require a port ID to be entered. The value of this port id ranges from 1 to 65535 ('FFFF'x). If not entered or in error, a default value of 8001 is used.

## Programmer response

Correct or enter the TCP/IP port ID.

**EQA1994E** **There is no load module named *loadmod_name*.**

## Explanation

Load module qualification is referring to a load module that cannot be located.

## Programmer response

The load module might be missing or it might have been loaded before z/OS Debugger was first used. z/OS Debugger is aware of additional load modules only if they were FETCHed after z/OS Debugger got control for the first time.

**EQA1995S** **There is no CU named &&&& within load module &&&&.**

## Explanation

The compilation unit may be misspelled or missing.

**EQA1996S** **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.

**EQA1997I** **VTAM 3270 waiting for LU *lu_name***

## Explanation

This message is issued if the specified dedicated terminal is currently in use.

## Programmer response

End the session that is currently using the LU.

**EQA1998S** **VTAM 3270 *error_type* error, RC=*rc insert1 insert2 insert3***

## Explanation

An unrecoverable error occurred acquiring or communicating with a dedicated terminal. *error_type* is one of the following:

**RPL or INQUIRE RPL**
A nonzero return code was received from a VTAM RPL operation.

**ACB**
A nonzero return code was received from a VTAM ACB operation. This may result from improper installation of z/OS Debugger.

**MODCB**
A nonzero return code was received from a VTAM MODCB operation.

**Logic**
An internal logic error was detected.

**Function**
An internal logic error was detected.

**Storage**
Sufficient memory could not be obtained by the VTAM interface program.

**Undefined LU**
The VTAM Logical Unit specified in the `MFI%xxxxxxxxx:` parameter was not known to VTAM.

**Unknown**
An internal logic error was detected.

This message is issued whenever a permanent error is detected communicating with the dedicated terminal. A terminal condition is then signaled to LE causing program termination.

*Table 14. Definitions for error_type, insert1, insert2, and insert3*

| ***error_type*** | ***insert1*** | ***insert2*** | ***insert3*** |
|---|---|---|---|
| RPL or INQUIRE RPL or No default LOGMODE | *ggnnoomm* where: *gg* - CsFlag, *nn* - CsFunc, *oo* - CSFlagOf, and *mm* - CSReqMod | *rcrnoooo* where: *rc*-R15, *rn*-R0, or *rcrn*-RPLSense if RtnCodeFdBk2 = 0404 or 0403, and *oooo*-Error Offset in EQAYVTAM | *ppddkkkk* where: *pp*-RPLCode, *dd*-RtnCode, and *kkkk*-FdBk2 |
| ACB | *ggnnoomm* where: *gg* - CsFlag, *nn* - CsFunc, *oo* - CSFlagOf, and *mm* - CSReqMod | Error Offset in EQAYVTAM | ACBERR |
| MODCB | *ggnnoomm* where: *gg* - CsFlag, *nn* - CsFunc, *oo* - CSFlagOf, and *mm* - CSReqMod | Error Offset in EQAYVTAM | *xxxxyyzz* where: *xxxx*-0000, *yy*-MODCB R0, and *zz*-MODCB R15 |
| Logic | *ggnnoomm* where: *gg* - CsFlag, *nn* - CsFunc, *oo* - CSFlagOf, and *mm* - CSReqMod | Error Offset in EQAYVTAM | 0 |
| Function | *ggnnoomm* where: *gg* - CsFlag, *nn* - CsFunc, *oo* - CSFlagOf, and *mm* - CSReqMod | Error Offset in EQAYVTAM | EQAYVTAM function code |
| Storage | *ggnnoomm* where: *gg* - CsFlag, *nn* - CsFunc, *oo* - CSFlagOf, and *mm* - CSReqMod | Error Offset in EQAYVTAM | 0 |
| Undefined LU | *ggnnoomm* where: *gg* - CsFlag, *nn* - CsFunc, *oo* - CSFlagOf, and *mm* - CSReqMod | Error Offset in EQAYVTAM | 0 |
| Unknown | *ggnnoomm* where: *gg* - CsFlag, *nn* - CsFunc, *oo* - CSFlagOf, and *mm* - CSReqMod | Error Offset in EQAYVTAM | 0 |

## Programmer response

If an ACB `error` is reported, check with your installer to ensure that the VTAM modifications required by z/OS Debugger have been made. If `Undefined LU error` is reported, check the `MFI%` operand of the TEST parameter to ensure that the correct dedicated terminal Logical Unit identifer was specified and that the terminal in known to VTAM.

Otherwise, contact IBM support.

**EQA1999I VTAM3270 acquired LU *lu_name***

## Explanation

This message is issued when the LU is acquired after EQA1997I is issued.

## Programmer response

None.

**EQA2000E The hardware required to support a referenced symbol is not present.**

## Explanation

A referenced symbol cannot be evaluated because the required hardware is not present. Binary Floating

Point requires 64-bit hardware. Decimal Floating Point requires 64-bit and decimal floating point hardware.

### Programmer response

Correctly qualify the referenced variable and retry the command.

**EQA2001E** **Syntax error at '*%1*'.**

### Explanation

Invalid syntax was detected while parsing the expression.

**EQA2002E** **Syntax error: The expression is incomplete.**

### Explanation

The specified expression is incomplete.

**EQA2003E** **Cannot find the symbol '*%1*'.**

### Explanation

The given symbol name cannot be located.

**EQA2004E** **The variable *%1* is undefined or is incorrectly qualified.**

### Explanation

The named variable cannot be located or is undefined.

### Programmer response

You need to qualify with a different block , or specify different IN/OF qualifiers.

**EQA2005E** **The variable *%1* is undefined, is incorrectly qualified, or has been removed due to optimization.**

### Explanation

The named variable cannot be located or is undefined.

### Programmer response

You need to recompile the program without the Optimize option, qualify with a different block, or specify different IN/OF qualifiers.

**EQA2009E** **Syntax error: Invalid literal starting at *%1*.**

### Explanation

An invalid character was found while parsing the expression.

**EQA2011E** ***%1* is not a valid operand.**

### Explanation

An unexpected operand was found.

**EQA2013E** ***%1* contains incompatible data type.**

### Explanation

An unexpected operand type was found.

**EQA2016E** **An unsupported operator/operand is specified.**

### Explanation

An operator or an operand was not understood, and therefore was not processed.

**EQA2043E** **Operation is not permitted in Browse Mode.**

### Explanation

The expression contains an operation that is not permitted when running in browse mode.

**EQA2046E** **A divisor of 0 is detected in a divide operation.**

### Explanation

The expression contains a divide operation where the divisor was determined to be zero.

**EQA2047E** **Array subscript or index *%1* is out of bounds.**

### Explanation

An array subscript or index must be between 1 and the number of elements in the array.

**EQA2048E** **Incorrect value for ODO variable *%1*.**

### Explanation

The ODO variable might not have been initialized, or the current value is out of range.

**EQA2049E** **Invalid specification of reference modification.**

## Explanation

The leftmost-character-position field or the length field contains invalid value.

**EQA2050E** **Program *%1* not found.**

## Explanation

A bad program name is specified in a CALL command and the processing is terminated unsuccessfully.

**EQA2051E** **The address of *%1* has been determined to be invalid.**

## Explanation

This can happen for items of one of the following cases:

- Items within a data record where the file is not active or the record area is not available
- Items in a structure following Occurs, depending on the item where the ODO variable was not initialized
- Items in the LINKAGE SECTION that are not based on a valid address

**EQA2052E** **Incorrect number of subscripts or indexes for *%1*.**

## Explanation

A data item defined as a table was referenced with an incorrect number of subscripts or indexes. The reference is not allowed.

**EQA2053E** **No subscript or index is allowed for *%1*.**

## Explanation

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.

**EQA2054E** **String contains non-numeric characters.**

## Explanation

The assignment of a string to a numerical variable cannot be performed, because the string contains non-numerical characters.

**EQA2055E** **Invalid sign for *%1* is found.**

## Explanation

The sign position of a signed data item contains an invalid sign. The item might not have been initialized.

**EQA2056E** **Invalid data for *%1* is found.**

## Explanation

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.

**EQA2057E** **Only session variables can 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.

**EQA2058W** **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.

**EQA2059E** **The environment is not yet fully initialized.**

## Explanation

You can STEP and try the command again.

**EQA2060E** **The variable *%1* has been optimized and cannot be accessed.**

## Explanation

Storage for the named variable was not allocated.

## Programmer response

Recompile the program without the Optimize option.

**EQA2061E** **The name *%1* is ambiguous and cannot be resolved.**

## Explanation

Names of data items might be ambiguous if they are not sufficiently qualified.

**EQA2201E** **The JUMPTO command is not allowed, perhaps because of optimization and SET WARNING is ON.**

## Explanation

The JUMPTO command is not recommended. For COBOL, this might be caused by optimization, or because register contents other than the code base cannot be guaranteed for the target. If SET WARNING is OFF, z/OS Debugger executes the command, but the results are unpredictable.

**EQA2246E** **The specified load module does not contain CU *CU_name*.**

## Explanation

The specified load module is known to z/OS Debugger but it does not contain a CU by the specified name.

**EQA2247E** **A load module name was not specified for CU *CU_name*. Both load module and CU names must be specified in the form LOADMOD::>CU.**

## Explanation

When an LDD is entered in Explicit Debug Mode, both load module and CU names must be specified.

**EQA2248I** **Explicit Debug Mode is now active.**

## Explanation

Explicit Debug Mode is now ON.

**EQA2249I** **Explicit Debug Mode is no longer active.**

## Explanation

Explicit Debug Mode is now OFF.

**EQA2250I** **Command not supported on the current platform.**

## Explanation

The command is not supported in the current environment (e.g., non-CICS)

**EQA2251I** **DTCN Pattern-match breakpoint disabled for:**

## Explanation

This is the title line for the LIST DTCN command.

**EQA2252I** **CADP Pattern-match breakpoint disabled for:**

## Explanation

This is the title line for the LIST CADP command.

**EQA2253I** **Load module = &&&& CU = &&&&**

## Explanation

This message lists the load module and CU names output by the LIST DTCN command.

**EQA2254I** **Program = &&&& CU = &&&&**

## Explanation

This message lists the program and compile unit names output by the LIST CADP command.

**EQA2255I** **This program and/or compile unit is not in the pattern-match breakpoint list.**

## Explanation

The ENABLE CADP command is not allowed since this particular program and/or compile unit is not in the pattern-match breakpoint list.

**EQA2256I** **This load module and/or compile unit is not in the pattern-match breakpoint list.**

## Explanation

The ENABLE DTCN command is not allowed since this particular program is not in the pattern-match breakpoint list.

**EQA2257I** **The pattern-match breakpoint list is empty.**

## Explanation

There are no entries in the pattern-match breakpoint list.

**EQA2258R** **There is no SOAP (DFHNODE) channel in the current program.**

## Explanation

There is no channel named DFHNODE known to the current program.

**EQA2259E** **z/OS Debugger encountered an error evaluating the condition expression following WHEN for the break point. Use LIST AT to**

**view the break point and the expression.**

## Explanation

There was an error evaluating the expression entered for the WHEN condition for the break point. LIST AT can be used to view the expression. z/OS Debugger continues processing.

**EQA2260E** **Failure to connect to the remote debugger. Address: *address_name*. Port: *port_number*. IP sockets returned: *rc*. Possible cause: invalid IP address.**

## Explanation

Please check the TCPIP address and port specified and verify that the remote debugger daemon is listening.

**EQA2261E** **An error occurred while opening file: &&&&. The file may not exist, or is not accessible.**

## Explanation

An error during the initial processing (OPEN) of the file occurred.

**EQA2262E** **Ending location should be higher than starting location.**

## Explanation

Modify the command providing an ending location that is higher than starting location.

**EQA2263E** **Total requested amount exceed size of the container.**

## Explanation

Reduce size being requested. Use DESCRIBE CHANNEL to verify the size of the container.

**EQA2264I** ***** Global preferences file commands follow *****

## Explanation

Start of commands in the global preferences file.

**EQA2265I** ***** User preferences file commands follow *****

## Explanation

Start of commands in the user preferences file.

**EQA2266I** ***** Commands file commands follow *****

## Explanation

Start of commands in the commands file.

**EQA2267I** ***** Global preferences file commands end *****

## Explanation

End of commands in the global preferences file.

**EQA2268I** ***** User preferences file commands end *****

## Explanation

End of commands in the user preferences file.

**EQA2269I** ***** Commands file commands end *****

## Explanation

End of commands in the commands file.

**EQA2270I** **Global Preferences file exists: &&&&**

## Explanation

The global preferences file is opened successfully.

**EQA2271I** **The setting of DEFAULT LISTINGS is:**

## Explanation

This message header is for QUERY DEFAULT LISTINGS.

**EQA2272W** **A CICS Storage Violation has been detected. The leading/trailing check zone associated with the storage that starts at '&&&&'X for a length of &&&& has been damaged.**

## Explanation

z/OS Debugger detected a storage violation.

**EQA2273I** **&&&& &&&&**

## Explanation

This message contains the output from the DESCRIBE CHANNEL command.

**EQA2274I** **A CICS Storage Violation has not been detected.**

## Explanation

z/OS Debugger did not detect any storage violation

**EQA2275E** **Insufficient storage is available to process command.**

## Explanation

There was not enough main memory available to process the command.

**EQA2276E** **The listing file name given is too long.**

## Explanation

Under MVS data definition names are limited to 8 characters and dataset names are limited to 44 characters.

**EQA2277I** **&&&&**

## Explanation

Display results of a command.

**EQA2278E** **CHKSTGV command is only available in a CICS Environment.**

## Explanation

The command is not supported on Non-CICS Environments.

**EQA2279E** **The Language Environment attempted to present a Language Environment event to z/OS Debugger when the user program was executing in AMode(64). z/OS Debugger does not currently support Language Environment events in Amode(64). The event is ignored.**

## Explanation

z/OS Debugger currently supports AMode(64) only for assembler and disassembly.

**EQA2280E** **LIST CONTAINER command is only available in a CICS Environment.**

## Explanation

The command is not supported on Non-CICS Environments.

**EQA2281E** **DESCRIBE CHANNEL command is only available in a CICS Environment.**

## Explanation

The command is not supported on Non-CICS Environments.

**EQA2282E** **This command is not supported in this CICS Version/Release.**

## Explanation

You must use CICS TS 3.1 or later to be able to use this feature.

**EQA2283E** **There is no container with that name in the specified channel.**

## Explanation

The container name provided does not belong to the channel specified or does not exist.

**EQA2284E** **There are no containers to display.**

## Explanation

There are no containers known to the current program.

**EQA2285E** **There are no channels to display.**

## Explanation

There are no channels known to the current program.

**EQA2286E** **There is no channel with that name in this program.**

## Explanation

A channel with that name was not found. Verify the name of.

**EQA2287E** **There is no current channel in this program**

## Explanation

Read CICS Documentation on Channels and Containers.

**EQA2288S** **The module "*module_name*" cannot be loaded - it was NOT found.**

## Explanation

The LOAD MODULE request failed, the module was not found.

**EQA2289S** **The module "*module_name*" cannot be loaded - there is NOT enough storage to do the load.**

## Explanation

The LOAD MODULE request failed, there is not enough storage.

**EQA2290S** **The module "*module_name*" cannot be loaded.**

## Explanation

The LOAD MODULE request was unsuccessful.

**EQA2291S** **The module "*module_name*" cannot be deleted.**

## Explanation

The DELETE MODULE request was unsuccessful.

**EQA2292S** **The module "*module_name*" was NOT loaded by the z/OS Debugger and therefore CANNOT be deleted.**

## Explanation

Only modules loaded by z/OS Debugger may be deleted.

**EQA2293S** **The module "*module_name*" cannot be loaded because it was already loaded by z/OS Debugger.**

## Explanation

The LOAD MODULE request was unsuccessful.

**EQA2294I** **The LOADDEBUGDATA command for *CU_name* has been deferred until the CU appears.**

## Explanation

The indicated CU is not currently known to z/OS Debugger. The LOADDEBUGDATA will be run when the CU appears in a loaded module.

**EQA2295I** **The CU specified for the LOADDEBUGDATA command is already an assembler or LangX COBOL CU.**

## Explanation

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.

**EQA2296E** **The CU specified for the LOADDEBUGDATA command is not a disassembly CU.**

## Explanation

Only a disassembly CU can be identified as assembler CU.

**EQA2297E** **An error occurred while attempting to load the debug (EQALANGX) file for a specified CU.**

## 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.

**EQA2298S** **There is no compilation unit named *compile_unit_name*.**

## Explanation

The compilation unit (program) that you named could not be located by z/OS Debugger.

**EQA2299E** **The EQALANGX file does not match the object for *object_name*. The EQALANGX file cannot be used.**

## 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.

**EQA2300E** **A pattern of "*" is invalid.**

## Explanation

The NAMES EXCLUDE command does not allow a pattern of "*".

**EQA2301E** **Value too long to display.**

**EQA2302E** **Not allocated**

| | |
|---|---|
| **EQA2303E** | **The value specified in the fourth operand of the TEST runtime parameter is not valid.** |

## Explanation

The value specified before the colon in the fourth operand of the TEST runtime parameter is not in the correct format.

## Programmer response

Correct the specification of the fourth TEST operand.

| | |
|---|---|
| **EQA2304E** | **Format of value failed** |
| **EQA2305E** | **NULL** |
| **EQA2306E** | **Register not used** |
| **EQA2307E** | **Invalid string** |
| **EQA2308E** | **Divide by zero** |
| **EQA2309E** | **Invalid expression** |
| **EQA2310E** | **Expression not supported** |
| **EQA2311E** | **Incompatible types** |
| **EQA2312E** | **Expression validation failed** |
| **EQA2313E** | **Expression evaluation failed** |
| **EQA2314E** | **Evaluation not supported** |
| **EQA2315E** | **Expression not evaluated** |
| **EQA2316E** | **Variable not found** |
| **EQA2317E** | **Invalid value for update** |
| **EQA2318E** | **Update of value not allowed at this time** |
| **EQA2319E** | **Update of value not supported** |
| **EQA2320E** | **Operation not supported** |
| **EQA2321I** | **Please see log window for messages** |
| **EQA2322E** | **Invalid address** |
| **EQA2323E** | **Storage unit style is not supported** |
| **EQA2324E** | **Storage address style is not supported** |
| **EQA2325E** | **Register variable(s) out of range** |
| **EQA2326E** | **Invalid program name** |
| **EQA2327E** | **Failure loading view information** |
| **EQA2328E** | **Failure evaluating expression context. Contact IBM support.** |
| **EQA2329E** | **Frequency data is not allowed with this breakpoint type.** |
| **EQA2330E** | **Invalid line to set a line breakpoint** |
| **EQA2331E** | **Initialization failure - z/OS Debugger and front-end levels are incompatible** |
| **EQA2332E** | **Invalid storage data** |
| **EQA2333E** | **Incomplete update - a portion of storage is not updateable** |
| **EQA2334E** | **Storage is not updateable** |
| **EQA2335E** | **Failure updating storage contents** |
| **EQA2336E** | **Procedure name is an internal procedure, not an entry point** |
| **EQA2337E** | **Breakpoint type not supported** |
| **EQA2338I** | **Program at end of job** |
| **EQA2339E** | **A DLL load occurred. Current program location cannot be determined** |
| **EQA2340I** | **Target program(s) loaded - START/CALL required** |
| **EQA2341E** | **Insufficient storage to load view** |
| **EQA2342E** | **Program exception has occurred** |
| **EQA2343I** | **Debug session initialization complete** |
| **EQA2344E** | **Debug file name could not be found** |

## Explanation

The debug file name cannot be found. The following list describes some of the possible conditions that could cause this:

- The file was deleted from the system.
- The file was renamed.
- The user does not have authorization to access the file.

| | |
|---|---|
| **EQA2345E** | **Debug file version not supported** |
| **EQA2346I** | **Maximum number of debug files (256) reached** |
| **EQA2347E** | **Invalid debug file name** |
| **EQA2348E** | **Debug file format is invalid** |
| **EQA2349E** | **Debug file not supported - contains multiple "@PROCESS" statements** |
| **EQA2350E** | **Insufficient storage to read debug file** |
| **EQA2351E** | **I/O error reading debug file** |

| | |
|---|---|
| **EQA2352E** | **I/O error opening debug file** |
| **EQA2353E** | **Debug file CSECT name does not match compile unit CSECT name** |
| **EQA2354I** | **USE file processing has paused - USE file is still active** |
| **EQA2355E** | **Altering the PSW is not supported** |
| **EQA2356E** | **Program not auto-started - debug file name could not be found** |
| **EQA2357E** | **Program not auto-started - debug file version not supported** |
| **EQA2358E** | **Program not auto-started - max number of debug files (256) reached** |
| **EQA2359E** | **Program not auto-started - invalid debug file name** |
| **EQA2360E** | **Program not auto-started - debug file format is invalid** |
| **EQA2361E** | **Program not auto-started - insufficient storage** |
| **EQA2362E** | **Program not auto-started - I/O error reading debug file** |
| **EQA2363E** | **Program not auto-started - I/O error opening debug file** |
| **EQA2364E** | **Program not auto-started - failure processing debug information** |
| **EQA2365E** | **Exit point name must be a primary entry** |
| **EQA2366E** | **Exit breakpoints not allowed on internal procedures** |
| **EQA2367E** | **Failure processing debug information. Program compiled with NOTEST** |
| **EQA2368I** | **Search has wrapped** |
| **EQA2369E** | **Invalid link pointer** |
| **EQA2370I** | **End of link chain reached** |
| **EQA2371E** | **Invalid breakpoint label name** |
| **EQA2372E** | **Jump to location must be within currently active program** |
| **EQA2373E** | **Initialization failure - extended qualification is not supported** |
| **EQA2374E** | **SOM object not instantiated** |
| **EQA2375E** | **Field is not updatable** |
| **EQA2376E** | **Update of field failed** |
| **EQA2377E** | **Invalid data** |
| **EQA2378I** | **Application has terminated** |
| **EQA2379E** | **Internal error. Please, contact IBM support** |
| **EQA2380E** | **Jump to Location while at initialization is not allowed** |
| **EQA2381E** | **The target of the Jump to Location is invalid** |
| **EQA2382E** | **The target of the Run to Location is invalid** |
| **EQA2383I** | **The environment is not yet fully initialized. Use Step or Run.** |
| **EQA2384E** | **The Defer option is not permitted for Line Breakpoints.** |
| **EQA2385E** | **The entry point is not active or debug data is not available.** |
| **EQA2386E** | **You are monitoring an inaccessible or uninitialized variable.** |
| **EQA2387E** | **Cannot modify BreakPoints. Delete existing BreakPoint then add new.** |
| **EQA2388E** | **Cannot monitor this type of expression or variable** |
| **EQA2389E** | **The C component of the LE runtime has not been initialized.** |
| **EQA2390E** | **Exec Imminent. Click any button to continue.** |
| **EQA2391E** | **Time stamp on listing does not match time stamp on object.** |
| **EQA2392E** | **Unable to find requested executable module.** |
| **EQA2393E** | **z/OS Debugger has frozen this thread. Registers are not available.** |
| **EQA2394E** | **Stop at date field references.** |
| **EQA2395E** | **The entered expression is invalid, please check the expression.** |
| **EQA2396E** | **Expression breakpoints are not supported for this language.** |
| **EQA2397E** | **The expression has inaccessable or uninitialized data.** |
| **EQA2398E** | **No hooks present at statements, breakpoint not set.** |
| **EQA2399E** | **z/OS Debugger has frozen this thread. Call stack is not available.** |

| | |
|---|---|
| **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 z/OS Debugger. 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 z/OS Debugger, so z/OS Debugger 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 z/OS Debugger 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. z/OS Debugger 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** **z/OS Debugger 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, z/OS Debugger could not stop the application because a valid save area back chain did not exist. The back chain 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** **z/OS Debugger detected a missing or invalid z/OS Debugger SVC EQA00SVC(IGC0014E). The installed version is *vers*. DYNDEBUG is disabled.**

## Explanation

During initialization z/OS Debugger did not find a usable z/OS Debugger 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 z/OS Debugger 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 z/OS Debugger 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** **z/OS Debugger detected a missing or invalid z/OS Debugger SVC EQA01SVC(IGX00051). The installed version is *vers*. DYNDEBUG is disabled for read only programs.**

## Explanation

During initialization z/OS Debugger did not find usable z/OS Debugger 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 z/OS Debugger SVCs.

**EQA2425I** **z/OS Debugger detected a down level z/OS Debugger SVC EQA01SVC(IGX00051). The installed version is *vers*. Version 2 is required when using CICS with TRANISO=YES. DYNDEBUG is disabled.**

## Explanation

During initialization z/OS Debugger detected a downlevel z/OS Debugger SVC version. The Dynamic Debug facility is disabled for read only programs.

## Programmer response

Have your installer install the correct z/OS Debugger SVCs.

**EQA2426I** **z/OS Debugger detected a down level z/OS Debugger SVC EQA01SVC(IGX00051). The installed version is *vers*. Version 17 is required for this version of z/OS Debugger.**

## Explanation

During initialization z/OS Debugger detected a downlevel z/OS Debugger SVC version.

## Programmer response

Have your installer install the correct z/OS Debugger SVCs.

**EQA2427I** **z/OS Debugger detected a down level z/OS Debugger SVC EQA01SVC(IGX00051) or EQA00SVC(IGC0014E). EQA01SVC is version *VERS*. EQA00SVC is version *VERS*. EQA01SVC version 6 and EQA00SVC version 5 are required.**

## Explanation

During initialization z/OS Debugger detected a downlevel z/OS Debugger SVC version.

## Programmer response

Have your installer install the correct z/OS Debugger SVCs.

| | |
|---|---|
| **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 for :** |
| **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 z/OS Debugger 15.0.*m*** |

## Explanation

This message is used to place the z/OS Debugger logo, a time stamp, and copyright at the beginning of the line.

| | |
|---|---|
| **EQA2442I** | **5724-T07 Copyright IBM Corp. 1992, 2017** |

## Explanation

This message is used to place the z/OS Debugger 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** | **The DYNDEBUG status was not changed because SET DYNDEBUG can only be executed at the beginning of a debug session, before you use STEP or GO commands, or start code coverage** |

**Programmer response:**
For more information, see "SET DYNDEBUG command" on page 225.

| | |
|---|---|
| **EQA2454I** | **This CU is not AUTOMONITOR capable for expressions.** |

## Programmer response

Refer to the description of the SET AUTOMONITOR command in the *IBM z/OS Debugger 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

z/OS Debugger has stopped in a program and the value in GPR 13 is not a valid address.

## Programmer response

z/OS Debugger will attempt to continue. However, some information may be missing or incorrect.

| | |
|---|---|
| **EQA2456I** | **Returning from enclave where z/OS Debugger was initialized. Handling of non-LE events has been suspended. Debugging of non-LE programs is suspended in this z/OS Debugger session.** |

## Explanation

The Language Environment was invoked with TEST/NOPROMPT or NOTEST causing z/OS Debugger to be initialized during an enclave that was not the top enclave. The enclave in which z/OS Debugger was initialize is now terminating. z/OS Debugger will no longer intercept non-Language Environment events and, therefore, you can no longer debug non-Language Environment programs.

## Programmer response

If you do not need to debug higher-level, non-Language Environment programs or to intercept non-Language Environment events, no action is required. Otherwise, re-run the job without NOPROMPT or with the CALL to CEETEST, PLITEST or ctest() in a higher-level enclave.

**EQA2457E** **This command is not supported in Browse Mode.**

## Explanation

This command is not supported in Browse Mode because it either alters storage or registers or it alters the control flow.

**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 z/OS Debugger session.**

## Explanation

The z/OS Debugger was invoked with an EQAOPTS options module that specified SVCSCREEN OFF. z/OS Debugger will not intercept non-Language Environment events and, therefore, debugging of non-Language Environment programs will be limited.

## Programmer response

If you do not need to debug non-Language Environment programs or to intercept non-Language Environment 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 z/OS Debugger session.**

## Explanation

The z/OS Debugger was invoked with an EQAOPTS options module that specified CONFLICT=NOOVERRIDE. z/OS Debugger will not intercept non-Language Environment events and, therefore, debugging of non-Language Environment program will be limited.

## Programmer response

If you do not need to debug non-Language Environment programs or to intercept non-Language Environment events, no action is required. Otherwise, you must terminate the prior use of SVC SCREENING (TCBSVCS, TCBSVCSP, TCBSVCA2) before starting z/OS Debugger or have your installer provide an EQAOPTS that specifies CONFLICT=OVERRIDE. CONFLICT=OVERRIDE allows z/OS Debugger to save and restore the previous use of SVC SCREENING (TCBSVCS, TCBSVCSP, TCBSVCA2).

**EQA2460I** **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 z/OS Debugger session.**

## Explanation

The z/OS Debugger was invoked with an EQAOPTS options module that specified CONFLICT=OVERRIDE. z/OS Debugger will save and restore the SVC Screening values.

SVC SCREENING is indicated by TCBSVCS, TCBSVCSP, and TCBSVCA2. These values are saved during z/OS Debugger startup and restored at z/OS Debugger termination.

**EQA2461W** **Code page in the VADSCP suboption is not between 00001 and 99999. Default code page 00037 is assumed.**

## Explanation

z/OS Debugger 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 z/OS Debugger 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.

## Programmer response

See z/OS Support for Unicode: Using Conversion Services book for explanation of return code, reason code, and appropriate action.

**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.

**EQA2466I** **&&&&**

## Explanation

This message lists the names output of the NAMES command.

**EQA2467I** **The EQALANGX debug data also contains data for the following CUs:**

## Explanation

This is the header used to display the additional CSECTs included in the EQALANGX data for the current CU.

**EQA2468I** **&&&& &&&&**

## Explanation

Used to display CSECTs also included in the EQALANGX data for the current CU.

**EQA2469I** **The SVC Screening required for z/OS Debugger will be merged with the SVC Screening already active for COPE.**

## Explanation

COPE's usage of SVC Screening will be restored at the end of this z/OS Debugger session. z/OS Debugger was invoked with an EQAOPTS options module that specified MERGE. z/OS Debugger will save the COPE screening values on entry, merge them with z/OS Debugger's during the execution of the program, and then restores the original COPE SVC Screening values when z/OS Debugger terminates. SVC Screening is indicated by TCBSVCS, TCBSVCSP, and TCBSVCA2.

## Programmer response

If you did not intend to MERGE the z/OS Debugger SVC Screening tables with another program, modify your EQAOPTS to specify what you require.

**EQA2471I** **Your Language Environment enabled application has stopped at a location where a non Language Environment compliant R13 savearea backchain exists. At location *LOCN*. Commands such as LIST CALLS will not operate properly.**

## Explanation

In order to be Language Environment compliant your application needs to follow the Language Environment rules.

## Programmer response

Follow the Language Environment rules.

**EQA2472I** **Your Language Environment enabled application has stopped at a location where the non Language Environment compliant R13 savearea backchain no longer exists. At location *LOCN*.**

## Explanation

The savearea has become Language Environment compliant and all commands will work properly.

## Programmer response

None.

**EQA2473I** **EQAOPTS setting SUBSYS=*ssss* is in effect for use when reading source code from a library system.**

## Explanation

In a non-CICS environment, an EQAOPTS with a SUBSYS specification of *ssss* was found. If z/OS Debugger needs to allocate a C, C++ or Enterprise PL/I source data set from a library system that stores the source in a data set that has a DSORG of DA or VSAM, then the SUBSYS=*ssss* allocation parameter will be used when z/OS Debugger allocates the data set.

## Programmer response

If you need this support, ensure that the *ssss* subsystem is running on the system that you are running z/OS Debugger on.

**EQA2474I** **z/OS Debugger 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 z/OS Debugger encountered a hook or other event when the PSW specified AMODE64. z/OS Debugger 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 (an expression that contains a conditional operator such as =, ^= or <=) 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 z/OS Debugger 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.

**EQA2490I** **The DEFAULT VIEW is now &&&&.**

**EQA2491I** **The setting of DEFAULT VIEW is &&&&.**

**EQA2492I** **The setting of current view is &&&&.**

**EQA2493I** **Program was stopped due to watch breakpoint on COBOL Level-88 condition-name &&&&.**

**EQA2494I** **Program was stopped due to watch breakpoint on COBOL Level-88 condition-name &&&& and the condition &&&& evaluated to be true.**

**EQA2495I** **Evaluation of the conditional expression &&&& failed.**

**EQA2496I** **The setting of LDD is now &&&&.**

## 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 &&&&.**

## 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 z/OS Debugger 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-Language Environment environment to load a load module that contained one or more Language Environment CUs. These CUs cannot be created until Language Environment is active. CUs will be created only for non-Language Environment 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.

**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 exists**

## Explanation

The add command failed because a profile for that terminal is already stored in the z/OS Debugger 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 z/OS Debugger 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 z/OS Debugger 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 Show failed - profile does not exist**

### Explanation

The profile for the Terminal does not exist in the z/OS Debugger 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 z/OS Debugger Profile Repository.

**EQA2510I DTCN closed**

### Explanation

DTCN deleted all profiles stored in the z/OS Debugger Profiles 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.

**EQA2513I One or more of the LoadMod or CU fields have been set to an '*'.**

### Explanation

When a LoadMod or CU field contains data and the corresponding field does not contain data, then the corresponding field will be set to an asterisk (*).

### Programmer response

If this is not what you want, then enter the data you want in the corresponding LoadMod field, CU field, or both.

**EQA2514I z/OS Debugger profile saved**

### Explanation

A profile was saved in the z/OS Debugger Profile Repository.

**EQA2515I z/OS Debugger profile replaced**

### Explanation

Existing profile was updated in the z/OS Debugger Profile Repository.

**EQA2516I z/OS Debugger profile deleted**

### Explanation

Existing profile was deleted from the z/OS Debugger 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 DTCNFORCE*xxxx* options in macro EQAXOPT.

## Programmer response

Enter a resource name in the field.

**EQA2520W** **Terminal mismatch. Press PF10 to set the value to the current terminal identifier.**

## Explanation

The terminal ID at the highlighted cursor location does not match the current terminal.

## Programmer response

Press PF10 to set the value to the current terminal identifier.

**EQA2521E** **Invalid field. PF10 may only be used to reset terminal or display ID.**

## Explanation

The field at the current cursor location is invalid for the PF10 action. This key may only be pressed when the current cursor location is in the terminal or display ID fields.

## Programmer response

If you wish to update the terminal ID or display ID fields, move the cursor to one of those fields and press PF10 again. Otherwise, no further action is required.

**EQA2522E** **You are not authorized to modify or delete another user's profile.**

## Explanation

You have tried to modify or delete another user's profile, but your *USERID* does not have sufficient authority to do that. Your *USERID* needs to be authorized to access the `EQADTOOL.DTCNCHNGEANY` resource in your security facility.

## Programmer response

Refer to the *IBM z/OS Debugger Customization Guide* for information on granting authority for DTCN functions.

**EQA2523E** **Duplicate profile exists. Specify more resources and press PF4 to Save.**

## Explanation

You have tried to activate an inactive profile, but there is already an active profile which specifies these resources.

## Programmer response

Add or change debug resources in your profile and press PF4 to save the new profile.

**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.

**EQA2608E** **The MONITOR command entered cannot be successfully executed because the Monitor Command Table is full and AUTOMONITOR is ON. You must either turn off AUTOMONITOR, replace an existing Monitor command, or clear an existing Monitor command. Use the LIST MONITOR command to list the Monitor commands in the Monitor Command Table. NOTE: When AUTOMONITOR is ON it will occupy 1 (for CURRENT or PREVIOUS) or 2 (for BOTH) slots in the Monitor Command Table and will not be included in the list of Monitor commands from the LIST MONITOR command.**

## Explanation

All 99 slots in the Monitor Command Table are currently being used.

## Programmer response

You must either turn off AUTOMONITOR, replace an existing Monitor command, or clear an existing Monitor command before this command can be successfully executed.

**EQA2609E** **The AUTOMONITOR command entered cannot be successfully executed because the Monitor Command Table is full. You must clear one or more existing Monitor commands. Use the LIST MONITOR command to list the Monitor commands in the Monitor Command Table. NOTE: AUTOMONITOR will occupy 1 (for CURRENT or PREVIOUS) or 2 (for BOTH) slots in the Monitor Command Table and will not be included in the list of Monitor commands from the LIST MONITOR command.**

## Explanation

All 99 slots in the Monitor Command Table are currently being used.

## Programmer response

You must clear one or more existing Monitor commands before this command can be successfully executed.

**EQA2610E** **There is no current channel in program.**

## Explanation

z/OS Debugger did not find a current channel in the program. This could be because the program has not been invoked with a channel.

## Programmer response

Go to the CICS Transaction Server V3.1 (or later) information center and look for the topic "The current channel", which describes a current channel and gives examples of how to invoke a channel.

**EQA2611E** **The channel *channel_name* was not found.**

## Explanation

z/OS Debugger could not find the channel for the program. The name might be misspelled.

## Programmer response

Verify that you have spelled the channel name correctly. If you aren't sure about the channel name, use `DESCRIBE CHANNEL *` command for a list of channels known.

**EQA2612E** **There are no channels to display.**

## Explanation

There are no channels known to the current program.

## Programmer response

Go to the CICS Transaction Server V3.1 (or later) information center and look for the topics "Creating a channel" or "The scope of a channel" for instructions and explanations.

**EQA2613E** **There are no containers to display.**

## Explanation

z/OS Debugger could not find any channels known to the program. This could be because no channels have been created or assigned.

## Programmer response

Go to the CICS Transaction Server V8.1 (or later) information center and look for the topics "Discovering which containers were passed to a program" or "Discovering which containers were returned from a link" to learn more about finding or identifying containers.

**EQA2614E** **There is no container with that name in the specified channel.**

## Explanation

z/OS Debugger could not find the container in the channel. The names might be misspelled.

## Programmer response

Verify that you have spelled the container name and channel name correctly. After you make any corrections, retry the command. If you aren't sure about the channel name or container name, use `DESCRIBE CHANNEL *` command.

**EQA2615E** **This command is not supported in this CICS Version/Release.**

## Explanation

You must use CICS Transaction Server V3.1 or later to be able to use this feature.

**EQA2616E** **The DESCRIBE CHANNEL command is available only in a CICS environment.**

## Explanation

You cannot use the `DESCRIBE CHANNEL` command in a non-CICS environment.

**EQA2617E** **The LIST CONTAINER command is available only in a CICS environment.**

## Explanation

You cannot use the `LIST CONTAINER` command in a non-CICS environment.

**EQA2618E** **There is no SOAP (DFHNODE) channel in the current program.**

## Explanation

There is no channel named DFHNODE known to the current program.

## Programmer response

Follow CICS directions on creating channels and containers.

**EQA2619E** **The CHKSTGV command is available only in a CICS environment.**

## Explanation

You cannot use the `CHKSTGV` command in a non-CICS environment.

**EQA2620E** **The requested bytes exceed the end of the container.**

## Explanation

Reduce size being requested. Use `DESCRIBE CHANNEL` to verify the size of the container.

**EQA2621E** **Ending location should be higher than starting location.**

## Explanation

Modify the command providing an ending location that is higher than starting location.

**EQA2622E** **The SET IGNORELINK command is available only in a CICS environment**

## Explanation

You cannot use the SET IGNORELINK command in a non-CICS environment.

**EQA2627E** **The command CALL %FM is available only in a CICS environment.**

## Explanation

This command requires CICS.

**EQA2628E** **IBM File Manager for z/OS is not installed in this CICS region.**

## Explanation

The CALL %FM command requires that IBM File Manager be installed and customized for CICS.

## Programmer response

Verify that IBM File Manager is installed and customized for CICS as described in the topic "Updating the CICS start up procedures" in the Customization Guide for IBM File Manager.

**EQA2629E** **IBM File Manager for z/OS could not be started.**

## Explanation

This command requires that IBM File Manager be installed.

## Programmer response

Verify that IBM File Manager is installed and customized for CICS as described in the topic "Updating the CICS start up procedures" in the Customization Guide for IBM File Manager.

**EQA2630E** **AT ENTRY or AT EXIT is not allowed for a C or C++ nested block.**

## Explanation

You cannot set a breakpoint at the entry or exit point of a nested block.

**EQA2631E** **Invalid character found in an address field.**

## Explanation

z/OS Debugger found an invalid character in the base address field or in the address column of the Memory window. You can put only hexadecimal characters or the G or R commands in those areas.

## Programmer response

Type in hexadecimal characters, the G or R commands, or clear any characters you might have accidentally entered. Then press Enter.

**EQA2632E** **Invalid character found in a data field.**

## Explanation

z/OS Debugger found an invalid character in the data column of the Memory window. You can put only hexadecimal characters or the G or R commands in those areas.

## Programmer response

Type in hexadecimal characters, the G or R commands, or clear any characters you might have accidentally entered. Then press Enter.

**EQA2633E** **Invalid command found in a history entry field.**

## Explanation

z/OS Debugger found an invalid command in the history entry field of the Memory window. You can put only the G or R commands in the history entry field.

## Programmer response

Type in hexadecimal characters, the G or R commands, or clear any characters you might have accidentally entered. Then press Enter.

**EQA2634E** **Multiple changes found in a history entry field.**

## Explanation

z/OS Debugger found multiple changes in a history entry field of the Memory window. You can only enter one command at a time.

## Programmer response

Clear the extra characters, then press Enter.

**EQA2635E** **The FIND command is not valid in the Memory window.**

## Explanation

You cannot use the FIND command in the Memory window.

## Programmer response

Do not use the FIND command in the Memory window.

**EQA2636E** **Invalid scroll amount is specified for the Memory window.**

## Explanation

You cannot use any one of the following scroll amounts for the Memory window: TOP, BOTTOM, MAX, and TO.

**EQA2637E** **Invalid window operation. The logical window not assigned to a physical window.**

## Explanation

You cannot use this command on a logical window that has not been assigned to a physical window.

## Programmer response

Use one of the following methods to assign the logical window to a physical window:

- Enter the SWAP command to assign the logical window you specified to a physical window.
- Enter the SET SCREEN command to assign the logical window you specified to a physical window.
- Enter the PANEL LAYOUT command to choose a window layout that assigns the logical window you specified to a physical window.

**EQA2638E** **Invalid Memory window width.**

## Explanation

The width of the physical window assigned to the Memory window is less than the full screen width.

## Programmer response

Do one of the following:

- Enter the SIZE SET SCREEN command and specify a width that is at least the same size as the full screen width.
- Enter the PANEL LAYOUT command to choose a window layout that assigns the Memory window a physical window with full screen width.

**EQA2639E** **Invalid Amode value is specified.**

## Explanation

The valid Amode values are 24 and 31.

## Programmer response

Enter a valid Amode value.

**EQA2640E** **SCROLL LEFT is not valid in the Memory window.**

## Explanation

You cannot enter the SCROLL LEFT command in the Memory window. The Memory window displays memory content in the entire width of the window.

**EQA2641E** **SCROLL RIGHT is not valid in the Memory window.**

## Explanation

You cannot enter the SCROLL RIGHT command in the Memory window. The Memory window displays memory content in the entire width of the window.

**EQA2642E** **Top of the memory area is reached.**

## Explanation

You have reached the top of memory space that the Memory window can display.

## Programmer response

Do not enter SCROLL UP command.

**EQA2643E** **Bottom of the memory area is reached.**

## Explanation

You have reached the bottom of memory space that the Memory window can display.

## Programmer response

Do not enter SCROLL DOWN command.

**EQA2644E** **L and E letters cannot be both used.**

## Explanation

Look at the Window Layout Select Panel. Verify that L and E are not used in the panel layout.

**EQA2645E** **SCROLL TO is not valid in the Memory window.**

## Explanation

You cannot enter the SCROLL TO command in the Memory window.

**EQA2661W** ***variable_name*** **has a length that is greater than its declared maximum. Breakpoint cannot be set until expression is fixed.**

## Explanation

The variable named has been declared as VARYING with length *n*, but its current length is greater than *n*. The variable may be unusable.

## Programmer response

Check the length of the variable.

**EQA2662E** ***identifier*** **is undefined.**

## 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.

**EQA2663E** **Unable to display the variable based on an invalid pointer.**

## Explanation

The result from the expression evaluation cannot be displayed. For example, the basing pointer has a zero or uninitialized value.

**EQA2664E** **The .dbg file for *compile_unit_name*, created by the FORMAT(DWARF) suboption of the DEBUG compiler option cannot be found. Use of the SET SOURCE command to indicate the location of the source file is not allowed at this time.**

## Explanation

A .dbg file containing the debugging tables is not available.

## Programmer response

Make sure the .dbg file for the program is available. If you know the location of the .dbg file, use the SET DEFAULT DBG command to specify the location of the file.

**EQA2665E** **The .mdbg file for *compile_unit_name*, cannot be found and you specified MDBG,YES in your EQAOPTS or exported environment variable EQA_USE_MDBG=YES.**

## Explanation

The .mdbg file containing the debugging tables cannot be found. Some of the conditions that could cause this are:

- the .mdbg file was deleted
- the .mdbg file was moved to a new location
- you do not have RACF authorization to access the file

## Programmer response

Make sure the file is in the default location or use the EQAUEDAT user exit, the EQAMDBG DD statement, or export EQA_MDBG_PATH environment variable before starting your debug session to indicate an alternate location. If your debug session is already started, use the SET DEFAULT MDBG or SET MDBG commands.

**EQA2666E** **The .mdbg file for *compile_unit_name*, cannot be found. Use of the SET SOURCE command to indicate the location of the source file is not allowed at this time.**

## Explanation

A .mdbg file containing the source, debugging tables, or both is not available.

## Programmer response

Make sure the .mdbg file for the program is available. If you know the location of the .mdbg file, use the SET DEFAULT MDBG or SET MDBG command to specify the location of the file.

**EQA2667E** **The location for the .mdbg file *File_Name*, for load module or DLL *Load_Module_Name*, could not be validated.**

## Explanation

The .mdbg file containing the debug tables could not be validated. Some of the possible conditions that could cause this are:

- the .mdbg file was deleted
- the .mdbg file was moved to a new location
- you do not have RACF authorization to access the file
- the contents are not compatible with the load module or DLL

### Programmer response

Make sure the correct .mdbg file for the program is available.

**EQA2668W** **The .mdbg file for load module or DLL *Load_Module_Name*, has already been validated. You cannot assign a new .mdbg file.**

### Explanation

The current .mdbg file for the load module or DLL is correct.

### Programmer response

None.

**EQA2669E** **The MDBG setting in your EQAOPTS or EQA_USE_MDBG is NO. The command is not allowed.**

### Explanation

The MDBG setting in your EQAOPTS or EQA_USE_MDBG is NO.

### Programmer response

Modify the MDBG setting to YES or EQA_USE_MDBG to YES.

**EQA2670E** **The MDBG setting in your EQAOPTS or EQA_USE_MDBG is YES. The command is not allowed.**

### Explanation

The MDBG setting in your EQAOPTS or EQA_USE_MDBG is YES

### Programmer response

Modify the MDBG setting to NO or EQA_USE_MDBG to NO.

**EQA2671E** **The source for *compile_unit*, was extracted from a .mdbg file and cannot be changed.**

### Explanation

Your .mdbg file contains the captured source.

### Programmer response

Rebuild your .mdbg file to make sure it uses the proper source files.

**EQA2672E** **The .mdbg file *mdbg_file_name* for load module or DLL *load_module_name* does not match. The .mdbg file cannot be used.**

### Explanation

Make sure you provide the location of the correct .mdbg file.

### Programmer response

Rebuild your .mdbg file or specify the proper location.

**EQA2673E** **An .mdbg file was found but the debug data does not match the load module or DLL for program *cu_name***

### Explanation

Make sure you provide the location of the correct .mdbg file.

### Programmer response

Rebuild your .mdbg file or specify the proper location.

**EQA2674E** **There are no .mdbg files to display.**

**EQA2675E** **The .mdbg file associated with *Load_Module_Name* is *mdbg_file_name***

**EQA2676I** **TRACE LOAD is already started.**

### Explanation

The TRACE LOAD command has been issued already.

**EQA2677I** **TRACE LOAD is not active.**

### Explanation

The TRACE LOAD command has not been issued previously.

**EQA2678I** **The following were loaded:**

**EQA2679I** ***Load Module Name* loaded from *Load Library Name*.**

**EQA2680E** **There are no LDD commands established.**

## Explanation

A `LIST LDD` or `CLEAR LDD` command was issued but there are no LDD commands established.

**EQA2681E** **No LDD command was established for LDD *ldd-number*.**

## Explanation

A `CLEAR LDD` command was issued but the number is greater than the highest numbered LDD command.

**EQA2682I** ***LDD-command-entry***

## Explanation

A `LIST LDD` command output is being formatted for display.

**EQA2683I** **Debug information could not be located for any CUs in load module *load module* name.**

## Explanation

No debug information could be located for any CUs in this load module. It appears that all CUs were compiled using `NOTEST`.

## Programmer response

To qualify to a CU in this module, the user must either compile at least one CU with debug information (e.g. TEST compiler option) or issue the `SET DISASSEMBLY ON` command prior to the `SET QUALIFY LOAD` command.

**EQA2684E** **Prefix command not supported for current CU.**

## Explanation

The CU was not compiled with the correct compile option to support the `M` or the `L` prefix command.

## Programmer response

User must compile with C/C++ Z/OS 2.1 or later using the DEBUG(FORMAT(DWARF)) compile option.

**EQA2685E** **Specified operand is not supported for current programming language.**

## Explanation

An operand is not supported for M and L prefix commands in C/C++.

**Prerequisite:** The user must enter M or L prefix command with no operand specified.

**EQA3001I** **.**

**EQA3002I** **( | )**

**EQA3003I** **Licensed Materials - Property of IBM**

**EQA3004I** **Copyright IBM Corp.| All Rights Reserved**

**EQA3005I** **US Government Users Restricted Rights - Use, duplication**

**EQA3006I** **or disclosure restricted by GSA ADP Schedule Contract**

**EQA3007I** **with IBM Corporation.**

**EQA3009I** **** Opened File List**

**EQA3010I** **|**

**EQA3011I** **** End Opened File List**

**EQA3012I** **** Job Search List**

**EQA3013I** **** End Job Search List**

**EQA3014I** **** Default Search List**

**EQA3015I** **** End Default Search List**

**EQA3016I** **** Referenced List**

**EQA3017I** **| | |**

**EQA3018I** **** End Referenced List**

**EQA3019E** **Container name exceeds maximum**

**EQA3020E** **File name exceeds maximum**

**EQA3021E** **Open failure on DD:JCLLIB, specifies default concatenation list**

**EQA3022E** **Number of containers exceeds default concatenation limit**

**EQA3023E** **Container name read from file too long**

**EQA3024E** **Number of containers on JCLLIB exceeds limit**

**EQA3025E** **Attempt to nest source too deeply**

**EQA3026E** **Number of source files used exceeds limit**

**EQA3027E** **File name for INCLUDE or external PROC name is too long**

| | |
|---|---|
| **EQA3028E** | **INCLUDE file or cataloged PROC not found: *name*** |
| **EQA3030E** | **Container Name parameter is not valid** |
| **EQA3031E** | **File Name parameter is not valid** |
| **EQA3032E** | **Open failure on DD:INFILE, specifies JCL file** |
| **EQA3033E** | **Comment continuation flag was not followed by valid card** |
| **EQA3034E** | **End of file after comment continuation flag** |
| **EQA3038E** | **DLM= delimiter not two characters** |
| **EQA3039E** | **DLM= parameter on DD statement < 2 chars.** |
| **EQA3040E** | **DLM= contains & not followed by &** |
| **EQA3043E** | **Illegal statement found in PROC definition** |
| **EQA3046E** | **Illegal statement within include group** |
| **EQA3048E** | **PEND without matching procedure invocation** |
| **EQA3050E** | **Memory allocation failure** |
| **EQA3054E** | **Includes nested too deeply** |
| **EQA3058E** | **Symbolic name too long** |
| **EQA3059E** | **Multiple JOB cards** |
| **EQA3062E** | **Too many parms** |
| **EQA3063E** | **Parenthesis and apostrophe nesting exceeded** |
| **EQA3065E** | **If nesting exceeds limit** |
| **EQA3066E** | **ENDIF without IF** |
| **EQA3067E** | **Keyword parameter or SET statement missing =** |
| **EQA3068E** | **JCLLIB missing ORDER=** |
| **EQA3070E** | **EXEC card has null first parameter** |
| **EQA3071E** | **EXEC card PGM= has null value** |
| **EQA3072E** | **DD concatenation without DD statement** |
| **EQA3073E** | **Missing JOB card** |
| **EQA3074E** | **DD after JOB not JOBLIB or JOBCAT or PROCLIB** |
| **EQA3075E** | **Multiple JOBCAT statements** |
| **EQA3076E** | **Multiple JOBLIB statements** |
| **EQA3078E** | **Illegal statement in procedure** |
| **EQA3079E** | **PROC nesting exceeded limit** |
| **EQA3080E** | **Query empty context stack** |
| **EQA3081E** | **PROC stack underflow** |
| **EQA3082I** | **Statement: \| \| \| \|** |
| **EQA3083I** | **Substitution: \| \| \| \|** |
| **EQA3084E** | **Too many steps in job** |
| **EQA3086E** | **StepName.DDName too long** |
| **EQA3087E** | **Could not find step named on override DD** |
| **EQA3088E** | **Could not add DD; no PGM= steps found** |
| **EQA3089I** | **Added to Proc:\| \| \| \|** |
| **EQA3090I** | **Override Proc:\| \| \| \|** |
| **EQA3091E** | **Fatal Error: \| \| \| \|** |
| **EQA3092E** | **Statement label is too long** |
| **EQA3093E** | **Statement parameter text is too long** |
| **EQA3095E** | **Instream PROC has no name** |
| **EQA3096E** | **Instream PROC PEND not found before end of file** |
| **EQA3097I** | **Input parameters:** |
| **EQA3098E** | **Internal error** |
| **EQA3100W** | **Could not substitute Db2 run unit for SYSTSIN *** |
| **EQA3101W** | **Could not open\|** |
| **EQA3102W** | **Could not find OUTPUT spec named\|** |
| **EQA3103W** | **Could not open DD DDITV02 for Db2/IMS** |
| **EQA3104W** | **Could not read DD DDITV02** |
| **EQA3105W** | **Could not substitute Db2 run unit for DSNMVT01** |
| **EQA3108W** | **Could not find IMS program name** |
| **EQA3109W** | **Could not find IMS PSB name\|** |
| **EQA3110E** | **Internal error** |
| **EQA3111E** | **Override not proceeded by named DD stmt** |
| **EQA3112E** | **Open failure on DD:ATTRBOUT, specifies MA flat file** |
| **EQA3113E** | **Open failure on DD:RDPRINT, specifies logfile** |
| **EQA3119E** | **Data set name is too long:\|** |

| | |
|---|---|
| **EQA3120I** | **5724-T07: z/OS Debugger** |
| **EQA3130I** | **Warning found in \| \| at line\|** |
| **EQA3131I** | **Warning found in \| \| at line\|** |
| **EQA3140E** | **Invalid input:\|** |
| **EQA3141E** | **Duplicate instream PROC name \| encountered** |

**Explanation:**
Since JES accepts duplicate in-stream PROC statements, this message reports when these duplicate statements are encountered.

**System action:**
Processing continues by matching the subsequent EXEC PROC statement to the appropriate in-stream PROC statement.

**User response:**
No action is required.

| | |
|---|---|
| **EQA3142E** | **symbolic variable value is not set at file\| in line\|** |
| **EQA4000E** | **The length of CSECT *CSectName* in memory is X'*LMLength*' which does not match the length of X'*LangXLength*' found in the EQALANGX data** |

## Explanation

The EQALANGX data does not correspond to the CSECT in the loaded load module because the CSECT length does not match.

## Programmer response

Regenerate the EQALANGX data or ensure that the matching object has been link-edited into the current load module.

| | |
|---|---|
| **EQA4001E** | **The object code at offset X'*offset*' in CSECT *CSectName* in memory is X'*LMObject*' which does not match the instructions X'*LangXObject*' found at that offset in the EQALANGX data.** |

## Explanation

The EQALANGX data does not correspond to the CSECT in the loaded load module because the object code at the specified offset does not match.

## Programmer response

Regenerate the EQALANGX data or ensure that the matching object has been link-edited into the current load module.

| | |
|---|---|
| **EQA4002E** | **The EQALANGX data for *language* CSECT *CSectName* is for a different programming language.** |

## Explanation

The EQALANGX data does not correspond to the CSECT in the loaded load module because CSECT was coded in the specified programming language but the EQALANGX data is for a different programming language.

## Programmer response

Regenerate the EQALANGX data or ensure that the matching object has been link-edited into the current load module.

| | |
|---|---|
| **EQA4003W** | **The Debug File creation date does not match the object for *CompileUnit*, 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.

| | |
|---|---|
| **EQA4004I** | **The setting of *SET-command keyword* is *query-status*.** |

## Explanation

The status of the object of a SET command is displayed when QUERYed individually.

| | |
|---|---|
| **EQA4005I** | **You cannot use the QUERY QUALIFY command in remote debug mode. However, the Modules view displays a list of currently loaded modules. You can expand each node on the list to view the compile unites in each module.** |

## Explanation

Use the Modules view to look at the load modules and programs.

| | |
|---|---|
| **EQA4006I** | **The current location is *CU-name*:>*Statement-id*.** |

## Explanation

Shows the place where the program was interrupted.

| | |
|---|---|
| **EQA4007I** | **You are executing commands in the *Bkp-Id* breakpoint.** |

## Explanation

Shows the bearings in an interrupted program.

**EQA4008I** **You are executing commands from the run-time command-list.**

## Explanation

Shows the bearings in an interrupted program.

**EQA4009I** **You were prompted because *promptCode* ended.**

## Explanation

Shows the bearings in an interrupted program.

**EQA4010I** **The program is currently entering block *Block-name*.**

## Explanation

Shows the bearings in an interrupted program.

**EQA4011I** **The program is currently exiting block *Block-name*.**

## Explanation

Shows the bearings in an interrupted program.

**EQA4012I** **The program is currently executing prolog code for *Block-name*.**

## Explanation

Shows the bearings in an interrupted program.

**EQA4013I** **You are executing commands within a *CeeCPliTest-name* or equivalent function.**

## Explanation

Shows the bearings in an interrupted program.

**EQA4014E** **There was insufficient storage to satisfy the request from *ModuleName* for X'*Length*' bytes of storage. z/OS Debugger might abend or your results might be unpredictable. Try increasing your region size.**

## Explanation

This message is issued when there is not enough storage available to satisfy the request.

## Programmer response

Increase the storage limit (for example, the region size).

**EQA4015I** **Load module information could not be retrieved for module *ModuleName* because the loading service (provider) that loaded the module loaded it from a system managed library (e.g. LPA). The load module cannot be debugged.**

## Explanation

z/OS Debugger uses Binder APIs to retrieve necessary information for the debugging of a load module. When the load module is loaded from a system managed library, the Binder APIs cannot retrieve this necessary information. Hence, z/OS Debugger cannot debug the module.

## Programmer response

Contact your System Programmer and have them place the module in a non-system managed library.

**EQA4017I** **Load module *ModuleName* was loaded from LLA. The load module information was processed from a data set found in file *DDName*.**

## Explanation

z/OS Debugger uses binder APIs to retrieve necessary information for the debugging of a load module. When the load module is loaded from a system managed library the binder APIs cannot retrieve this necessary information. In this case, a module by the same name and with the same length was found in the data set(s) allocated to the specified *DDName*. That module was used by the binder APIs.

## Programmer response

None.

**EQA4018E** **Load module *ModuleName* was loaded from LLA. A load module by the same name was found in a data set found in file *DDName*. However, the lengths of the load modules did not match.**

## Explanation

z/OS Debugger uses binder APIs to retrieve necessary information for the debugging of a load module. When the load module is loaded from a system managed

library the binder APIs cannot retrieve this necessary information. In this case, a module by the same name and but with a different length was found in the data set(s) allocated to the specified *DDName*. Because the lengths do not match, that module cannot be used by the binder APIs and, therefore, the load module cannot be debugged.

### Programmer response

None.

**EQA4019E** **When the binder APIs extracted the debug information for load module *ModuleName* the length of the load module in memory did not match the length extracted by the binder APIs.**

### Explanation

z/OS Debugger uses binder APIs to retrieve necessary information for the debugging of a load module. The load module length returned by the binder APIs does not match the length of the load module in memory.

### Programmer response

None.

**EQA4020E** **Load module information could not be retrieved for module *ModuleName* because the loading service (provider) that loaded the module loaded it from LLA and no matching copy was found in either EQALOAD or STEPLIB. The load module cannot be debugged.**

### Explanation

z/OS Debugger uses binder APIs to retrieve necessary information for the debugging of a load module. When the load module is loaded from LLA, the binder APIs cannot retrieve this necessary information. z/OS Debugger then looks in file EQALOAD or STEPLIB to attempt to find a load module with the same name and length in order to obtain this information. No such copy was found. Hence, z/OS Debugger cannot debug the module.

### Programmer response

Allocate either EQALOAD or STEPLIB to a data set containing a copy of the specified program that matches the copy in LLA or do an LLA refresh to ensure that the system is using the latest copy.

**EQA4021I** **The setting of ASSEMBLER STEPOVER is now &&&&.**

### Explanation

This message is issued by the remote interface in response to the SET ASSEMBLER STEPOVER command.

### Programmer response

None.

**EQA4022I** **Console: File not suitable for I/O interception.**

### Explanation

This message is issued by the remote interface in response to SET INTERCEPT being turned on and for COBOL DISPLAY UPON CONSOLE output.

### Programmer response

None.

**EQA4023I** **Console: I/O interception not supported.**

### Explanation

This message is issued by the remote interface in response to SET INTERCEPT being turned on and for COBOL DISPLAY UPON CONSOLE output.

### Programmer response

None.

**EQA4024I** **Console: *UponConsoleOP***

### Explanation

This message is issued by the remote interface in response to SET INTERCEPT being turned on and for COBOL DISPLAY UPON CONSOLE output.

### Programmer response

None.

**EQA4025I** **&&&& &&&& &&&& &&&&.**

### Explanation

This message contains the output from the DESCRIBE LOADMODS command.

**EQA4026E** **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 an active load module.

**EQA4027E** **The Debug File creation date does not match the object for &&&&. The Debug File cannot 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.

**EQA4028I** **Default User Preferences file exists: &&&&**

## Explanation

The default user preferences file is opened successfully.

**EQA4029I** **Default User Commands file exists: &&&&**

## Explanation

The default user commands file is opened successfully.

**EQA4030I** **Variable too large for automonitor.**

## Explanation

The variable is too large to view. Right-click and select *Evaluate large expression* to display.

**EQA4031E** **DYNDEBUG cannot be SET OFF when running without the Language Environment runtime.**

## Explanation

Dynamic Debug cannot be turned off when running without the Language Environment runtime.

**EQA4032I** **Its linkage is Pre-Language Environment OS/Standard**

## Explanation

This compile unit uses OS/Standard linkage and was generated by a pre-Language Environment compiler.

**EQA4033I** **Its linkage is Language Environment FastLink**

## Explanation

This compile unit uses Language Environment FastLink linkage.

**EQA4034I** **Its linkage is Language Environment OS/Standard**

## Explanation

This compile unit uses Language Environment OS/Standard Linkage.

**EQA4035I** **Its linkage is OS/Standard**

## Explanation

This non-Language Environment compile unit uses OS/Standard linkage.

**EQA4036I** **Its linkage is Language Environment XPLINK *precision-subtype***

## Explanation

This compile unit uses Language Environment 31 or 64-bit XPLink linkage. The -1, -2, etc. indicates the type of Language Environment header found.

**EQA4037I** **z/OS Debugger is starting debug of *Program* in *environment* for user *userid*.**

## Explanation

This message is issued during z/OS Debugger initialization if requested by the EQAOPTS STARTSTOPMSG command.

**EQA4038I** **z/OS Debugger is ending debug of *Program* in *environment* for user *userid*.**

## Explanation

This message is issued during z/OS Debugger termination if requested by the EQAOPTS STARTSTOPMSG command.

**EQA4039I** **IMS system ID: *IMS_ID*, Region ID: *IMS_RID*, Transaction ID: *tran_ID***

## Explanation

This message is issued during z/OS Debugger initialization and termination if requested by the EQAOPTS STARTSTOPMSG command.

**EQA4040W** **The "Stop When Date Fields are Accessed" breakpoint is set, but**

**only COBOL compile units support these types of breakpoints.**

## Explanation

The "Stop When Date Fields are Accessed" breakpoints are also known as DATE breakpoints, which are supported only in COBOL compile units compiled with the DATEPROC compiler option. The application will not stop if a date field in any of the following types of compile units is accessed:

- A COBOL compile unit compiled without the DATEPROC compiler option.
- A compile unit that is not a COBOL compile unit.

## Programmer response

If the application being debugged does not contain a COBOL compile unit compiled with the DATEPROC compiler option, remove this breakpoint.

**EQA4041I** **The setting of LIST BY SUBSCRIPT is on.**

**EQA4042I** **The setting of LIST BY SUBSCRIPT is off.**

**EQA4043I** **The setting of LIST BY SUBSCRIPT cannot be changed while monitoring expressions. Remove the monitored expressions from Monitor and Variables Views in order to change the setting of LIST BY SUBSCRIPT.**

## Explanation

z/OS Debugger is already monitoring expressions and local variables using the current LIST BY SUBSCRIPT settings. You cannot change the settings at this point.

**EQA4050E** **The expression or variable &&&& is not valid, undefined, or not correctly qualified.**

## Explanation

z/OS Debugger could not obtain an address for this expression or variable. The expression or variable might be out of scope or the syntax might not be valid.

## Programmer response

Check that the expression or variable is typed in correctly. Check that the variable or variables used are within scope of the current compile unit. Check that the expression or variables are supported in the current programming language.

**EQA4051E** **This command is only valid in a CICS Environment**

## Explanation

The command is only valid in a CICS Environment.

## Programmer response

None.

**EQA4052I** **IGNORELINK mode is now active.**

**EQA4053I** **IGNORELINK mode is no longer active.**

**EQA4054I** **The setting of DEFAULT DBG is:**

## Explanation

This message header is for QUERY DEFAULT DBG.

**EQA4055I** **The setting of DEFAULT MDBG is:**

## Explanation

This message header is for QUERY DEFAULT MDBG.

**EQA4056E** **You cannot change the location of the source for this program because the .mdbg file with the debug information is not available or the source is contained in the .mdbg file.**

## Explanation

The source for this program might be part of the .mdbg file.

**EQA4057E** **The .mdbg file with the debug data for program &&&& could not be found.**

## Explanation

Make sure there is a .mdbg file available.

**EQA4058E** **A .mdbg file was found, but the debug data does not match program &&&&.**

## Explanation

Make sure the proper .mdbg file is available.

**EQA4059E** **The .dbg file with the debug data for program &&&& could not be found.**

## Explanation

Make sure there is a .dbg file available.

**EQA4060E** **A .dbg file was found, but the debug data does not match program &&&&.**

## Explanation

Make sure the proper .dbg file is available.

**EQA4061I** **The following errors were detected processing the dynamic EQAOPTS specifications.**

## Explanation

Errors were detected processing the dynamically specified EQAOPTS commands. Each command containing an error will be listed, followed by the applicable error message.

**EQA4062I** **&&&&**

## Explanation

An EqaOpts command.

**EQA4063I** **The following EQAOPTS were specified dynamically.**

## Explanation

The EQAOPTS specifications shown following this message were specified dynamically.

**EQA4064I** **The following EQAOPTS were obtained from the EQAOPTS load module.**

## Explanation

The EQAOPTS specifications shown following this message were obtained from the EQAOPTS load module.

**EQA4065E** **A close-quote in operand &&&& was not followed by a blank or a comma.**

## Explanation

An close-quote indicates the end of an operand. Therefore, it must be followed by a comma to indicate that another operand is present or by a blank to indicate that this is the last operand.

**EQA4066E** **A continuation was not found.**

## Explanation

The last line contained a non-blank character in column 72 indicating a continued statement but no continuation line was found.

**EQA4067E** **The last operand ended in a comma but column 72 is blank.**

## Explanation

The last operand on the current line ended with a comma but a non-blank was not present in column 72 to indicate a continuation.

**EQA4068E** **The specified numeric value of operand &&&& is not within the expected range. It is either too small or too large.**

## Explanation

The specified number is outside of the range allowed for the indicated operand.

**EQA4069E** **Not enough operands were specified.**

## Explanation

Some of the operands required for this EQAOPTS function were not specified.

**EQA4070E** **Too many operand were specified.**

## Explanation

More operands were specified than are allowed for this EQAOPTS function.

**EQA4071E** **EQAXOPT opcode was not found.**

## Explanation

All EQAOPTS functions must contain the EQAXOPT opcode.

**EQA4072E** **A value is not allowed for operand &&&&.**

## Explanation

The indicated operand keyword was followed by an '=' and an operand. However, this operand does not allow the specification of a value.

**EQA4073E** **No value was specified for operand &&&&.**

## Explanation

The indicated operand requires a value to be specified but none was found.

**EQA4074E** **Internal error (invalid operand type) in operand &&&&.**

## Explanation

An unexpected situation was encountered. Contact z/OS Debugger support.

**EQA4075E** **A non-decimal digit was found in the numeric value for operand &&&&.**

## Explanation

The indicated operand expects a numeric value. However, a character other than a decimal digit was found in the value.

**EQA4076E** **The first operand is not a valid EQAOPTS function.**

## Explanation

The first operand must be a valid EQAOPTS function name.

**EQA4077E** **EQAXOPT END is not the last command.**

## Explanation

The 'EQAXOPT END' command must be the last command entered.

**EQA4078E** **Incorrect use of quotes in operand &&&&.**

## Explanation

Quotes were incorrectly used in the indicated operand. If you want to include a quote or an ampersand within a quoted string, a pair of quotes or ampersands must be specified.

**EQA4079E** **Column 1 to 15 of a continuation line are non-blank.**

## Explanation

Continuation lines must begin in column 16. Make sure that you did not begin the continuation line before column 15.

**EQA4080E** **Operand &&&& is too long.**

## Explanation

The indicated operand is longer than the maximum length allowed.

**EQA4081E** **Internal error (more than one value for a function) in operand &&&&.**

## Explanation

An unexpected situation was encountered. Contact z/OS Debugger support.

**EQA4082E** **Unmatched parenthesis were found in operand &&&&.**

## Explanation

A left parenthesis was found without a corresponding right parenthesis or vice-versa.

**EQA4083E** **An asterisk was specified in operand &&&& but is not allowed in that operand.**

## Explanation

An asterisk was detected in an operand that does not allow an asterisk to be specified.

**EQA4084E** **An unrecognized keyword was specified in operand &&&&.**

## Explanation

An unrecognized keyword was specified for the indicated operand.

**EQA4085E** **Return code &&&& and reason code &&&& were encounted while obtaining the dynamic EQAOPTS specifications. The dynamic EQAOPTS were not processed.**

## Explanation

The return code value indicates the function being processed as follows: 1 = Allocating memory, 2 = Obtaining data set attributes, 3 = Opening data set, 4 = Reading data set, and 5 = Closing data set.
The specified reason code indicates the specific error encountered by this function.

**EQA4086E** **This EQAXOPT command must be entered in the EQAOPTS load module.**

## Explanation

The indicated command cannot be specified dynamically.

**EQA4087E** **The EQAOPTS data set, '*DSName*', cannot be opened or cannot be read.**

## Explanation

An error occurred trying to OPEN or READ the EQAOPTS data set.

**EQA4088W** **An ATTACH has been detected which uses the GSPV parameter to give ownership of subpool *SPNumber* to the new task. Use the NONLESP option to direct z/OS Debugger to use a different subpool.**

## Explanation

z/OS Debugger uses the indicated subpool to allocate its data areas and control blocks. The ATTACH GSPV parameter gives the ownership of the subpool to the new task, and the entire subpool is freed when the new task is completed. This causes unpredictable behavior in z/OS Debugger, including abends.

## Programmer response

If you need to use z/OS Debugger in this process after the subtask is completed, use the NONLESP runtime parameter to instruct z/OS Debugger to use a different subpool for its storage.

**EQA4089E** **The COBOL program *CUName* is link-edited with an unsupported level of the CSECT CEEBETBL.**

## Explanation

A COBOL program compiled with Version 5 or higher is link-edited with an old copy of CEEBETBL.

## Programmer response

Link-edit the load module with a current version of CEEBETBL from the Language Environment SCEELKED library.

**EQA4090E** **The COBOL program *CUName* is compiled with COBOL Version 5 or higher and the debug data in the load module does not match the currently executing program.**

## Explanation

There is a mismatch between the currently executing COBOL program and the debug data in the load module. This can occur for a COBOL program compiled with COBOL V5 with the TEST compiler option or for a COBOL program compiled with COBOL V6 or higher with the TEST(NOSEPARATE) compiler option.

## Programmer response

If you are on CICS, do a CEMT SET PROGRAM(xxxx) NEWCOPY before you run the program. For more information, see the "CEMT SET PROGRAM" topic in the CICS documentation and the "Running NEWCOPY on programs by using DTNP transaction" topic in *IBM z/OS Debugger User's Guide*.

**EQA4091W** **The version of the z/OS Debugger SVC installed on the system does not support STEP processing for GO programs. The installed version is *installed_version*. The required version is *required_version*. STEP will be disabled for this debug session.**

## Explanation

Version 20 or higher of the z/OS Debugger SVC must be active to allow STEP processing for GO programs.

## Programmer response

Contact your systems programmer to ensure the correct SVC version is activated.

**EQA4092E** **The COBOL program *CUName* is in a load module linkedited with the binder PAGE statement.**

## Explanation

The debug information for the COBOL program cannot be located in the load module because the load module was link-edited with the binder PAGE statement. The problem can occur for a COBOL program compiled with COBOL V5 with the TEST compiler option or for a COBOL program compiled with COBOL V6 or higher with the TEST(NOSEPARATE) compiler option.

## Programmer response

Link-edit the load module without the binder PAGE statement or compile the program with the compiler option TEST(SEPERATE).

**EQA4093E** **A valid DBRM file for *CUName* is not found.**

## Explanation

A valid DBRM file for the compile unit is not found.

## Programmer response

Specify the name of the data set that contains the DBRM file in either the EQA_DBG_DBRM environment variable or the EQADBRM DD name.

**EQA4094I**

## Explanation

This compile unit was optimized by ABO.

After this COBOL program was compiled, the module was optimized by Automatic Binary Optimizer for z/OS.

For programs complied with Enterprise COBOL for z/OS Version 5 or later, you can debug them as the programs that are not optimized by ABO.

For programs complied with Enterprise COBOL for z/OS Version 4 or earlier, you can use the LangX COBOL support in z/OS Debugger to debug (with restrictions) a load module or program object generated by the ABO. For more information, see "Debugging a program processed by the Automatic Binary Optimizer for z/OS" in IBM z/OS Debugger User's Guide.

**EQA4700E** **A parse error was detected by the z/OS XML System Services parser: Return Code=*rc*, Reason Code=X'*reason*', Offset=X'*buffoffset*'. See the XML System Services User's Guide and Reference for a description of this error.**

## Explanation

The z/OS XML System Services parser returned the indicate return code and reason code. See the z/OS XML System Services documentation for a complete description of the associated error. This message may be accompanied by message EQA4701I and/or EQA4702I and EQA4703I.

**EQA4701I** ***error_description***

## Explanation

This message may be issued following message EQA4700E. It contains a short description of the error associated with the Return Code and Reason Code included in that message.

**EQA4702I** **Context=|*context*|**

## Explanation

This message may be issued following message EQA4700E and will be followed by message EQA4703I. It contains a few characters of the XML source surrounding the point at which the error was detected by the XML parser.

**EQA4703I** **|*cursor*|**

## Explanation

This message follows message EQA4702I and uses an asterisk to indicate the column in that message at which the error was detected by the XML parser.

**EQA4704I** **XML(*EBCDICorASCII*) assumed.**

## Explanation

The XML keyword was specified without the EBCDIC or ASCII suboperand. In this case, all characters with a value less than X'80', except for X'40', are assumed to be ASCII characters. All characters with a value greater than or equal to X'80' are assumed to be EBCDIC characters. If the specified area contains more ASCII characters than EBCDIC characters, ASCII is defaulted. Otherwise, EBCDIC is defaulted.

## Programmer response

If the correct encoding was not defaulted, specify the EBCDIC or ASCII keyword suboperand of the XML keyword.

**EQA4705E** **z/OS XML System Services are not installed. The command cannot be processed.**

## Explanation

z/OS Debugger XML processing requires the z/OS XML System Services that are only available when running z/OS V1R8 or later or on z/OS V1R7 with the proper APAR installed. The command cannot be processed.

**EQA4836E** **You can not step back into a parent enclave when PLAYBACK BACKWARD is in effect.**

## Explanation

PLAYBACK BACKWARD into parent enclave is not allowed for programs compiled with Enterprise COBOL for z/OS Version 5.

**EQA4837I** **The following LABELS are known in *CU_Name*.**

## Explanation

This message is for informational purposes only.

**EQA4838I** **There are no LABELS in the current program *CU_Name*.**

## Explanation

There are no LABELS in the current program.

**EQA4839E** **This command is not supported for program *CU_Name*.**

## Explanation

Programming language that the program is written in is not supported.

**EQA4840W** **The GLOBAL LABEL breakpoint is already established.**

## Explanation

The AT GLOBAL LABEL breakpoint is already established.

**EQA4841W** **The GLOBAL LABEL breakpoint has not been established.**

## Explanation

The AT GLOBAL LABEL breakpoint is not established.

**EQA4842I** **The GLOBAL LABEL breakpoint is established.**

## Explanation

The AT GLOBAL LABEL breakpoint is active.

**EQA4843E** **IDISNAP could not be loaded. Verify Fault Analyzer is available.**

## Explanation

IDISNAP is a part of the product IBM Fault Analyzer.

**EQA4844I** **IBM Fault Analyzer History File Created.**

## Explanation

An IBM Fault Analyzer History File is created.

**EQA4845I** **Program was stopped due to a LABEL breakpoint.**

## Explanation

The program was stopped due to a label breakpoint.

**EQA4846I** **The following label breakpoints exist for program *ProgramName*.**

## Explanation

Label breakpoints exist in the program.

**EQA4847I** ***LabelName***

## Explanation

Label breakpoint.

**EQA4848W** **A Label breakpoint does not exist for label *LabelName*.**

## Explanation

A Label breakpoint does not exist for this label.

**EQA4849E** **The label variable is not in scope or does not exist in current compilation unit.**

## Explanation

The label variable is not in scope or does not exist in current compilation unit.

**EQA4850W** **Label breakpoints are not established.**

## Explanation

Label breakpoints are not defined *LabelName*.

**EQA4851I** **A Label breakpoint already exists for label *LabelName*.**

## Explanation

A Label breakpoint already exists for this label.

**EQA9870E** **IMS with ID *imsid* is inactive or not set up for transaction isolation.**

## Explanation

A program was called to generate resource definitions for the IMS Transaction Isolation facility. However, the program could not find an active IMS control region with ID *imsid* configured for transaction isolation.

## Programmer response

Verify that step 4 in "Scenario F: Enabling the Transaction Isolation Facility" in the *IBM z/OS*

*Debugger Customization Guide* has been completed with the IMS control region configured for transaction isolation running, and that the correct IMS ID is provided to the program.

**EQA9871E** **IBM z/OS Debugger - No product registration that supports running with Rational Integration Tester was found.**
**The enclave will be terminated.**

## Explanation

Running the debugger with Rational Integration Tester is supported only when an enabled product registration for one of the following products is found:

- Application Delivery Foundation for z/OS
- IBM Developer for z/OS Enterprise Edition
- IBM Debug for z/OS

However, none was found.

## User response

Ask the installer to check the product registration. If z/OS Debugger is not licensed for any of these products, do not run the debugger with Rational Integration Tester.

## Programmer response

If IBM z/OS Debugger is licensed to run on this machine as Application Delivery Foundation for z/OS, IBM Developer for z/OS Enterprise Edition, or IBM Debug for z/OS, register and enable the product registration.

For more information about how to register, see *IBM z/OS Debugger Customization Guide*.

If z/OS Debugger is not licensed for one of these products, do not run the debugger with Rational Integration Tester.

**EQA9872E** **CSL message ID: *message-id***

## Explanation

IMS Transaction Isolation Facility received an error response to a type-2 IMS command. The message identifier *message-id* is associated with the error. See message EQA9873E for the corresponding message text.

## System action

None.

## User response

Consult the *IMS Messages and Codes, Volume 2: Non-DFS Messages* book for a description of the error message.

**EQA9873E** **CSL message text: *message-text***

## Explanation

IMS Transaction Isolation Facility received an error response to a type-2 IMS command. The message text *message-text* is associated with the error. See message EQA9872E for the corresponding message identifier.

## System action

None.

## User response

See User Response in message EQA9872E.

**EQA9874E** **Debug Tool compatibility mode does not support source level debug of 64-bit PL/I programs.**
**Processing will continue as if the PL/I program had no debug data.**

## Explanation

The debugger does not support debugging PL/I programs that are compiled for 64-bit (AMODE(64)) in Debug Tool compatibility mode.

## System action

Processing will continue as if the PL/I program had no debug data.

## Programmer response

Use standard mode to debug 64-bit PL/I programs. Specify DBM% or DIRECT& in the TEST runtime option.

**EQA9875E** **z/OS Debugger does not support line, batch, or full-screen mode for 64-bit COBOL, PL/I, and C/C++ programs.**
**Processing will continue as if a QUIT DEBUG command had been entered.**

## Explanation

The debugger does not support debugging COBOL, PL/I, or C/C++ programs that are compiled for 64-

bit (AMODE(64)) in line, batch, or full-screen (3270) mode.

## System action

Processing will continue as if a QUIT DEBUG command had been entered.

## Programmer response

Use remote debug mode to debug this type of program. Specify DBMDT% or TCPIP& in the TEST runtime option to debug 64-bit COBOL and C/C++ programs in Debug Tool compatibility mode, or DBM% or DIRECT& to debug 64-bit PL/I and C/C++ programs in standard mode.

**EQA9876W** **The saved Binder Info should be in short form. However, a long form was detected.**

**Explanation:**
IBM z/OS Debugger detected a wrong form when processing information from the z/OS Binder. z/OS Debugger might not be able to provide correct module information. For example, the CU length from DESCRIBE CUS might not be accurate.

**Programmer response:**
If the problem persists, contact IBM Software Support.

**EQA9878E** **IBM z/OS Debugger - No product registration that supports an EQA_STARTUP_KEY value of DCC was found.**
**The enclave will be terminated.**

## Explanation

The EQA_STARTUP_KEY environment variable with a value of DCC is supported only when an enabled product registration for one of the following products is found:

- Application Delivery Foundation for z/OS
- IBM Developer for z/OS Enterprise Edition
- IBM Debug for z/OS
- IBM Developer for z/OS

**User response:**
Ask the installer to check the product registration. If z/OS Debugger is licensed for IBM Wazi Developer for Red Hat CodeReady Workspaces or IBM Z and Cloud Modernization Stack, do not run with an EQA_STARTUP_KEY value of DCC.

## Programmer response

If IBM z/OS Debugger is licensed to run on this machine as Application Delivery Foundation for z/OS, IBM Developer for z/OS Enterprise Edition, IBM Debug for z/OS or IBM Developer for z/OS, register and enable the product registration.

For more information about how to register, see *IBM z/OS Debugger Customization Guide*.

If z/OS Debugger is not licensed for any of these products, do not use an EQA_STARTUP_KEY value of DCC.

**EQA9879E** **IBM z/OS Debugger - No product registration that supports full-screen mode was found.**
**The enclave will be terminated.**

## Explanation

To use full-screen mode on a 3270, z/OS Debugger must find an enabled product registration for one of the following products:

- Application Delivery Foundation for z/OS
- IBM Developer for z/OS Enterprise Edition
- IBM Debug for z/OS

However, none was found.

**User response:**
Ask the installer to check the product registration. Do not use full-screen mode on a 3270 if z/OS Debugger is not licensed for Application Delivery Foundation for z/OS, IBM Developer for z/OS Enterprise Edition or IBM Debug for z/OS.

## Programmer response

If IBM z/OS Debugger is licensed to run on this machine as Application Delivery Foundation for z/OS, IBM Developer for z/OS Enterprise Edition or IBM Debug for z/OS, register and enable the product registration.

For more information about how to register, see *IBM z/OS Debugger Customization Guide*.

If z/OS Debugger is not licensed for one of these products, full-screen mode on a 3270 in not available.

**EQA9880E** **IBM z/OS Debugger - No product registration that supports an EQA_STARTUP_KEY value of LD was found.**
**The enclave will be terminated.**

## Explanation

The EQA_STARTUP_KEY environment variable with a value of LD is supported only when an enabled product registration for one of the following products is found:

- Application Delivery Foundation for z/OS
- IBM Developer for z/OS Enterprise Edition
- IBM Debug for z/OS

- IBM Developer for z/OS

However, none was found.

**User response:**
Ask the installer to check the product registration. If z/OS Debugger is licensed for IBM Wazi Developer for Red Hat CodeReady Workspaces or IBM Z and Cloud Modernization Stack, do not run with an EQA_STARTUP_KEY value of LD.

## Programmer response

If IBM z/OS Debugger is licensed to run on this machine as Application Delivery Foundation for z/OS, IBM Developer for z/OS Enterprise Edition, IBM Debug for z/OS, IBM Developer for z/OS, register and enable the product registration.

For more information about how to register, see *IBM z/OS Debugger Customization Guide*.

If z/OS Debugger is not licensed for one of these products, do not run with an EQA_STARTUP_KEY value of LD.

**EQA9882S** **DEREGISTER *transaction_name IMSId ** is not supported. Use DEREGISTER ALL *IMSId ** instead.**

## Explanation

DEREGISTER *transaction_name IMSId ** is not a supported syntax for batch EQANIPSB. The command is ignored.

**Programmer response:**
Use DEREGISTER ALL *IMSId ** instead.

**EQA9884E** **DTCN Cache is corrupted**

## Explanation

A z/OS Debugger control block has been corrupted.

**Programmer response:**
z/OS Debugger will attempt to recover the control block. If the problem persists, investigate any application storage violations. If the problem still persists after all application storage violations have been resolved, contact IBM Software Support.

**EQA9885I** **EQA10XSC terminating - *abend code***

## Explanation

The debugger is terminating via abend code *abend code*.

## Programmer response

No action is required.

**EQA9886E** **IBM z/OS Debugger - the workstation could not be reached to check license. The enclave will be terminated.**

## Explanation

z/OS Debugger called IFAEDREG to check for an enabled product registration for Application Delivery Foundation for z/OS, IBM Developer for z/OS Enterprise Edition, or IBM Debug for z/OS. No enabled product registration was found. The TEST parm specified that remote debug mode was to be used, so the debugger attempted to reach the workstation to determine whether it was licensed (typically as IBM Developer for z/OS, IBM Wazi Developer for Red Hat CodeReady Workspaces, or IBM Z and Cloud Modernization Stack). However, the debugger could not reach the workstation.

**Programmer response:**
Correct the communications problem with reaching the workstation.

**EQA9887E** **IBM z/OS Debugger - workstation failed the license check. The enclave will be terminated.**

## Explanation

z/OS Debugger called IFAEDREG to check for an enabled product registration for Application Delivery Foundation for z/OS, IBM Developer for z/OS Enterprise Edition, or IBM Debug for z/OS. No enabled product registration was found. The TEST parm specified that remote debug mode was to be used, so the debugger asked the workstation whether it was licensed (typically as IBM Developer for z/OS, IBM Wazi Developer for Red Hat CodeReady Workspaces, or IBM Z and Cloud Modernization Stack). Either the workstation code was not licensed or was too old to respond with license information (older than Rational Developer for System z® 9.5.1).

**Programmer response:**
License the workstation or use a newer workstation code that can respond with license information.

**EQA9888I** **EQANCDBG can not continue. User: *CICS_Userid* is not authorised to issue EXEC CICS INQUIRE.**

## Explanation

User *CICS_Userid* is not authorized to issue EXEC CICS INQUIRE.

**Programmer response:**

Authorize User *CICS_Userid* to be able to issue EXEC CICS INQUIRE.

**EQA9889I z/OS Debugger unable to initialize.**

## Explanation

z/OS Debugger was not able to load the operating system services module.

**Programmer response:**
Review your installation to ensure that the z/OS Debugger SEQAMOD library is in the search path for the current task.

**EQA9890E EQA00OHT Invalid *type* address**

## Explanation

Standard mode is being used. An invalid *type* address was selected.

**Programmer response:**
If the problem persists, contact IBM Software Support.

**EQA9891S Installed SVC version is not sufficient to support this function. Aborting EQANIPSB.**

## Explanation

During initialization, z/OS Debugger detected a downlevel z/OS Debugger SVC version. The IMS Transaction Isolation Facility is disabled.

**Programmer response:**
Have your installer install the correct z/OS Debugger SVCs.

**EQA9892S Installed SVC version is not sufficient to support this function. Aborting EQANBSWT.**

## Explanation

During initialization, z/OS Debugger detected a downlevel z/OS Debugger SVC version. The IMS Transaction Isolation Facility is disabled.

**Programmer response:**
Have your installer install the correct z/OS Debugger SVCs.

**EQA9893I Userid *MVS_UserId* received an IOERR response when trying to access resource *datasetname*. Possible NOTAUTH condition.**

## Explanation

z/OS Debugger received an IOERR response from CICS while trying to access a data set.

**Programmer response:**
Investigate the cause of the IOERR response.

**EQA9894S EXTNAME *string* HAS AN INVALID LENGTH *length***

## Explanation

The EXTNAME tag in the code coverage Options data set has a string value that is longer than 8 characters.

**Programmer response:**
Correct the value of the EXTNAME tag so that it has 8 or less characters.

**EQA9895I Unsupported option specified**

## Explanation

An unsupported option was specified in the parm string for EQAYSESM, the Terminal Interface Manager started task. For a list of the options for EQAYSESM, see the "Enabling debugging in full-screen mode using the Terminal Interface Manager" in the *IBM z/OS Debugger Customization Guide*.

**Programmer response:**
Correct or remove the unsupported option.

**EQA9896I Z/OS DEBUGGER TIM USERS USER ID FLAGS TERMINAL JOBNAME**

## Explanation

This is header of a list of Terminal Interface Manager users shown on the z/OS console.

## Programmer response

No action is required.

**EQA9897I EQADCDEL: DTCN profile owned by terminal *terminal action***

## Explanation

The Profile Clean Up mechanism of DTCN has processed a profile.

## Programmer response

No action is required.

**EQA9898E nonLECAA 0 and LE CAA 0 ABENDING**

## Explanation

z/OS Debugger is unable to locate vital control blocks.

**Programmer response:**

Save the system dump that is produced, and send it to IBM Software Support.

**EQA9899E nonLECAA 0 and LE CAA 0 ABENDING**

## Explanation

z/OS Debugger is unable to locate vital control blocks.

**Programmer response:**
Save the system dump that is produced, and send it to IBM Software Support.

**EQA9900S EQADBFIL Could not allocate '*xxxxxxxx*' X bytes of storage. Process terminated.**

## Explanation

EQADBFIL was processing a SYSDEBUG file and could not get enough storage to complete.

## Response

Specify a larger region size.

**EQA9901E Data set '*dsname*' member not specfied; process is aborted.**

## Explanation

EQAOPTS command CCPROGSELECTDSN specified a PDS or PDSE, but no member name was specified.

## Response

Specify a member name.

**EQA9902E Data set '*dsname*' does not exist; process is aborted.**

## Explanation

z/OS Debugger detected an allocation error for the specified data set. If no data set is specified in the message, it was not entered in EQAOPTS command CCPROGSELECTDSN.

## Response

Check the spelling of the specified data set or specify it by EQAOPTS command CCPROGSELECTDSN.

**EQA9903E Data set '*dsname*' contains invalid data; process is aborted.**

## Explanation

z/OS Debugger detected invalid XML data in the specified data set.

## Response

Correct the XML error in the specified data set and resubmit the job.

**EQA9904E Error reading code coverage data set '*dsname*'; process is aborted.**

## Explanation

z/OS Debugger encountered a read error in the specified data set.

## Response

Correct the data set and resubmit the job.

**EQA9905I Data set '*dsname*' contains an unsupported XML tag.**

## Explanation

z/OS Debugger encountered an unsupported XML tag in the specified data set.

## Response

Processing continues. Correct the tag and resubmit the job if this is unexpected.

**EQA9906I Data set '*dsname*' is held by another process; waiting for it to be available.**

## Explanation

z/OS Debugger can not allocate the specified data set as MOD, because it is in use by another process.

## Response

z/OS Debugger waits for the specified data set to become available. Cancel the job if the waiting is not desired.

**EQA9907S Cannot open XML output data set '*dsname*'; process is aborted.**

## Explanation

z/OS Debugger encountered an error while attempting to open the specified data set.

## Response

Verify that the data set exists and that it is spelled correctly in the EQAOPTS command CCOUTPUTDSN and resubmit the job.

**EQA9908S** **Partitioned data set specified for the XML output; process is aborted.**

## Explanation

A partitioned data set was specified for the z/OS Debugger output XML data set. This is not supported.

## Response

Specify a sequential data set for the z/OS Debugger XML output and resubmit the job.

**EQA9909I** **XML output data set '*dsname*' is not cataloged; will create new data set with default attributes.**

## Explanation

The specified output data set cannot be found. The data set is created using the default attributes.

## Response

Specify the EQAOPTS command CCOUTPUTDSNALLOC if you want to control the allocation of the XML output data set.

**EQA9910I** **DTST User *Userid* has changed storage at address, *address* from *oldvalue* to *newvalue*.**

## Explanation

A CICS user has used the DTST transaction to change storage.

## Response

None.

**EQA9911W** **VTAM session terminated due to inactivity. Elapsed time between user inputs exceeded *hh*:*mm*:*ss*:*tt*.**

## Explanation

A z/OS Debugger session using full-screen mode on a dedicated terminal or full-screen mode using Terminal Interface Manager has been idle for longer than the elapsed time specified in the EQAXOPT SESSIONTIMEOUT command. The time out parameter from the EQAXOPT SESSIONTIMEOUT command is presented in the message, in the form *hh*:*mm*:*ss*:*tt*, where *hh* is the number of hours, *mm* is the number of minutes, *ss* is the number of seconds, and *tt* is the number of hundredths of seconds. This message will be followed by message EQA9931W if the **QUITDEBUG** option of SESSIONTIMEOUT was selected; it will be followed by a CEE2F1 Language Environment condition if the **QUIT** option was chosen.

**EQA9912W** **VTAM% specified and user is already in a z/OS Debugger session.**

## Explanation

The TEST parameter specifies that a full-screen mode using Terminal Interface Manager session is to be started, but the specified user is already in a debugging session using the Terminal Interface Manager. This message will be followed by EQA9931W, which indicates that z/OS Debugger will quit processing events and the application will continue to run, as though QUIT DEBUG had been specified.

**EQA9913E** **EQADTCN2 invalid recordsize. All regions which share the EQADTCN2 queue must be at the same version of DT.**

## Explanation

The DTCN TS-Queue Repository EQADTCN2 has items which have an incorrect record size for this version of z/OS Debugger. This could be because you are sharing the repository among regions which are using different versions of z/OS Debugger.

## Response

Ensure that all regions which share the EQADTCN2 repository are using the same version of z/OS Debugger.

**EQA9914I** **SAVE of invalid MONITOR command suppressed.**

## Explanation

An invalid MONITOR command was detected during SAVE or RESTORE of MONITORS. The invalid command is ignored.

**EQA9914I** **RESTORE of invalid MONITOR command suppressed.**

## Explanation

An invalid MONITOR command was detected during SAVE or RESTORE of MONITORS. The invalid command is ignored.

**EQA9915E** **EQADPFMB invalid recordsize. File must be rebuilt. See SEQASAMP(EQAWCRVS).**

## Explanation

The DTCN VSAM Repository data set EQADPFMB has an incorrect record size for this version of z/OS Debugger. This could be because it was created for an older version of z/OS Debugger than the one that is in use.

## Response

Allocate and use a new EQADPFMB data set using the JCL in member EQAWCRVS of the SEQASAMP data set for the version of z/OS Debugger that is in use. Also consider using the JCL in member EQADPCNV of the SEQASAMP data set to convert EQADPFMB profiles from previous versions of z/OS Debugger to the new format.

**EQA9916I** **RPL error closing session RPL error codes :*rpl-error-codes*: RPL sense codes :*rpl-sense-codes*: RPL request :*rpl-request***

## Explanation

z/OS Debugger has attempted to close an existing Terminal Interface Manager debug session, but has encountered a SNA RPL error. This message is informational, but may be necessary to diagnose ongoing issues with the z/OS Debugger VTAM interface.

*rpl-errror-codes* is formatted as *rcfbr0rf*, where *rc* is the RPL error code, *fb* is the RPL feedback code, *r0* is the register 0 value returned from the RPL macroinstruction, and rf is the register 15 value returned by the RPL macroinstruction. All values are hexadecimal.

*rpl-sense-codes* is the hexadecimal value of the RplErrFdBk2 field, which contains the sense codes returned by the RPL macroinstruction.

*rpl-request* is the name of the RPL macroinstruction that caused the error.

**EQA9918U** **Unexpected LangX COBOL code sequence detected at <*hex-address*>. Abend.**

## Explanation

An unexpected situation was encountered. Contact z/OS Debugger support.

**EQA9919W** **Last lock owner: <*owner_token*>, New lock owner: <*owner_token*> .**

## Explanation

z/OS Debugger has detected a problem when trying to secure a lock on its hook table. z/OS Debugger has recovered from this problem. This message is reported in case it is needed for diagnostic purposes.

## Programmer response

This message can be ignored unless it is issued as part of another z/OS Debugger problem.

**EQA9920E** **EQA00OHT: Bad control block**

## Explanation

z/OS Debugger has found that one of its control blocks has been corrupted.

## Programmer response

If the application that is being debugged has caused a storage violation which might have corrupted z/OS Debugger storage, then investigate that storage violation. If the message is not related to an application problem, set a SLIP trap with *ACTION=SVCD* and *MSGID=EQA9920E* to capture an SVC dump and send the resulting SDUMP to IBM for analysis.

**EQA9921U** **SCREEN SIZE IS TOO LARGE. ROWS x COLUMNS MUST BE < 10923.**

## Explanation

When CICS or a VTAM terminal is used, the maximum screen size (number of rows times the number of columns) must be less than 10923. For example, 68x160 is acceptable but 69x160 is invalid.

**EQA9922I** ***loud-processing-message***

## Explanation

The LOUD parameter was used in one of these EQAOPTS commands: COMMANDSDSN, LOGDSN, LOGDSNALLOC, PREFERENCESDSN, SAVEBPDSNALLOC and SAVESETDSNALLOC. This is an informational message that indicates various processing or error conditions.

## Programmer response

Correct any errors noted in the message (if any).

**EQA9923E** **BPXWDYN not loaded. Return Code = *rc***

## Explanation

The BPXWDYN load module could not be loaded. This module is needed for the following EQAOPTS commands; LOGDSNALLOC, SAVEBPDSNALLOC and SAVESETDSNALLOC

## Programmer response

1. Have your system support person verify that the BPXWDYN module is available (in the search path).
2. Check the z/OS Debugger 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.

**EQA9924U** **The current user, *userid*, does not have RACF access to use z/OS Debugger.**

## Explanation

The current user has RACF access of NONE to the currently active RACF facility. Under CICS, the facility EQADTOOL.BROWSE.CICS is used. In other environments, the facility EQADTOOL.BROWSE.MVS is used. When z/OS Debugger is not able to determine the user ID, UNKNOWN is displayed.

## Programmer response

Ask the administrator of the RACF facility in effect to grant READ access or higher to this user ID.

For more information, check console message ICH408I.

**EQA9926I** ***** Allocate attempted from OPEN exit.**

## Explanation

z/OS Debugger attempted to allocate a file while the user program was processing in a OPEN exit. MVS does not allow this. This message should be followed by another message explaining the action taken by z/OS Debugger.

## Programmer response

Refer to the following message and take the appropriate action.

**EQA9927I** *****Did ESPIE with CeeCaaXHCL on**

## Explanation

This message is an internal diagnostic message and should not be seen unless you are using special processes as instructed by z/OS Debugger support.

## Programmer response

Report this message to z/OS Debugger support.

**EQA9928W** *****No ESPIE with CeeCaaXHCL on**

## Explanation

z/OS Debugger was entered for an SVC or overlay hook after Language Environment routines had set the CeeCAAXHCL flag but z/OS Debugger was not able to establish and ESPIE.

## Programmer response

If this message is followed by unexpected z/OS Debugger behavior, report the message to z/OS Debugger support.

**EQA9929E** **z/OS Debugger failed Product Registration. IFAEDREG RC = 00000004. The enclave will be terminated.**

## Explanation

z/OS Debugger called IFAEDREG to check for an enabled product registration for z/OS Debugger. IFAEDREG indicated that the check did not succeed.

## Programmer response

If z/OS Debugger is licensed to run on this machine as Application Delivery Foundation for z/OS, IBM Developer for z/OS Enterprise Edition, or IBM Debug for z/OS, the system programmer should register and enable the product registration per the *IBM z/OS Debugger Customization Guide*.

**EQA9930W** **EQA50DSP did not find a matching shift-in character so one was inserted.**

## Explanation

z/OS Debugger found a DBCS shift-out control character in the screen buffer, but was unable to find a corresponding shift-in control character. A shift-in control character has been inserted. Debugging results may be unpredictable now.

## Programmer response

Make sure there is a matching shift-in control character for each shift-out control character.

**EQA9931W** **Requested user interface not available. Processing will continue as if a QUIT DEBUG command had been entered.**

## Explanation

The requested user interface is not available so z/OS Debugger will quit processing events and the application will continue to run. (Note that any calls to restart z/OS Debugger are ignored.) This behavior was specified in the EQAOPTS customization module via the EQAXOPT macro invocation option NODISPLAY,QUITDEBUG.

## Programmer response

Make sure the user interface specified as a suboption in the TEST runtime option is correct and available. However, if you do not want the processing of a QUIT DEBUG command when this situation is detected then change the EQAXOPT macro invocation option to NODISPLAY,DEFAULT in your EQAOPTS customization module.

**EQA9932S** **Association does not exist for VTAM% specification**

## Explanation

The user name specified on the VTAM% option has not been associated to a terminal using the z/OS Debugger Terminal Interface Manager.

## Programmer response

Use the z/OS Debugger Terminal Interface Manager to associate the user name with a terminal and rerun the application.

**EQA9933W** **CEE3MBR failed for *load-module*. FC=*xxxxxxxx***

## Explanation

The Language Environment routine CEE3MBR failed and returned the indicated feedback code.

## Programmer response

Determine the cause of the error using the indicated feedback code.

**EQA9934W** **z/OS Debugger EQA00CIC: Error loading Program. See **.SEQASAMP(EQACCDS)**

## Explanation

z/OS Debugger Program EQA00CIC in load module EQA00OSX was unable to load the specified program.

## Programmer response

Ensure that the group (EQA) that contains the z/OS Debugger run time routines is in the group list used during CICS start-up. If required, rerun the EQACCSD job and restart the CICS region. Check the z/OS Debugger website for any applicable updates your system might require. If the problem persists, contact z/OS Debugger support.

**EQA9935E** **XXXXXXXX**

## Explanation

z/OS Debugger has experienced a problem, and is reporting diagnostic information (usually return codes given to z/OS Debugger 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

z/OS Debugger 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

z/OS Debugger is reporting that another CICS XPCFTCH global user exit has set a modified entry address (MEA) and prevented z/OS Debugger from any possible debugging of a specific non-Language Environment program. *XXXXXXXXX* 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 z/OS Debugger 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 z/OS Debugger, you will be unable to debug any non-Language Environment programs when the MEA was changed by another XPCFTCH exit. The behavior of z/OS Debugger in this kind of scenario will likely be unpredictable.

**EQA9938E** **Error in deactivate of NewProg exits.**

## Explanation

z/OS Debugger detected an error in attempting to deactivate the NewProg supporting exits.

## Programmer response

An error has likely occurred during z/OS Debugger CICS region initialization. Ensure that z/OS Debugger is properly installed in the CICS region. Also verify that the z/OS Debugger *hlq*.SEQAMOD data set is in the region DFHRPL DD and the CICS resource definitions from *hlq*.SEQASAMP(EQACCSD) have been added.

**EQA9939I** **IBM z/OS Debugger NewProg support deactivated.**

## Explanation

z/OS Debugger 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 z/OS Debugger NonLE exits enabled.**

## Explanation

z/OS Debugger 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 EQA0CPLT and starting with INITPARM=(EQA0CPLT='NLE').

**EQA9941I** **IBM z/OS Debugger NewProg support activated.**

## Explanation

z/OS Debugger 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 z/OS Debugger Screen stack exits enabled.**

## Explanation

z/OS Debugger 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 z/OS Debugger from starting the exits when it starts a debug session for a user. This was accomplished using PLTPI program EQA0CPLT and starting with INITPARM=(EQA0CPLT='STK').

**EQA9943E** **Error in activate of NonLE exits.**

## Explanation

z/OS Debugger detected an error while attempting to activate the non-Language Environment supporting exits.

## Programmer response

The error most likely occurred during z/OS Debugger CICS region initialization. Verify that the z/OS Debugger *hlq*.SEQAMOD data set is the in the region DFHRPL and the CICS resource definitions from *hlq*.SEQASAMP(EQACCSD) have been added.

**EQA9944E** **Error in activate of NewProg exits.**

## Explanation

z/OS Debugger detected an error in attempting to activate the NewProg supporting exits.

## Programmer response

An error has likely occurred during z/OS Debugger CICS region initialization. Ensure that Language Environment is properly installed in the CICS region. Also verify that the z/OS Debugger *hlq*.SEQAMOD data set is in the region DFHRPL DD and the CICS

resource definitions from *hlq*.SEQASAMP(EQACCSD) have been added.

**EQA9945S z/OS Debugger DTRCB Unavailable.**

## Explanation

The z/OS Debugger non-Language Environment CICS global user exits were made active in a CICS region where z/OS Debugger did not successfully initialize during CICS region startup.

## Programmer response

Ensure that Language Environment is installed in the CICS region and verify that the z/OS Debugger installation steps were executed properly. For example, the z/OS Debugger *hlq*.SEQAMOD data set is in the DFHRPL DD and that the *hlq*.SEQASAMP(EQACCSD) job was run to add z/OS Debugger resource definitions to the CICS region.

**EQA9946S EQA01SVC TCBSVCA2 invalid - *xxx* where *xxx*=start, stop, term, strtX, stopX**

## Explanation

Internal z/OS Debugger SVC Screening error or z/OS Debugger SVC (109 extended code 51) issued outside z/OS Debugger. The SVC will abend.

## Programmer response

If using z/OS Debugger contact your IBM representative.

**EQA9947S EQA01SVC EQASVCSCREEN N/T create**

## Explanation

Internal z/OS Debugger SVC Screening error or z/OS Debugger SVC (109 extended code 51) issued outside z/OS Debugger. The SVC will abend.

## Programmer response

If using z/OS Debugger contact your IBM representative.

**EQA9948S EQA01SVC No DTRCB at InitScreen**

## Explanation

Internal z/OS Debugger SVC Screening error or z/OS Debugger SVC (109 extended code 51) issued outside z/OS Debugger. The SVC will abend.

## Programmer response

If using z/OS Debugger contact your IBM representative.

**EQA9949S or EQA9949I TCB SVC Screening already active and NOOVERRIDE is specified by EQAOPTS.**

## Explanation

SVC Screening is in use by another product and SVC Screening CONFLICT=NOOVERRIDE is specified by EQAOPTS. Handling of non-Language Environment events is not available. Debugging of non-Language Environment programs will be restricted in this z/OS Debugger session.

## System action

The z/OS Debugger was invoked with an EQAOPTS options module that specified CONFLICT=NOOVERRIDE. z/OS Debugger will not intercept non-Language Environment events and, therefore debugging of no-Language Environment programs will be limited.

## Programmer response

If you do not need to debug non-Language Environment programs or to intercept non-Language Environment events, no action is required. Otherwise, you must terminate the prior use of SVC SCREENING (TCBSVCS, TCBSVCSP, TCBSVCA2) before starting z/OS Debugger or have your installer provide an EQAOPTS that specified CONFLICT=OVERRIDE. CONFLICT=OVERRIDE allow z/OS Debugger to save and restore the previous use of SVC SCREENING (TCBSVCS, TCBSVCSP, TCBSVCA2).

**EQA9950E Error enabling XEIIN screen exit.**

## Explanation

z/OS Debugger detected an error during the ENABLE of a required CICS exit program.

## Programmer response

Determine if the z/OS Debugger *hlq*.SEQAMOD library is available in the DFHRPL concatenation of the CICS region and the resource definitions provided in *hlq*.SEQASAMP(EQACCSD) have been added to the CICS region that is initializing.

**EQA9951E Error enabling XEIOUT screen exit.**

## Explanation

z/OS Debugger detected an error during the ENABLE of a required CICS exit program.

## Programmer response

Determine if the z/OS Debugger *hlq*.SEQAMOD library is available in the DFHRPL concatenation of the CICS region and the resource definitions provided in *hlq*.SEQASAMP(EQACCSD) have been added to the CICS region that is initializing.

**EQA9952E** **Error in locate of z/OS Debugger RCB.**

## Explanation

z/OS Debugger CICS PLT program EQA0CPLT detected an error during the search for z/OS Debugger region-level resources.

## Programmer response

An error has likely occurred during z/OS Debugger CICS region initialization. Ensure that Language Environment is properly installed in the CICS region. Also, verify that the z/OS Debugger *hlq*.SEQAMOD data set is in the region DFHRPL DD and the CICS resource definitions from *hlq*.SEQASAMP(EQACCSD) have been added.

**EQA9953E** **NOTAUTH Error issuing CICS EXTRACT EXIT.**

## Explanation

z/OS Debugger detected a NOTAUTH condition during an EXTRACT EXIT call to CICS.

## Programmer response

Determine if the current z/OS Debugger 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 z/OS Debugger PLT initialization routine, EQA0CPLT, is recommended. Refer to the *IBM z/OS Debugger Customization Guide* for information on EQA0CPLT.

**EQA9954E** **Invalid EXIT ENABLE request.**

## Explanation

z/OS Debugger CICS PLT program EQA0CPLT 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 z/OS Debugger screen-stack exits. Contact IBM Support Center and report the error.

**EQA9955E** **User not authorized for EXIT ENABLE.**

## Explanation

z/OS Debugger CICS PLT program EQA0CPLT 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

z/OS Debugger CICS PLT program EQA0CPLT detected an error during the ENABLE of a required CICS exit program.

## Programmer response

Determine if the z/OS Debugger *hlq*.SEQAMOD library is available in the DFHRPL concatenation of the CICS region and the resource definitions provided in *hlq*.SEQASAMP(EQACCSD) have been added to the CICS region that is initializing.

**EQA9957E** **Invalid CICS release. Latest CICS used.**

## Explanation

z/OS Debugger detected an unsupported release of CICS and defaults to the latest release of CICS that this version of z/OS Debugger supports.

## Programmer response

Determine if z/OS Debugger is starting on a supported release of CICS. See the Program Directory for IBM z/OS Debugger for the list of CICS releases that are supported.

**EQA9958I** **IBM z/OS Debugger CICS PLT init start.**

## Explanation

z/OS Debugger program EQA0CPLT is starting. This program activates various z/OS Debugger resources during CICS region startup. This includes starting up z/OS Debugger support for running in CICS multi-region configurations (`INITPARM=(EQA0CPLT='NWP')`) and starting z/OS Debugger screen stack exits once at region initialization (`INITPARM=(EQA0CPLT='STK')`) and starting z/OS Debugger non-Language Environment-supporting exits (`INITPARM=(EQA0CPLT='NLE')`). Combinations of these selections are also supported. For example:

```
INITPARM=(EQA0CPLT='NWP,STK,NLE')
```

**EQA9959I** **IBM z/OS Debugger CICS PLT init end.**

## Explanation

z/OS Debugger program EQA0CPLT is ending. This program activates various z/OS Debugger resources during CICS region startup. This includes starting up z/OS Debugger support for running in CICS multi-region configurations (`INITPARM=(EQA0CPLT='NWP')`) and starting z/OS Debugger screen stack exits once at region initialization (`INITPARM=(EQA0CPLT='STK')`) and starting z/OS Debugger non-Language Environment-supporting exits (`INITPARM=(EQA0CPLT='NLE')`).

**EQA9960I** **Program abend: *Abcode* Prog: *Abprogram* Ret@: *XXXXXXXX***

## Explanation

z/OS Debugger 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 z/OS Debugger 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 z/OS Debugger 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 z/OS Debugger to capture abends. This message is written to the CICS region's JES message log.

**EQA9961I** **Program interrupt: *Intcd* Prog: *Abprogram* Int@: *XXXXXXXX***

## Explanation

z/OS Debugger has detected program check interrupt code *Intcd* 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 z/OS Debugger 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 `z/OS Debugger` 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 `z/OS Debugger` to capture program checks. This message is written to the CICS region's JES message log.

**EQA9962I** **IBM z/OS Debugger Exit Activation PLT start.**

## Explanation

z/OS Debugger program EQANCPLT is starting. This program activates the z/OS Debugger non-Language Environment CICS global exits which must be executed as either a stage 2 or 3 PLT post initialization program.

**EQA9963I** **IBM z/OS Debugger Exit Activation PLT end.**

## Explanation

z/OS Debugger program EQANCPLT is ending. This program activates the z/OS Debugger 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

z/OS Debugger 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 CEEDBGEVNTEXT Error. RC: *RC***

## Explanation

z/OS Debugger 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 z/OS Debugger SVC detected. V5R1 or later SVCs required for Non-LE support.**

## Explanation

z/OS Debugger is unable to initialize for a non-Language Environment assembler program under CICS due to back-level z/OS Debugger SVCs.

## Programmer response

Verify that the latest version of the z/OS Debugger SVCs are installed. The level of the SVCs can be checked by running the exec in *hlq*.SEQAEXEC(EQADTSVC).

**EQA9967I EQA00SVC Level:*Eqa00svcVersion* EQA01SVC Level:*Eqa01svcVersion***

## Explanation

z/OS Debugger is unable to initialize for a non-Language Environment assembler program under CICS due to back-level z/OS Debugger SVCs. This message occurs with message EQA9966E and indicates the detected levels of the two z/OS Debugger SVCs.

## Programmer response

Verify that the latest version of the z/OS Debugger SVCs are installed. The level of the SVCs can be checked by running the exec in *hlq*.SEQAEXEC(EQADTSVC). For non-Language Environment support, EQA00SVC must be at least 04 and EQA01SVC at least 05.

**EQA9968E Invalid Exit Type ...**

## Explanation

z/OS Debugger 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=XPCFTCH exit or E=E=XEIIN 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 Select O=On or F=Off**

## Explanation

z/OS Debugger 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 CICS exit activation successful.**

## Explanation

z/OS Debugger CICS global user exits activated successfully.

**EQA9971I CICS exit deactivation successful.**

## Explanation

z/OS Debugger CICS global user exits deactivated successfully.

**EQA9972I z/OS Debugger *glueexitname* CICS exit now ON.**

## Explanation

z/OS Debugger CICS exit activation transaction DTCX successfully started the *glueexitname* exit where *glueexitname* is either XPCFTCH, XEIIN, XEIOUT, XPCTA, or XPCHAIR.

**EQA9973I z/OS Debugger *glueexitname* CICS exit now OFF.**

## Explanation

z/OS Debugger CICS exit activation transaction DTCX successfully stopped the *glueexitname* exit where *glueexitname* is either XPCFTCH, XEIIN, XEIOUT, XPCTA, or XPCHAIR.

**EQA9974E** **Error enabling *glueexitname* - *dtexitname*.**

## Explanation

z/OS Debugger CICS exit activation transaction DTCX was unable to activate *glueexitname* - *dtexitname* where *glueexitname* is either XPCFTCH, XEIIN, XEIOUT, XPCTA, or XPCHAIR and *dtexitname* is either EQANCFTC, EQANCXEI, EQANCXAB, or EQANCXHA.

## Programmer response

Verify that the latest *hlq*.SEQASAMP(EQACCSD) CICS resource definitions are installed and the z/OS Debugger *hlq*.SEQAMOD library is in the CICS DFHRPL DD concatenation. If this has already been done, contact IBM support center and report the error.

**EQA9975E** **Error disabling *glueexitname* - *dtexitname*.**

## Explanation

z/OS Debugger CICS exit activation transaction DTCX was unable to deactivate *glueexitname* - *dtexitname* where *glueexitname* is either XPCFTCH, XEIIN, XEIOUT, XPCTA, or XPCHAIR and *dtexitname* is either EQANCFTC, EQANCXEI, EQANCXAB, or EQANCXHA.

## Programmer response

Verify that the latest *hlq*.SEQASAMP(EQACCSD) CICS resource definitions are installed and the z/OS Debugger *hlq*.SEQAMOD library is in the CICS DFHRPL DD concatenation. If this has already been done, contact IBM support center and report the error.

**EQA9976I** **z/OS Debugger *glueexitname* exit already active.**

## Explanation

The requested CICS global user exit for non-Language Environment assembler support was already active. *glueexitname* is either XPCFTCH, XEIIN, XEIOUT, XPCTA, or XPCHAIR.

**EQA9977E** ***dtsvcname* is backlevel. Exits not enabled.**

## Explanation

z/OS Debugger is unable to activate the non-Language Environment CICS global exits due to back-level z/OS Debugger SVCs. *dtsvcname* is either EQA00SVC or EQA01SVC.

## Programmer response

Verify that the latest version of the z/OS Debugger SVCs are installed. The level of the SVCs can be checked by running the exec in *hlq*.SEQAEXEC(EQADTSVC). For non-Language Environment support, EQA00SVC must be at least 04 and EQA01SVC at least 05.

**EQA9978I** **Unable to set hook because debug data cannot be located for *program_name***

## Explanation

z/OS Debugger 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 z/OS Debugger by using the SET DEFAULT LISTINGS or SET SOURCE command, the EQAUEDAT user exit or the EQADEBUG DD name.

**EQA9979I** **Unable to load user program *UserProgram***

## Explanation

EQANMDBG was unable to load the user program specified as the first positional parameter.

## 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** ***error_description***

## Explanation

*error_description* is replaced with the following text:

- z/OS Debugger unrecoverable CICS task error.
- CICS abend code is *aaaa*
- z/OS Debugger session ending.

(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** **EQAx0STO Internal Error *WTO_Num***

## Explanation

The internal z/OS Debugger storage allocation chains have been corrupted. Other forms of this message might also appear with additional information about the error.

## Programmer response

Ensure that your program is not overwriting z/OS Debugger storage. Check the z/OS Debugger web site for any applicable service updates your system might require. If the problem persists, contact z/OS Debugger support.

**EQA9982E** **A non-zero response code was returned from EXEC CICS "command". Resp value = EIBRESP**

## Explanation

z/OS Debugger has issued an EXEC CICS command, and has received an unexpected response.

## 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** **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

EQANMDBG 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 z/OS Debugger.**

## Explanation

The non-Language Environment version of z/OS Debugger (EQANMDBG) was invoked but the z/OS Debugger SVCs required for dynamic debug support have not been installed.

## Programmer response

Have your system programmer complete the installation of the required z/OS Debugger SVCs.

**EQA9986E** **Error in CEEEV006 loading OSI**

## Explanation

One of the required z/OS Debugger load modules is missing.

## Programmer response

Contact your system programmer to verify the proper installation of z/OS Debugger.

**EQA9987I** **First parameter to *load_module* is not addressable. z/OS Debugger might not be able to 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 z/OS Debugger can add the TEST parameter.

## Programmer response

Correct the parameter address passed to the `LINK` SVC.

**EQA9988S** **z/OS Debugger has terminated the enclave.**

## Explanation

A z/OS Debugger `QUIT` command was issued in a multi-enclave environment.

## Programmer response

No action is necessary.

**EQA9989I EQANMDBG requires z/OS Debugger V5R1 or later SVC.**

## Explanation

The z/OS Debugger V5R1 (or later) non-Language Environment program was invoked but the installed version of the z/OS Debugger SVCs were from a earlier version of z/OS Debugger.

## Programmer response

Have your system programmer install the current z/OS Debugger SVCs.

**EQA9990I LOAD detect.**

## Programmer response

This is an internal z/OS Debugger 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 z/OS Debugger session. Check the z/OS Debugger website for any applicable service updates your system might require. If the problem persists, contact z/OS Debugger support.

**EQA9992E Internal error processing Language Environment service *FunctionCode***

## Explanation

An internal z/OS Debugger error has occurred.

## Programmer response

Check the z/OS Debugger web site for any applicable service updates your system might require. If the problem persists, contact z/OS Debugger 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.

## System action

If the application has been compiled with hooks (for example, with `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 z/OS Debugger RCB**

## Explanation

There is insufficient storage for z/OS Debugger to initialize.

## Programmer response

Increase the region size available to the program and rerun.

**EQA9995E REQUIRED TEXT**

## Explanation

All EQA9995E messages signify that a severe error has occurred in the z/OS Debugger SVC routine while processing an 0A91 instruction.

## Programmer response

1. Make sure none of the applications you are debugging issue the reserved 0A91 (SVC 145) instruction.
2. If you have non-IBM products installed on your system, make sure none of them issue the reserved 0A91 (SVC 145) instruction.
3. Try running the Dynamic Debug facility IVP (Installation Verification Program). This program can be found in member EQAWIVPS of data set *hlq*.SEQASAMP.
4. Have your system support person re-install the z/OS Debugger SVCs using member EQAWISVC of data set *hlq*.SEQASAMP and then run the IVP (see

step 3). If the IVP runs normally, have your support person add *hlq*.SEQALPA to your system LPA list. This ensures that the z/OS Debugger SVCs are available after the next IPL.

5. Check the z/OS Debugger 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 z/OS Debugger SVC routine EQA01SVC. EQA01SVC is SVC 109 with extended function code 51.

## Programmer response

Check the z/OS Debugger 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 z/OS Debugger 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 z/OS Debugger DTCN profile skipped.**

## Explanation

The z/OS Debugger 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 z/OS Debugger Module
Please contact your IBM Representative
Failure address - *xxxxxxxx*
Program Check at *module+offset*

## Programmer response

See details of message issued for appropriate response or indication of potential problem. Check the z/OS Debugger web site for any applicable service updates your system might require. If the problem persists, contact z/OS Debugger support.

**EQA9999W ERROR DESCRIPTION**

## Explanation

Warning Message issued by z/OS Debugger Module

## Programmer response

See details of message issued for appropriate response or indication of potential problem. Check the z/OS Debugger web site for any applicable service updates your system might require.

# Chapter 10. Debug Manager messages

All messages 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 EQACM *nnnx*, where CM indicates that the message is a Debug 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 z/OS Debugger attempted to fix.

**E**
An ***error*** message describes an error that Debug Manager 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 Manager from continuing.

**Symbols in messages**

Many of the Debug Manager 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:

```
EQACM001I Debug Manager startup complete (external_port/internal_port),
build date: date, time,trace level: trace_level
```

**EQACM001I** **Debug Manager startup complete (*external_port*/*internal_port*), build date:*date*, *time*, trace level:*trace_level***

## Explanation

All the components were successfully loaded. The information of the external/internal ports used, build date, and trace level is displayed.

## System action

None.

## User response

No action is required.

**EQACM010E** **Debug Manager must be APF authorized.**

## Explanation

The Debug Manager executable does not reside in an APF-authorized library.

## System action

Debug Manager exits.

## User response

Ask the system administrator to mark the Debug Manager library as APF-authorized.

**EQACM011E** **Debug Manager invoked with invalid parameters**

## Explanation

Debug Manager was started. However, the parameters that invoked Debug Manager contain an invalid port number or SVC number. The range for port number is 1 - 65535, and the range for SVC number is 200 - 255.

The SVC number is needed only in versions earlier than V14.0.

### System action

Debug Manager exits.

### User response

Update JCL with valid parameters.

**EQACM0012E** **A secured z/OS Debugger connection to the Debug Manager on *IP address:external_port* could not be established. Refer to the IBM z/OS Debugger Customization Guide to ensure that the Remote System Explorer (RSE) and the Debug Manager host components are configured correctly.**

### Explanation

An attempt was made to establish a secured connection to the Debug Manager on *IP address:external_port*. However, the connection cannot be established because RSE and Debug Manager host components are not configured in the same security mode.

### System action

A secured connection cannot be established.

### User response

Ensure that RSE and Debug Manager are configured in the same security mode. By default, the host certificates used by the secured RSE connections will be re-used by the secured z/OS Debugger connection. If the RSE and Debug Manager ports are configured to use different certificates or signing authorities, use the z/OS Debugger Preference page to override the keystore and certificates used by z/OS Debugger. Ask the System z system programmer for the keystore file.

**EQACM0018E** **The debug session cannot be established because Debug Manager is in secure mode but RSE is in unsecured mode.**

### Explanation

The debug session is not established.

### System action

The debug session cannot be established.

### User response

Connect to a secured RSE port or change Debug Manager to unsecured mode to start a debug session.

**EQACM0019W** **A secured z/OS Debugger connection cannot be established. Do you want to try in unsecured mode?**

### Explanation

The debug session cannot be established because Debug Manager is in secured mode but RSE is in unsecured mode.

### System action

The debug session is not established.

### User response

Click **Yes** to establish the debug session in unsecure mode. Click **No** to stop the current handshake.

**EQACM100E** **Invalid command entered**

### Explanation

The command entered is not supported.

### System action

None.

### User response

Enter a valid command and try again. For more information about the Debug Manager commands, see *IBM z/OS Debugger Customization Guide*.

**EQACM101I** **TRACE command processed normally**

### Explanation

The trace level was updated successfully with the TRACE command.

### System action

None.

### User response

No action is required.

**EQACM102E** **TRACE command has invalid option**

## Explanation

A TRACE command was issued to update the trace level. However, the trace level was not updated successfully because the trace level entered is invalid, or the command format is not supported.

## System action

The trace level is not updated.

## User response

Follow the correct command format, and enter a valid trace level. The valid trace levels are E, I, D, and V.

**EQACM103I There are no active users.**

## Explanation

No users are connected to RSE.

## System action

None.

## User response

No action is required.

**EQACM104I Listing active users**

## Explanation

The users that are connected to RSE are displayed.

## System action

None.

## User response

No action is required.

**EQACM105I Incompatible RSE and DBM**

## Explanation

In Debug Tool compatibility mode, both RSE and DBM must be at version 1.1.1. Otherwise, the debug session cannot be established.

## System action

The debug session is not established.

## User response

To start a debug session in Debug Tool compatibility mode, ensure that both RSE and DBM are at V1.1.1.

**EQACM112I Invalid SVC number *svc_number* (valid values are 200 to 255)**

## Explanation

Debug Manager was started. However, it was invoked by parameters that contain an invalid SVC value. The valid values for SVC are 200 - 255. You can ignore this message if you are using V14.0, or later.

## System action

Processing stops.

## User response

Update the SVC number in JCL with a valid value.

**EQACM114I SVC *svc_number* replaced (*version*, *time*).**

## Explanation

SVC was replaced with a newer version. SVC will be reverted to the original version on exit. You can ignore this message if you are using V14.0, or later.

## System action

None.

## User response

No action is required.

**EQACM115I SVC *svc_number* installed (*version*, *time*).**

## Explanation

SVC was successfully installed. You can ignore this message if you are using V14.0, or later.

## System action

None.

## User response

No action is required.

**EQACM116I Current SVC svc_number is *version*, *time*.**

## Explanation

The number and version of the current SVC is displayed. You can ignore this message if you are using V14.0, or later.

### System action

None.

### User response

No action is required.

**EQACM117E Internal error - cannot update SVC**

### Explanation

SVC cannot be updated because of an internal error. This message applies only to versions earlier than V14.0.

### System action

SVC is not updated.

### User response

Ask the system administrator to reinstall SVC, and restart Debug Manager. IPL the system if necessary.

**EQACM120I SVC code restored to the original version**

### Explanation

SVC was reverted to the system installed version on exit.

### System action

None.

### User response

No action is required.

**EQACM130E Failed to create communications token**

### Explanation

Debug Manager cannot create a name and token pair.

### System action

Debug Manager exits.

### User response

Check the system abend code. IPL the system if necessary.

**EQACM150E This program is not authorized to start as a job.**

### Explanation

Debug Manager was started as a batch job. However, the RACF profile defined to start Debug Manager as a user job is missing, or the security class is not active.

### System action

Debug Manager failed to start.

### User response

Define a RACF profile named EQADTOOL.START.BATCH.*jobname*.*port*. For more information, see "Starting Debug Manager as a user job" in IBM z/OS Debugger Customization Guide.

# Chapter 11. Debug Profile Service API messages

All messages 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 EQAPS *nnnx*, where PS indicates that the message is a Debug Profile Service API 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 z/OS Debugger attempted to fix.

**E**
An ***error*** message describes an error that Debug Profile Service API 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 Profile Service API from continuing.

**EQAPS1000E The `dtcn.ports` file or TCP/IP service resource is not configured correctly, or the CICS region is offline.**

### Explanation

Debug Profile Service was unable to contact the specified region, or the region was not correctly configured.

### System action

The debug profile was not activated.

### User response

Ensure that the CICS region is running.

If necessary, ask the system programmer to confirm whether the ports configured in `/etc/debug/dtcn.ports` are accurate and verify that the DTCN API TCPIPSERVICE is configured as described in "Defining the CICS TCPIPSERVICE resource" in the *IBM z/OS Debugger Customization Guide*.

**EQAPS1001E The password is incorrect.**

### Explanation

The password entered was incorrect for the user ID.

### System action

Connection to Debug Profile Service fails.

### User response

Provide a correct password for the specified user ID.

**EQAPS1002E Access was revoked for the user ID. Reset or unlock your password and try again.**

### Explanation

Access to the system was revoked after incorrect passwords were provided in previous attempts.

### System action

Connection to Debug Profile Service fails.

### User response

Unlock or reset your password before you try again.

**EQAPS1003E The user ID is invalid.**

### Explanation

Authentication failed because an invalid user ID was provided. The user ID does not exist or has no access to this system.

### System action

Connection to Debug Profile Service fails.

### User response

Provide a user ID that has access to the system.

**EQAPS1004E The password or password phrase has expired. Reset your password and try again.**

### Explanation

The password or password phrase provided has expired.

### System action

Connection to Debug Profile Service fails.

### User response

Reset your password before you try again.

**EQAPS1005E Debug Profile Service is already running. Stop the existing instance before starting a new one.**

### Explanation

An attempt was made to start Debug Profile Service again when an instance was already running. Only a single instance of Debug Profile Service can be running on the system.

### System action

Debug Profile Service failed to start again.

### User response

Stop the first instance of Debug Profile Service before you try again.

**EQAPS1006E Debug Profile Service is not supported by the currently installed SVC.**

### Explanation

Debug Profile Service attempted to register its ports using the z/OS Debugger SVC.

### System action

Debug Profile Service failed to start.

### User response

Ensure that the z/OS Debugger SVC of Version14.2 or later is installed.

**EQAPS1007E The CICS profile was not activated because the region requires that a terminal ID be specified.**

### Explanation

The region requires a terminal ID to activate the CICS profile.

### System action

The CICS profile was not activated.

### User response

Specify a terminal ID.

**EQAPS1008E The CICS profile was not activated because the region requires that at least one load module be specified.**

### Explanation

The region requires at least one load module to activate the CICS profile.

### System action

The CICS profile was not activated.

### User response

Specify at least one load module.

**EQAPS1009E The CICS profile was not activated because the region requires that a transaction be specified.**

### Explanation

The region requires a transaction to activate the CICS profile.

### System action

The CICS profile was not activated.

### User response

Specify a transaction.

**EQAPS1010E The CICS profile was not activated because the region requires that**

**at least one compile unit be specified.**

## Explanation

The region requires at least one compile unit to activate the CICS profile.

## System action

The CICS profile was not activated.

## User response

Specify at least one compile unit.

**EQAPS1011E** **The CICS profile was not activated because the region requires that a user ID be specified.**

## Explanation

The region requires a user ID to activate the CICS profile.

## System action

The CICS profile was not activated.

## User response

Specify a user ID.

**EQAPS1012E** **The CICS profile was not activated because the region requires that a net name be specified.**

## Explanation

The region requires a net name to activate the CICS profile.

## System action

The CICS profile was not activated.

## User response

Specify a net name.

**EQAPS1013E** **The CICS profile was not activated because the region requires that a client IP be specified.**

## Explanation

The region requires a client IP to activate the CICS profile.

## System action

The CICS profile was not activated.

## User response

Specify a client IP.

**EQAPS1014E** **Specify environment variables *'port'* and *'SECURE'* in the `eqaprof.env` file to start Debug Profile Service on a specific port.**

## Explanation

A port needs to be specified in the `eqaprof.env` file to start Debug Profile Service.

## System action

Debug Profile Service was not started.

## User response

Specify `port=`*`port_number`* and `SECURE=Y/N` in the `eqaprof.env` file, and start Debug Profile Service again.

**EQAPS1015E** **The user ID is not authorized to manage other users' profiles.**

## Explanation

Only authorized user IDs can modify or delete other users' profiles.

## User response

If you need to manage other users' profiles, ask the security administrator to give you UPDATE permission to the `EQADTOOL.DTCNCHNGEANY` RACF profile. For more information, see "Defining who can create, modify, or delete DTCN profiles" in the *IBM z/OS Debugger Customization Guide*.

**EQAPS2000E** **The profile does not contain a specific user ID, IP, or terminal ID. Debug sessions might be triggered for other users unexpectedly.**

## Explanation

The profile does not contain a specific user ID, IP, or terminal ID. The profile that contains generic values might unexpectedly trigger z/OS Debugger to consume unnecessary resources.

## User response

Evaluate if this profile is causing problems. Ask the owner to specify a user ID, IP, or terminal ID in the profile. You can delete the profile in z/OS Debugger Profile Management if necessary.

To prevent generic profiles, you can specify the DTCNFORCE*xxxx* settings to make some fields mandatory.

**EQAPS2001E The profile contains generic values for load module or program name without a transaction filter specified. Debug sessions might be triggered unexpectedly.**

## Explanation

The profile contains an empty or wildcard value for load module or program, and there is no additional transaction filter. The profile that contains generic values might unexpectedly trigger z/OS Debugger to consume unnecessary resources.

## User response

Evaluate if this profile is causing problems. Ask the owner to modify the profile with more specific values for the load module or program name, or alternatively provide a transaction name. You can delete the profile in z/OS Debugger Profile Management if necessary.

To prevent generic profiles, you can specify the DTCNFORCE*xxxx* settings to make some fields mandatory.

# Chapter 12. Non-Language Environment IMS messages

Each message has a number of the form EQAI*nnnnx*, 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 z/OS Debugger attempted to fix.

**E**
An ***error*** message describes an error that z/OS Debugger 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 z/OS Debugger from continuing.

**Symbols in messages**

Many of the z/OS Debugger 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 z/OS Debugger located on the host notifies you of errors associated with debugging functions carried out by the host.

**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**

## Explanation

Load z/OS Debugger non-Language Environment front end failed.

## User response

Consult your system programmer.

**EQAI1009S Invalid EQASET specification**

## Explanation

Load z/OS Debugger 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* (EQA*user_ID*). The return code is *returncode*.

## User response

Run the EQASET transaction from the terminal where the application transaction is invoked.

**EQAI1020S Retrieve *token* failed**

## Explanation

Fail to retrieve *token*.

## User response

Start EQASET transaction with a valid keyword (MFI, TCP, VTCP, VTAM).

**EQAI2005I DEBUG SET ON FOR MFI SETTING**

## Explanation

Debugging is turned on and setting is MFI.

## User response

None.

**EQAI2006I DEBUG SET ON FOR TCP SETTING**

## Explanation

Debugging is turned on and setting is TCP.

## User response

None.

**EQAI2007I DEBUG SET ON FOR VTCP SETTING**

## Explanation

Debugging is turned on and setting is VTCP.

## User response

None.

**EQAI2008I DEBUG SET OFF**

## Explanation

Debugging is turned off.

## User response

None.

**EQAI2009I DEBUG SET ON FOR SPECIFIED MFI LU**

## Explanation

Debugging is turned on and setting is the specified MFI LU name.

## User response

None.

**EQAI2010I DEBUG SET ON FOR SPECIFIED TCP IP**

## Explanation

Debugging is turned on and setting is the specified TCP IP address.

## User response

None.

**EQAI2011I DEBUG SET ON FOR SPECIFIED VTCP IP**

## Explanation

Debugging is turned on and setting is the specified VTCP IP address.

## User response

None.

**EQAI2012I VALID KEYWORDS: ON, OFF, MFI=, TCP=, VTCP, VTAM=, STATUS**

## Explanation

Valid keywords for EQASET transaction.

### User response

None.

**EQAI2013E USERID FROM IMS ENVIRONMENT IS BLANK**

### Explanation

User ID is blank or cannot be found.

### User response

Contact your system programmer.

**EQAI2014E NO DATA RECEIVED**

### Explanation

EQASET transaction is entered without keyword.

### User response

Reference documentation for EQASET definition and usage.

**EQAI2015E NAME/TOKEN SAVE FAILED RC= *returnCode***

### Explanation

Fail to save setting value.

### User response

Contact your system programmer.

**EQAI2016E MFI/TCP/VTCP VALUE MUST BE SET TO USE KEYWORD ON**

### Explanation

No setting exists when debugging is turned on with EQASET ON.

### User response

Set value of one of the settings (MFI, TCP, VTCP) with EQASET transaction.

**EQAI2017I NO STATUS**

### Explanation

No on/off switch exists.

### User response

Set value of one of the settings (MFI, TCP, VTCP) with EQASET transaction.

**EQAI2018I DEBUG SET ON FOR SPECIFIED VTAM LU**

### Explanation

Debugging is turned on and setting is the specified VTAM LU associated with user id.

### User response

None.

**EQAI2019I DEBUG SET ON FOR VTAM SETTING**

### Explanation

Debugging is turned on and setting is VTAM.

### User response

None.

**EQAI2020I USERID:*userid* STATUS:*status* ID:*value***

### Explanation

Displays the current debugging preferences for IMS user ID *userid*. *status* is one of the following: OFF, VTAM, TCP, VTCP, MFI. *value* is one of the following values:

- Blank, if *status* is OFF.
- TSO or IMS user ID, if *status* is VTAM.
- IP address (and port number), if *status* is TCP or VTCP.
- Network identifier (and terminal LU name), if *status* is MFI.

### User response

None.

**EQAI2021I The TSO user ID *tsoid* has been associated with the current IMS terminal.**

### User response

None.

**EQAI2022I z/OS Debugger SVC IS BACKLEVEL. CANNOT USE EQASET.**

## Explanation

The EQASET transaction detected a down level z/OS Debugger SVC EQA01SVC(IGX00051). Version 13 is required for this version of EQASET.

## User response

Have your installer install the correct z/OS Debugger SVCs.

# Chapter 13. Load Module Analyzer Messages

All messages 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 EQALM*nnnx*, 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 z/OS Debugger 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.

**EQALM000S** ****** UNKNOWN ERROR ******

## Explanation

An unexpected or unrecognized error has occurred.

**EQALM001E** ****** Unable to open *filename* ******

## Explanation

This indicated file cannot be opened.

**EQALM002I** ****** Unable to open EQAIN. All members will be processed. ******

## Explanation

The EQAIN file was not allocated. All members of the PDS(E) concatenation allocated to EQALIB will be processed.

**EQALM003E** ****** Unknown member specified in SELECT statement ******

## Explanation

The member specified on the SELECT statement was not found in the EQALIB concatenation.

**EQALM004E** ****** Unrecognized control statement ******

## Explanation

An unrecognized control statement was encountered while processing the EQAIN file.

**EQALM005E** ****** Work area overflow ******

## Explanation

An internal work area has overflowed.

**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 IEWBIND return and reason codes.

**EQALM007W** ****** text is an unrecognized option ******

## Explanation

The specified text is not a supported option.

**EQALM008S** ****** z/OS Debugger failed Product Registration. ******

## 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 SEQAMOD data set shipped with the Japanese feature of z/OS Debugger.

**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 IEWBIND 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 IEWBIND. See the appropriate Binder documentation for a description of the indicated return code and reason code.

# Chapter 14. z/OS Debugger Language Environment user exit messages

Each message has a number of the form EQAU*nnnx*, where EQAU indicates that the message is a z/OS Debugger Language Environment user exit message, *nnn* is the number of the message, and *x* indicates the severity level of each message.

The following messages apply to all environments.

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 z/OS Debugger attempted to fix.

**E**
An ***error*** message describes an error that z/OS Debugger 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 z/OS Debugger from continuing.

**Symbols in messages**

Many of the z/OS Debugger 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 z/OS Debugger located on the host notifies you of errors associated with debugging functions carried out by the host.

The following messages apply to all environments:

**EQAU999I NO READ ACCESS OF z/OS DEBUGGER USER EXIT DATA SET**

## Explanation

z/OS Debugger tries to read the user exit data set but the access is denied because of RACF or other security protection.

## System action

No debug session is started.

## User response

Contact system administrator to allow the owner ID of a job (batch), WLM address space (Db2 stored procedure), or IMS region (IMS transaction) read access to the data set.

**EQAU999I SEE ICH408I MESSAGE**

## Explanation

Direct the user to the ICH408I message for more details of the NO READ ACCESS OF z/OS Debugger USER EXIT DATA SET problem.

## System action

None.

## User response

Look for ICH408I message in the system log.

# Chapter 15. z/OS Debugger Terminal Interface Manager messages

Each message has a number of the form EQAY*nnnx*, where EQAY indicates that the message is a z/OS Debugger 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 z/OS Debugger attempted to fix.

**E**
An ***error*** message describes an error that z/OS Debugger 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 z/OS Debugger from continuing.

**Symbols in messages**

Many of the z/OS Debugger 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 z/OS Debugger located on the host notifies you of errors associated with debugging functions carried out by the host.

**EQAY001E User not authorized to edit data set**

## Explanation

The user does not have authority to edit the specified TEST runtime options data set.

## Programmer response

Specify a data set name to which the user ID has UPDATE access.

**EQAY002E Data set does not exist on the specified volume**

## Explanation

The TEST runtime options data set you specified does not exist on the volume specified in the Volume Serial field.

## Programmer response

Correct the Volume Serial specification.

**EQAY003E TEST run-time option data set name must be fully-qualified**

## Explanation

The TEST runtime options data set name must be preceded and followed by an apostrophe ('), to designate a fully-qualified data set name.

**EQAY004E EQAOPTS data set name must be fully-qualified**

## Explanation

The EQAOPTS data set name must be preceded and followed by an apostrophe ('), to designate a fully-qualified data set name.

**EQAY005E TEST option must be either TEST or NOTEST**

## Explanation

Please specify either TEST or NOTEST in the **Test Option** field.

**EQAY006E** **TEST level must be one of ALL, ERROR, or NONE**

## Explanation

Please specify either ALL, ERROR, or NONE in the **Test Level** field.

**EQAY008E** **Full screen mode selected, but no user ID supplied.**

## Explanation

You have selected `Full-screen mode using the z/OS DebuggerTerminal Interface Manager`, but you have not supplied a user ID. This field is required.

## Programmer response

Specify a user ID in the **User ID** field.

**EQAY009E** **No session type selected. Please select one of the options.**

## Explanation

You must select either `Full-screen mode using the z/OS Debugger Terminal Interface Manager` or `Remote debug mode`.

**EQAY010E** **Remote debug mode selected, but address not supplied.**

## Programmer response

Please supply the IP address or host name of the remote workstation where you run the remote debugger.

**EQAY011E** **Remote debug mode selected, but port not supplied.**

## Programmer response

Please supply the port number on which the remote debugger is listening.

**EQAY012I** **Security check failed for debug session for ID=*userid***

## Explanation

The specified *userid* does not have authority to debug tasks that are executed by a generic ID.

## Programmer response

Contact your security administrator to permit the specified *userid* to access the `EQADTOOL.GENERICID.`*`generic_user_id`* FACILITY.

**EQAY013I** **Generic ID debugging not enabled for ID=*userid***

## Explanation

The current task is running under a generic user ID, but the security administrator has not created a FACILITY to control debugging tasks that are run by that user ID.

## Programmer response

See the usage notes for the **DLAYDBGXRF EQAOPTS** command in the *IBM z/OS Debugger Customization Guide* for more information.

**EQAY014I** **TEST RUN-TIME OPTION DATA SET MUST HAVE DSORG OF PS**

## Explanation

The TEST run-time option data set must be a sequential data set.

## Programmer response

Specify a TEST run-time option data set that has a DSORG of PS.

**EQAY999S** **Invalid userid/password. Respecify.**

## Explanation

An invalid userid or incorrect password has been specified to z/OS Debugger Terminal Interface Manager.

## Programmer response

Respecify the userid and/or password.

**EQAY999S** **Error receiving lu name**

## Explanation

Severe internal error (VTAM 3270) in z/OS Debugger Terminal Interface Manager. Please contact your IBM representative.

**EQAY999S** **Screen dimensions could not be determined**

## Explanation

Severe internal error (VTAM 3270) in z/OS Debugger Terminal Interface Manager. Please contact your IBM representative.

**EQAY999S Logon message not available**

## Explanation

Severe internal error (VTAM 3270) in z/OS Debugger Terminal Interface Manager. Please contact your IBM representative.

**EQAY999S Session parameters inquiry error**

## Explanation

Severe internal error (VTAM 3270) in z/OS Debugger Terminal Interface Manager. Please contact your IBM representative.

**EQAY999S TPEND exit entered**

## Explanation

Severe internal error (VTAM 3270) in z/OS Debugger Terminal Interface Manager. Please contact your IBM representative.

**EQAY999S LOSTERM entered with reason code X'*xx*'**

## Explanation

Severe internal error (VTAM 3270) in z/OS Debugger Terminal Interface Manager. Please contact your IBM representative.

**EQAY999S No appl ids available.**

## Explanation

Severe internal error (VTAM 3270) in z/OS Debugger Terminal Interface Manager. Please contact your IBM representative.

**EQAY999S Session not connected**

## Explanation

Severe internal error (VTAM 3270) in z/OS Debugger Terminal Interface Manager. Please contact your IBM representative.

**EQAY999I Shutting down z/OS Debugger Terminal Interface Manager**

## Explanation

z/OS Debugger Terminal Interface Manager has been requested to shut down.

# Chapter 16. IBM z/OS Debugger 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 IBM z/OS Debugger 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 z/OS Debugger Utilities attempted to fix.

**E**
An ***error*** message describes an error that IBM z/OS Debugger 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 IBM z/OS Debugger Utilities from continuing.

**Symbols in messages**

Many of the IBM z/OS Debugger 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 IBM z/OS Debugger Utilities" in *IBM z/OS Debugger 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 IBM z/OS Debugger Utilities' in *IBM z/OS Debugger 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

z/OS Debugger Utility ends.

## User response

Follow the instructions "Customizing IBM z/OS Debugger Utilities" in *IBM z/OS Debugger 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

z/OS Debugger Utility ends.

## User response

Follow the instructions "Customizing IBM z/OS Debugger Utilities" in *IBM z/OS Debugger 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.

**EQAZ011I** **Invalid Command *cmd* for Panel *pnlId***

## Explanation

Invalid command *cmd* is entered in panel *pnlId*.

## System action

The command is not processed

## User response

Enter a valid command.

**EQAZ012I** **Invalid Member name specified. Dataset *dsnName* is not partitioned.**

## Explanation

Data set *dsnName* is a sequential file. A member name cannot be specified.

## System action

You are prompted to correct the problem.

## User response

Remove the member name or specify a partitioned data set

**EQAZ013W** **EXECIO Error for Data Set *dsnName*.**

## Explanation

I/O error in read/write data set *dsnName*.

## System action

I/O is ended.

## User response

Report the problem to the system administrator.

**EQAZ014E** **Multiple jobs detected. Only one job is allowed. By default, only the first job in the sequence will be run.**

## Explanation

Multiple jobs are found in the JCL being copied into the setup file.

## System action

Only the first job is copied.

## User response

Make sure that you select only one job when copying in the JCL.

**EQAZ015W** **Multiple programs detected. Only one program is allowed. By default, only the first program in the sequence will be selected.**

## Explanation

Multiple steps in a job are selected in the JCL that is being copied into the setup file.

### System action

Only the first step is copied.

### User response

Make sure that you select only one step in the job.

**EQAZ016W** **Invalid Concatenation *ddn*. DISP *disposition* not allowed in the middle of a concatenation.**

### Explanation

DISP *disposition* is not allowed in the middle of a concatenation.

### System action

RUN command ends.

### User response

Make sure that DISP is specified correctly.

**EQAZ017I** **Program *pgmName* ended with Return Code *rc***

### Explanation

Program *pgmName* has been executed in the foreground with a return code of *rc*.

### System action

none

### User response

Make sure that return code *rc* is what you expect.

**EQAZ018I** **Specify the allocation defaults for new Setup File *fileName***

### Explanation

A new setup file *fileName* is entered.

### System action

You are prompted for allocation defaults.

### User response

Enter allocation defaults in the next panel.

**EQAZ019W** ***cmd* allowed for the first library of concatenation only.**

### Explanation

Delete or Rename command is allowed for the first library of concatenation only.

### System action

*cmd* is not executed.

### User response

None.

**EQAZ020I** ***fileName* has been actionPerformed.**

### Explanation

File *fileName* has been actionPerformed. (such as saved)

### System action

Processing continues.

### User response

None.

**EQAZ021W** **Member not found in copy from data set *dsnName*. No data has been copied.**

### Explanation

*dsnName* is not found in a copy command processing.

### System action

Copy command ends; no data is copied.

### User response

Enter an existing data set name.

**EQAZ022W** ***dsnName* does not contain JCL or a valid setup file.**

### Explanation

*dsnName* is not a valid JCL file or a valid setup file.

### System action

Copy command ends; no data is copied.

### User response

Enter an valid file.

**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.

**EQAZ037E** **Invalid keyword *kwd* found in member *fileName.lineNo***

### Explanation

Invalid keyword *kwd* found in line *lineNo* of settings file member *fileName*.

### System action

Include statement is not processed.

### User response

Contact your system administrator. Make sure that the statement in line *lineNo* of settings file member *fileName* is a valid statement.

**EQAZ038I** **You must supply a valid job card to use batch.**

### Explanation

A request to run the application in batch is submitted but no valid job card is found.

### System action

You are prompted for job card information.

### User response

Enter job card information.

**EQAZ039W** **An invalid sequence number *seqValue* was entered.**

## Explanation

A non-numeric value *seqValue* is entered in the sequence number field.

## System action

*seqValue* is removed from the field.

## User response

Enter a numeric value.

**EQAZ040I** ***Action* started for *srcName* using *inName*.**

## Explanation

*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 *srcName* using input *inName*.

## System action

Processing continues.

## User response

None.

**EQAZ041I** ***loadlib* library: *modName* invoked.**

## Explanation

The load module *modName* in library *loadlib*, which is the data set where a compiler or assembler resides, is invoked.

## System action

Processing continues.

## User response

None.

**EQAZ042I** **CICS translator started for *fileName*.**

## Explanation

CICS translator starts to translate *fileName*.

## System action

Processing continues.

## User response

None.

**EQAZ043I** **CICS translator *transName* invoked from *complib*.**

## Explanation

CICS translator *transName* is invoked from *complib* (LINKLIST or a library).

## System action

Processing continues.

## User response

None.

**EQAZ044I** **Converter *convName* invoked from *complib*.**

## Explanation

COBOL converter *convName* is invoked from *complib* (LINKLIST or a library).

## System action

Processing continues.

## User response

None.

**EQAZ045I** **Db2 preprocessor started for *fileName*.**

## Explanation

Db2 preprocessor starts to process *fileName*.

## System action

Processing continues.

## User response

None.

**EQAZ046I** **Db2 Preprocessor *db2preName* invoked from *complib*.**

## Explanation

Db2 Preprocessor *db2preName* is invoked from *complib* (LINKLIST or a library).

## System action

Processing continues.

## User response

None.

**EQAZ047I IDILANGX started for *fileName*.**

## Explanation

IDILANGX starts to process *fileName*.

## System action

Processing continues.

## User response

None.

**EQAZ048I IDILANGX *idilName* invoked from *complib*.**

## Explanation

IDILANGX *idilName* is invoked from *complib* (LINKLIST or a library).

## System action

Processing continues.

## User response

None.

**EQAZ049W You must specify the parameter string format.**

## Explanation

You must specify the parameter string format in order to modify parameters.

## System action

You are prompted for the correct choice.

## User response

Choose a format (1 or 2).

**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.

**EQAZ055S** **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 z/OS Debugger

Utility is properly installed and configured. See *IBM z/OS Debugger Customization Guide* for details.

**EQAZ061E IMS SPOC command failed**

## Explanation

IMS SPOC command failed. Return Code = *RC*.

## System action

The action on LE runtime options is not performed.

## User response

Verify that IMSPlex ID is correctly specified. Contact your system administrator and verify that IMS V8 is installed on your system and that z/OS Debugger Utility is properly installed and configured. See *IBM z/OS Debugger Customization Guide* for details.

**EQAZ062E IMS OM security error**

## Explanation

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*.

## System action

The action on LE runtime options is not performed.

## User response

Contact your system administrator to request that your ID be authorized to use IMS QUERY LE and UPD LE commands.

**EQAZ063E Incorrect data**

## Explanation

Quote is not allowed in this field.

## System action

You are prompted for correct data.

## User response

Enter a valid value for this field.

**EQAZ064E IMS command failed**

## Explanation

IMS *command_name* command failed. Return code = *IMS_RC*, Reason code = *IMS_reason_code*.

## 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 z/OS Debugger Utility is properly installed and configured. See *IBM z/OS Debugger Customization Guide* for details.

**EQAZ065E Non-LE program cannot have load module name = EQANMDBG**

## Explanation

EQANMDBG is a reserved load module name for z/OS Debugger when debugging a non-Language Environment program.

## System action

You are prompted to correct the problem.

## User response

Enter a correct load module name.

**EQAZ066E Invalid DTU setup file**

## Explanation

The input file is not a valid DTU setup file and may be overwritten. Press **Cancel** to return.

## System action

The original content of the input file may be overwritten if processing continues.

## User response

Press **Cancel** to return and enter a valid input file or an empty file.

**EQAZ067W Not enabled for MORE**

## Explanation

This field is not enabled for additional input space.

**EQAZ068W Cursor not in a field**

## Explanation

Cursor is not in a field when MORE command is entered.

**EQAZ069E *dsnName* is not a sequential data set**

## Explanation

*dsnName* is not a sequential data set. The TEST runtime option data set must be a sequential data set.

## System action

The action on the data set is not performed.

## User response

Provide a sequential data set.

**EQAZ070E** ***GDSname* is not a valid generation data set names.**

## Explanation

*GDSname* is not a valid generation data set name. The generation base name may not exist.

## System action

*GDSname* data set is not allocated.

## User response

Provide a valid generation data set name.

**EQAZ071E** **No generation data set name for *GDSname*.**

## Explanation

*GDSname* is not a generation data set name. The last qualifier is not in G*nnnn*V*nn* format.

## System action

*GDSname* data set is not allocated.

## User response

Provide a valid generation data set name.

**EQAZ072E** **Generation number exceeds 9999 for *GDSname*.**

## Explanation

The generation number for *GDSname* exceeds the maximum number allowed.

## System action

*GDSname* data set is not allocated.

## User response

Provide a valid generation data set name.

**EQAZ076W** **CHARS '*charstring*' was not found on any rows.**

## Explanation

Search for '*charstring*' was not successful.

## Programmer response

Provide a different search argument.

**EQAZ077I** **Search for CHARS '*charstring*' was successful.**

## Explanation

The row that contains '*charstring*' is positioned as the top row.

**EQAZ078W** **CHARS '*charstring*' was not found. Enter TOP command and press FINDNEXT key to continue from row 1.**

## Explanation

Search for '*charstring*' was not found from the cursor position to the bottom of the list.

**EQAZ079W** **The FINDNEXT command works only after a FIND command character string is entered.**

## Explanation

`FINDNEXT` command requires a previously entered search argument.

## Programmer response

Use `FIND` command with a search argument.

**EQAZ080W** **Data set *datasetname* is not found.**

## Explanation

Data set *datasetname* does not exist or is not cataloged.

## Programmer response

Provide a valid data set name.

**EQAZ081I** **Multiple actions are present. Only the first one is processed.**

## Explanation

More than one step has an action specified. The utility processes the first one and ignores the rest.

## Programmer response

Specify one action at a time.

The window that displays this message might display the name of the window (EQAPMGP) and the instruction to press Enter to continue. Press Enter to close the message window.

**EQAZ082E** **Allocate data set *datasetname* failed. Return code is *retcode*.**

## Explanation

The user exit data set *datasetname* cannot be created. User exit invocation method will not work.

## Programmer response

Consult with system administrator.

**EQAZ083W** **Allocate data set *datasetname* failed. Return code is *retcode*.**

## Explanation

*datasetname* data set cannot be created. The updated JCL is not saved.

## Programmer response

Consult with system administrator.

**EQAZ084I** **A new data set *datasetname* is created.**

## Explanation

*datasetname* data set is created. The data set is a user exit data set or contains the copy of the updated JCL.

**EQAZ085W** **CEEOPTS DD statement invocation method cannot be used in *steptype* of step (number: *stepno*).**

## Explanation

IMS batch does not process the CEEOPTS DD statement.

## Programmer response

Choose one of the user exit invocation methods in the user settings panel.

**EQAZ100W** **A member name must be specified when the output: *datasetname* is a partitioned data set.**

## Explanation

The output data set *datasetname* is a partitioned data set and a member name must be specified.

## Programmer response

Add a member name to the data set or specify a different data set.

**EQAZ101W** **A member name cannot be specified when the output: *datasetname* is a sequential data set.**

## Explanation

The output data set, *datasetname* is a sequential data set and a member name cannot be specified.

## Programmer response

Remove the member name or specify a different data set.

**EQAZ102W** **A valid job card is required to generate proper JCL.**

## Explanation

There is no job card setting or the job card is not valid.

## Programmer response

Enter job card information in the User Settings panel.

**EQAZ103W** **No IMS ID defined in the system.**

## Explanation

No IMS ID is defined in the table in SEQATLIB data set.

## Programmer response

Check EQAZDSYS and EQAZDUSR members for IMS system ID variable (yb2iidn) definition.

**EQAZ104W** **IMS ID does not match the ones in the system.**

## Explanation

IMS ID is not defined in the table in SEQATLIB data set.

## Programmer response

Choose one from the list shown on the panel.

**EQAZ105W** **JCL not defined for IMS system: *imsid*.**

## Explanation

The base JCL is not found for IMS system '*imsid*'.

## Programmer response

Check EQAZDSYS and EQAZDUSR members for JCL variable definition.

**EQAZ109W** **Invalid substitution variable found: &*variablename***

## Explanation

Invalid substitution variable is found in z/OS Debugger user exit data set naming pattern.

## Programmer response

Check EQAZDSYS and EQAZDUSR members z/OS Debugger user exit data set naming pattern variable (yb2dtnmp) definition.

**EQAZ110W** **Default base JCL cannot be edited.**

## Explanation

The base JCL contains the common components for a IMS system and cannot be edited.

**EQAZ111W** **JCL parser failed on JCL: *jclname***

## Explanation

z/OS Debugger JCL parser failed on parsing *jclname* JCL. The JCL could be the base JCL or user provided JCL that contains additional application DD statements.

## Programmer response

Check EQAZDSYS and EQAZDUSR members z/OS Debugger user exit data set naming pattern variable (yb2dtnmp) definition.

**EQAZ112W** ***commandname* command is invalid for IMS BTS debugging.**

## Explanation

The *commandname* command is not valid in panel EQAPFORB for IMS BTS debugging.

## Programmer response

Do not use the command.

**EQAZ114W** **User exit data set naming pattern is required for invocation method: *invmethod*.**

## Explanation

For user exit invocation method ('E' or 'A'), the user exit data set naming pattern must be present. The utility needs it to open and update the data set.

## Programmer response

Provide the naming pattern.

**EQAZ115W** **Data set *datasetname* is not found.**

## Explanation

Data set *datasetname* in a file list does not exist or is not cataloged.

## Programmer response

Provide a valid data set name.

**EQAZ116W** **Member in data set *datasetname* is not found.**

## Explanation

Member does not exist in data set *datasetname*.

## Programmer response

Provide a valid member.

**EQAZ117W** **Data set *datasetname* is not valid.**

## Explanation

Data set *datasetname* is not a valid data set name.

## Programmer response

Provide a valid data set name.

**EQAZ118W** **Line command *command* is not valid.**

## Explanation

Line command *command* is not a valid command.

## Programmer response

Provide a valid line command.

**EQAZ119W** **String has mismatched or uneven number of quotation marks: *string*.**

## Explanation

The string is enclosed in quotation marks. Either the enclosing quotation marks are mismatched, or there are quotation marks within the string without an escape sequence.

## User response

Correct the problem.

**EQAZ120W** **String containing blanks should be enclosed in quotation marks: *string*.**

## Explanation

The string contains blanks. It should be enclosed in quotation marks.

## User response

Correct the problem.

**EQAZ121W** **Token is not supported in naming pattern: *namingpattern*.**

## Explanation

The naming pattern contains a token that is not supported.

## User response

Correct the problem.

**EQAZ122E** **Incomplete XML tag: *xmltag* in data set: *datasetname*.**

## Explanation

Incomplete XML tag is found in the cross reference table data set.

## User response

Consult with system administrator.

**EQAZ123E** **Transaction ID, *transctionid*, exists in cross reference table.**

## Explanation

The transaction ID used in the CREATE command already exists in the cross reference table.

## User response

Use a different transaction ID or use the UPDATE command.

**EQAZ124E** **Transaction ID, *transctionid*, does not exist in cross reference table.**

## Explanation

The transaction ID used in the UPDATE command does not exist in the cross reference table.

## User response

Use a different transaction ID.

**EQAZ125E** **The command, *command*, must have one operand.**

## Explanation

The command requires one operand: transaction ID.

## User response

Provide a transaction ID.

**EQAZ126W** **The maximum number of entries allowed is: *maxentry***

## Explanation

The maximum number of entries allowed in the cross reference table is reached.

## User response

Consult with system administrator.

**EQAZ127W** **The maximum number of generic IDs allowed is: *maxentry***

## Explanation

The maximum number of generic IDs allowed in the cross reference table is reached.

## User response

Consult with system administrator.

**EQAZ128W** **One or more table entries are expired. They are deleted from the table when Exit.**

## Explanation

One or more table entries are expired, exceeding the active period. They will be deleted when you exit the panel.

## User response

Use the KEEP command to change the entry timestamp if you want to keep the entry in the table.

**EQAZ129E Cross reference data set name is not defined in table library (EQAZDSYS).**

## Explanation

The data set name of the cross reference table is not found in the table library (EQAZDSYS).

## User response

Consult with system administrator.

**EQAZ130E Generic ID cannot be used in the cross reference table entry.**

## Explanation

Generic ID cannot be used in the CREATE or UPDATE command.

## User response

Use your own user ID.

**EQAZ131E Transaction ID, *transactionid*, exceeds the maximum length of 8.**

## Explanation

The transaction ID specified in the CREATE, UPDATE, or DELETE command must be eight characters or less in length.

## User response

Use a valid transaction ID.

**EQAZ132E Only one ID is allowed.**

## Explanation

You can use only one user ID in the CREATE or UPDATE command.

## User response

Enter only one ID in the **User ID** field.

**EQAZ133E The command, *command*, has only one operand.**

## Explanation

The CREATE, UPDATE, or DELETE command has only one operand: transaction ID.

## User response

Use only one transaction ID in the command.

**EQAZ134E Line command is not allowed on row, **TOP OF DATA**.**

## Explanation

The row, **TOP OF DATA**, does not accept line command.

## User response

Do not use line command on this row.

**EQAZ135E Transaction ID, *transactionid*, contains non-alphanumeric character.**

## Explanation

Non-alphanumeric characters are not allowed in transaction ID.

## User response

Use only alphanumeric characters.

**EQAZ136I *datasetname* data set has been saved.**

## Explanation

The changes in the data set have been saved.

## User response

None.

**EQAZ137E Debug session start and stop message data set name is not defined in table library (EQAZDSYS).**

## Explanation

The data set name of the start and stop message file is not found in the table library (EQAZDSYS).

## User response

Consult with system administrator.

**EQAZ138E** **Duplicate sort order, *sortorder*, is specified in column: *columnname*.**

## Explanation

A sort order in a column must be unique among the sort orders specified.

## User response

Use a unique sort order.

**EQAZ139E** **No source/listing is available.**

## Explanation

No SYSDEBUG file is associated with the code coverage observation. The SYSDEBUG file is needed for source or listing extraction.

## System action

No source or listing is shown.

## User response

Make sure that the SYSDEBUG file from the application compilation exists.

**EQAZ140E** **Code coverage observation data set name is not defined in table library (EQAZDSYS).**

## Explanation

A code coverage observation data set name is not defined in table library (EQAZDSYS).

## System action

Code coverage function is stopped.

## User response

Consult with system administrator.

**EQAZ141E** **More than *max xmltag* tags in data set: *datasetname*.**

## Explanation

The number of *xmltag* tags exceeds the maximum value.

## System action

Processing of XML tags in the data set *datasetname* is stopped.

## User response

Make sure that the number of *xmltag* tags that the data set has does not exceed the maximum number.

**EQAZ142E** **Invalid value: *xmltagvalue*, in tag: *xmltag*, in data set: *datasetname***

## Explanation

The value of *xmltag* tag is not valid. It is not a valid attribute name or a numerical value.

## System action

Processing of XML tags in the data set *datasetname* is stopped.

## User response

Provide a valid value for the *xmltagvalue* .

**EQAZ143E** **Error processing file: *filename***

## Explanation

A problem occurred during the processing of *filename* data set, like data set allocation, reading from or writing to the data set.

## System action

Processing of the data set is stopped.

## User response

Make sure that the data set exists and you have proper access authority.

**EQAZ144I** **Code coverage observation extraction is completed.**

## Explanation

The extraction operation is completed.

## System action

Processing continues.

## User response

None.

**EQAZ145E** **Invalid XML tag: *xmltag*, in DS or DD name: *datasetname***

## Explanation

A problem occurred with *xmltag* tag, for example, no tag value or no corresponding end tag.

## System action

Processing of XML tags in the data set *datasetname* is stopped.

## User response

Correct the problem in the *xmltag* tag.

**EQAZ146I** **Code coverage report generation is completed.**

## Explanation

The report generation operation is completed.

## System action

Processing continues.

## User response

None.

**EQAZ147E** **The column start: *column number* is greater than column end: *column number*.**

## Explanation

The source marker has the column start number greater than the column end number.

## System action

Panel EQAPCCS2 is displayed with the error message.

## User response

Enter a correct column number for the source marker.

**EQAZ148E** **The column start or column end: column number is greater than 80.**

## Explanation

The source marker has a column start or end number greater than 80.

## System action

Panel EQAPCCS2 is displayed with the error message.

## User response

Enter a correct column number for the source marker.

**EQAZ149E** **The string: *stringValue* contains non-hexadecimal characters.**

## Explanation

The source marker has invalid characters in the hex adecimal string.

## System action

Panel EQAPCCS2 is displayed with the error message.

## User response

Enter a correct hex adecimal string value.

**EQAZ150E** **The `SECTION` markers are not in pairs. `SECTIONBEGIN` without `SECTIONEND` or vice versa.**

## Explanation

The `SECTION` source markers are not in pairs.

## System action

Panel EQAPCCS2 is displayed with the error message.

## User response

Make sure that the `SECTION` markers are in pairs.

**EQAZ151E** **The source marker type: *markertype* is not valid.**

## Explanation

Valid source marker types are `SINGLE`, `SECTIONBEGIN`, and `SECTIONEND`.

## System action

Panel EQAPCCS2 is displayed with the error message.

## User response

Enter a correct source marker type.

**EQAZ152E** **The source marker selection: *markertype* is not valid.**

## Explanation

Valid source marker selections are `INCLUDE` and `EXCLUDE`.

### System action

Panel EQAPCCS2 is displayed with the error message.

### User response

Enter a correct source marker selection.

**EQAZ153E** **Code coverage report XML tag *xmltag* is found in DS name or DD name: *datasetname*.**

### Explanation

The report XML data set is specified as input to where the code coverage observation XML data set is expected.

### System action

Processing of XML tags in the data set *datasetname* is stopped.

### User response

Enter a correct data set name.

**EQAZ154E** **No value is entered in required field: *fieldname***

### Explanation

The named field of a source marker is a required field. You must provide a value.

### User response

Provide a value for the named field.

**EQAZ155W** **No observation meets selection criteria.**

### Explanation

The observation extraction process did not find any observation in the input code coverage observation data set that meets the selection criteria.

### User response

Check the selection criteria using the E.3 option: Observation selection criteria to ensure that selection criteria is specified correctly, and then retry the extraction process.

**EQAZ156W** **No observation found in file: *filename***

### Explanation

The report generation process did not find any observation in the input code coverage extracted observation data set, *filename*.

### User response

Check the input data set to ensure it is not empty, and then retry the report generation process.

**EQAZ157E** **Cannot SAVE a template file. Use SAVE AS to save to a private DTSU library.**

### Explanation

You used the SAVE command to save changes to a message region template. The templates are read-only from this panel.

### System action

Panel EQAPMPRT is displayed with the error message.

**User response:**
To save your changes to a private library, use the SAVE AS command.

**EQAZ158E** **The CEEOPTS DD is present but references a data set. Debugging may be inhibited.**

### Explanation

Debugging is enabled by modifying an inline CEEOPTS DD to include the TEST option. The chosen region specifies a data set for the CEEOPTS DD; therefore, z/OS Debugger was not able to update CEEOPTS to include a TEST parameter.

### System action

Panel EQAPMPRT is displayed with the error message.

### User response

Contact a message region template administrator to convert the contents of the data set specified on the CEEOPTS DD to an inline DD in the template.

**EQAZ159E** **The CEEOPTS DD was not present. A default in-line CEEOPTS has been added.**

### Explanation

In order to specify a TEST runtime parameter, z/OS Debugger has created an inline CEEOPTS DD for this message region template.

## System action

Panel EQAPMPRS is displayed with the error message.

## User response

None required.

**EQAZ160E** **The specified message region template data set does not exist. Please supply a valid data set name.**

## Explanation

The data set specified for Template Data Set does not exist.

## System action

Panel EQAPMPRS is displayed with the error message.

## User response

Supply a valid message region template data set. If you are unsure, contact a message region template administrator for a list of valid data set names.

**EQAZ161E** **The data set you have chosen is already used to store message region templates that have been pre-defined by an administrator. Please select a different data set for storing your private copy of the message region template.**

## Explanation

You have used the SAVE AS command to save a private copy of a message region template. However, you have chosen a data set which is already used to store shared message region templates.

## System action

Panel EQAPMPRS is displayed with the error message.

## User response

Specify a data set name that is not used to contain shared message region templates.

**EQAZ162E** **The data set you have chosen is not a valid template data set. Please enter the name of a data set that contains templates that have been pre-defined by an administrator. You may also place / in the first line of the member list to select a private message region template.**

## Explanation

The data set that you specified as the Template Data Set does not contain pre-defined message region templates.

## System action

Panel EQAPMPRS is displayed with the error message.

## User response

Supply a data set name that contains pre-defined message region templates. If you wish to select a data set with private message region templates, place a forward slash (/) next to the first line of the member list.

**EQAZ163E** **The data set you have chosen is not a valid template data set. The data set is not partitioned, or it has members that are not pre-defined message region templates.**

## Explanation

The data set that you specified as the Template Data Set does not contain pre-defined message region templates.

## System action

Panel EQAPMPRS is displayed with the error message.

## User response

Supply a data set name that contains pre-defined message region templates.

**EQAZ164E** **Cross reference table data set *datasetname* is not available. Enqueue return code is *RC***

## Explanation

The option, IMS Transaction and User ID Cross Reference Table, allows you to update the data set. You must have exclusive control of the data set. Currently, another user is using the option.

## System action

None

## User response

Try to access the option at a later time.

To determine who is holding the cross reference table data set, issue TSO ISRDDN from the ISPF command line, type ENQ and press enter. Fill in SPFEDIT as the Major name prefix, clear the other input fields and press enter. You should then see who has an enqueue on that data set held by another user using the same function or ISPF editor.

**EQAZ165E** **Debug profile data set *datasetname* is not available. Enqueue return code is *RC*.**

## Explanation

You can update the data set by using this option. You must have exclusive control of the data set. Currently, another user is using the option.

## System action

None.

## User response

Try to access the option at a later time.

To determine who is holding the cross reference table data set, issue TSO ISRDDN from the ISPF command line, type ENQ and press enter. Fill in SPFEDIT as the Major name prefix, clear the other input fields and press enter. You should then see who has an enqueue on that data set held by another user using the same function or ISPF editor.

**EQAZ166I** **Breakpoints data set: *datasetname* is updated.**

## Explanation

The breakpoints data set or the debug commands data set is updated successfully.

## System action

Processing continues.

## User response

None.

**EQAZ167E** **The chosen transaction is not assigned to a class which has an active message region. Therefore, z/OS Debugger cannot use an active region to build the debugging region.**

## Explanation

The IMS Transaction Isolation facility uses the current execution environment for the selected transaction to clone a new, private message region. z/OS Debugger cannot find an active message region that serves the assigned class of the selected transaction. Therefore, z/OS Debugger cannot start your private message region.

## System action

The private message region is not started.

## User response

Ensure that the transaction is assigned to a class with an active message region.

**EQAZ168E** **The selection that was entered is invalid. Valid selections are `(R)egister`, `(D)eregister`, `(S)tart`, `Sto(P)`, and `(E)dit`.**

## Explanation

An invalid selection character was entered.

## System action

None.

## User response

Use one of the supported selection characters.

**EQAZ169E** **The identifier that was entered for the IMS system is invalid. The IMS system is not set up for transaction isolation or the identifier does not correspond to a valid IMS subsystem ID.**

## Explanation

The IMS Transaction Isolation facility was not set up for the IMS system that was entered in the **IMS system** field.

## System action

None.

## User response

Check that the IMS system name was entered correctly. If the string that was entered is a valid IMS system, check with the IMS administrator whether the

IMS Transaction Isolation facility is set up for that IMS system.

**EQAZ170E The number of transactions exceeded the maximum number. To register for this transaction, de-register from one of the other transactions in the list.**

## Explanation

The IMS system administrator placed a limit on the number of transactions. The selection that was made exceeded that limit.

## System action

The selected transaction is not registered for debug.

## User response

De-register from another transaction that was registered. You can also request that the IMS system administrator increase the limit of transactions.

**EQAZ171E There are no free IMS region classes to reserve. Please ask the administrator to add more classes for IMS Transaction Isolation or wait until a class becomes available.**

## Explanation

A set of message classes were reserved for z/OS Debugger to use when the private message regions were created. All of those classes are currently reserved by other users.

## System action

The transaction that was selected is not registered for debug.

## User response

You can contact one of the users who registered to debug in the list of transactions to request that they de-register from their transactions. To view the list, change the **Filter** selection to `Display full transaction list`. You can also request that the IMS system administrator add more classes to those that were reserved for z/OS Debugger.

**EQAZ172E A private message region was started, but no delay debug options data set was allocated. The region was started with a hardcoded TEST parameter of TEST(ALL,*,PROMPT,VTAM%userid:*).**

## Explanation

z/OS Debugger attempts to start your private message region in Delay Debug mode, so that you can use an extra level of pattern-matching when the transaction is routed to your private message region. However, to use the Delay Debug mode, you must have a data set allocated that conforms to your site's Delay Debug options data set naming pattern. Because the data set does not exist, z/OS Debugger started your region with a hardcoded TEST parameter.

## System action

The private message region is started with the parameter **TEST**(**ALL**,*****,**PROMPT**,**VTAM%userid:***).

## User response

You can use Delay Debug mode in your private message region by using IBM z/OS Debugger Utilities option B, `Delay Debug Profile`, to create the data set.

**EQAZ173E This function requires z/OS Debugger SVC Version 18. The installed z/OS Debugger SVC is back-level.**

## Explanation

The IMS Transaction Isolation facility requires at least Version 18 of the z/OS Debugger SVC.

## System action

The requested action is not performed.

## User response

Install the proper version of the z/OS Debugger SVC.

**EQAZ174I z/OS Debugger commands data set: *datasetname* is updated.**

## Explanation

The z/OS Debugger commands data set is updated with z/OS Debugger commands created from breakpoint definitions.

## System action

Processing continues.

## User response

None.

**EQAZ175E The specified data set does not exist. Make sure the data set name is spelled correctly.**

## Explanation

The specified data set does not exist and a decision is made not to create one. Make sure the data set name is spelled correctly.

## System action

No z/OS Debugger command is created and saved in the specified data set.

## User response

Check the spelling of the data set name and try it again.

**EQAZ176I The breakpoint table is empty. No z/OS Debugger command is created.**

## Explanation

There is no breakpoint definition in the table. No z/OS Debugger commands can be created.

## System action

Processing continues.

## User response

None.

**EQAZ177E A partitioned data set for *output-ds* was specified, but no member name was specified.**

## Explanation

The data set name that was entered for the output data set is a partitioned data set, but no member name was specified.

## System action

None.

## User response

Specify a member name for the output data set, or specify the name of a sequential data set.

**EQAZ178E The character char is not a valid selection. Use the / character to select classes for debug.**

## Explanation

The selection character char that was entered for a message class is not valid.

## System action

The selected class is not reserved for debug users.

## User response

Use the **/** character to select message classes to reserve for z/OS Debugger.

**EQAZ179I The IMS ID list is empty. Consult with your IMS administrator.**

## Explanation

No IMS system ID is available.

## System action

No transaction is displayed.

## User response

Consult with your IMS system administrator whether the IMS Transaction Isolation facility is enabled.

**EQAZ180I No transaction matches the selection criteria. Check your transaction filter setting.**

## Explanation

No transaction matches the selection criteria.

## System action

No transaction is displayed.

## User response

Check your transaction filter setting:

- Transaction name filter: ensure that the name pattern is spelled correctly.
- User ID filter: confirm whether you registered any transaction for debug.
- All transactions: check with your IMS system administrator whether any transaction is defined in the selected IMS system.

**EQAZ181I** **More transactions match the selection criteria. Check your transaction filter setting.**

## Explanation

More transactions match the selection criteria than the maximum number of transactions that are allowed for display.

## System action

The maximum number of transactions are displayed.

## User response

Change the maximum number of transactions that are allowed for display to a larger number or 0 (no limit).

**EQAZ182E** **The private message region is busy performing a debug session. It cannot be stopped at this time. Please try again later.**

## Explanation

An attempt was made to stop the private message region. However, the region cannot be stopped because a debug session is running in the region.

## System action

The private message region is not stopped.

## User response

Wait until debugging is completed and try again.

**EQAZ183E** **You are attempting to deregister the last transaction, which will free up the class you have been assigned. However, your private region is still started. Please STOP your region and try again.**

## Explanation

An IMS message class is reserved for use if any transaction is registered for debugging using IMS Transaction Isolation facility. The request made would deregister the last transaction and release the message class. However, the private message region is still started. When another user who registers for a transaction receives the released message class and starts a private message region serving the class, unpredictable results can happen.

## System action

None.

## User response

Stop the private message region and try again.

**EQAZ184E** **You are attempting to start a private message region for the selected transaction. However a private region is already started for your assigned class. You will be registered for the transaction, but no region will be started.**

## Explanation

An attempt was made to start a private message region for the selected transaction. However, no private region is started because the region is already started for the assigned class.

## System action

The transaction is registered for debugging, but no region will be started.

## User response

No action is required.

**EQAZ185E** **A private message region *region-name* was started, but the job may not be running. Double check if *region-name* is running. If not, use (P) Stop Region to stop the private region and then use (S) Start Region to start it again.**

## Explanation

A private region was started for IMS Transaction Isolation. However, the region is not running now and IMS Transaction Isolation might not know that. The private region was probably stopped from an IMS terminal, or abended because of the debugger or user code.

## System action

None.

## User response

Manually stop the private region in IMS Transaction Isolation by entering P. After the status of the private region is changed to Stopped, restart the private region by entering S.

**EQAZ186E** **IMS Transaction Isolation Facility attempted to execute a type-2 IMS command, but cannot register with the Structured Call Interface(SCI). Check with your systems programmer to ensure that the IMSplex has been properly identified.**

## Explanation

IMS Transaction Isolation Facility uses type-2 IMS commands in cases where the type-1 equivalents are disallowed. To issue the type-2 command, IMS Transaction Isolation must register with the Structured Call Interface (SCI) for the appropriate IMSplex. The registration for the IMSplex identified for the selected IMS system has failed.

## System action

The list of transactions failed to be retrieved.

## User response

Work with your systems programmer to ensure that the IMSplex is properly identified for the IMS system you have chosen.

**EQAZ187E** **IMS Transaction Isolation Facility attempted to execute a type-2 IMS command, but the Common Service Layer (CSL) returned an error. Ask your systems programmer to check the Operations Manager (OM) job log for error messages.**

## Explanation

An error was encountered after a type-2 IMS command was issued.

## System action

The list of transactions failed to be retrieved.

## User response

Ask your systems programmer to examine the Operations Manager (OM) component job log for error messages.

**EQAZ188E** **IMS Transaction Isolation Facility attempted to execute the QUERY type-2 IMS command, but your user ID is not authorized to issue the command. Ask your systems programmer to grant you the proper authority.**

## Explanation

IMS Transaction Isolation Facility attempted the QUERY type-2 IMS command, but authorization failed for your user ID.

## System action

The list of transactions failed to be retrieved.

## User response

Ask your systems programmer to authorize your user ID to the IMS.CSL*implx*.QRY.TRAN resource in the OPERCMDS class, where *implx* is the IMSplex name.

# Appendix A. z/OS Debugger commands supported in Debug Tool compatibility mode

**Note:** This section is applicable only to Debug Tool compatibility mode, a subset of the remote debug mode.

You can use some z/OS Debugger commands in Debug Tool compatibility mode through the following methods:

- Enter these commands through the **Debug Engine Command** field or the **Debug Console Commands** window of the remote debugger.
- When you add a breakpoint through the remote debugger, specify these commands in the **Action** field, which is in the **Optional Parameters** section of the **Add a Breakpoint** task.
- Use them in a commands or preferences file.

Using any of these methods, you can use the following commands in remote debug mode:

- “LOAD command” on page 166
- “LOADDEBUGDATA command” on page 167 (for assembler only)
- “NAMES DISPLAY command” on page 179
- “NAMES EXCLUDE command” on page 179
- “NAMES INCLUDE command” on page 180
- QUERY ASSEMBLER, which is described in “QUERY command” on page 194
- QUERY AUTOMONITOR, which is described in “QUERY command” on page 194
- QUERY BROWSE MODE, which is described in “QUERY command” on page 194
- QUERY CURRENT VIEW, which is described in “QUERY command” on page 194
- QUERY DEFAULT DBG, which is described in “QUERY command” on page 194
- QUERY DEFAULT LISTINGS, which is described in “QUERY command” on page 194
- QUERY DEFAULT MDBG, which is described in “QUERY command” on page 194
- QUERY DEFAULT VIEW, which is described in “QUERY command” on page 194
- QUERY DISASSEMBLY, which is described in “QUERY command” on page 194
- QUERY DYNDEBUG, which is described in “QUERY command” on page 194
- QUERY EQAOPTS, which is described in “QUERY command” on page 194
- QUERY EXPLICITDEBUG, which is described in “QUERY command” on page 194
- QUERY IGNORELINK, which is described in “QUERY command” on page 194
- QUERY INTERCEPT, which is described in “QUERY command” on page 194
- QUERY LDD, which is described in “QUERY command” on page 194
- QUERY LIST BY SUBSCRIPT, which is described in “QUERY command” on page 194
- QUERY LOCATION, which is described in “QUERY command” on page 194
- QUERY LOG, which is described in “QUERY command” on page 194
- QUERY REWRITE, which is described in “QUERY command” on page 194
- QUERY WARNING, which is described in “QUERY command” on page 194
- “QUIT command” on page 199
- QUIT ABEND, which is described in “QUIT command” on page 199
- QUIT DEBUG, which is described in “QUIT command” on page 199
- “QQUIT command” on page 200
- “SET ASSEMBLER ON/OFF command” on page 210
- “SET ASSEMBLER STEPOVER command” on page 211
- “SET AUTOMONITOR command” on page 212
- “SET DEFAULT DBG command” on page 219
- “SET DEFAULT LISTINGS command” on page 220
- “SET DEFAULT MDBG command” on page 221
- “SET DEFAULT VIEW command” on page 223
- “SET DISASSEMBLY command” on page 224
- “SET DYNDEBUG command” on page 225
- “SET EXPLICITDEBUG command” on page 228
- “SET IGNORELINK command” on page 231
- “SET INTERCEPT command (COBOL, remote debug mode)” on page 234
- “SET LDD command” on page 235
- “SET LIST BY SUBSCRIPT command (COBOL)” on page 236

- SET LOG OFF, which is described in “SET LOG command” on page 239
- SET LOG ON, which is described in “SET LOG command” on page 239
- SET QUALIFY CU, which is described in “SET QUALIFY command” on page 250
- SET QUALIFY LOAD, which is described in “SET QUALIFY command” on page 250
- “SET REWRITE command (remote debug mode)” on page 255
- “SET WARNING command (C, C++, COBOL, and PL/I)” on page 263

You can use the following commands in remote debug mode only in the **Action** field, which is in the **Optional Parameters** section of the **Add a Breakpoint** task:

- “JUMPTO command” on page 136
- “RUNTO command” on page 203

You can use the following commands in remote debug mode in the **Action** field, which is in the **Optional Parameters** section of the **Add a Breakpoint** task, and in the Startup Commands section of RD/z Debug Configuration:

- “GO command” on page 124
- “STEP command” on page 269

Through the **Debug Engine Command** field or the **Debug Console Commands** window, you can view a list of z/OS Debugger commands supported in remote debug mode by doing one of the following tasks:

- Press Ctrl+SPACE BAR.
- Type in the first few letters of a command name. Press Ctrl+SPACE BAR. A list of z/OS Debugger commands that begin with those same letters is displayed.

z/OS Debugger supports Remote Playback through the Playback Toolbar in the Debug View.

You can display z/OS Debugger variables in remote debug mode. z/OS Debugger reserves variables for its own information. To distinguish z/OS Debugger variables from program variables, the names of z/OS Debugger variables begin with a percent sign. You can access z/OS Debugger variables when you test programs in supported languages. To display the value of a z/OS Debugger variable, add it as a monitor expression in the Monitor view. For more information about z/OS Debugger variables, see Chapter 8, “z/OS Debugger variables,” on page 333.

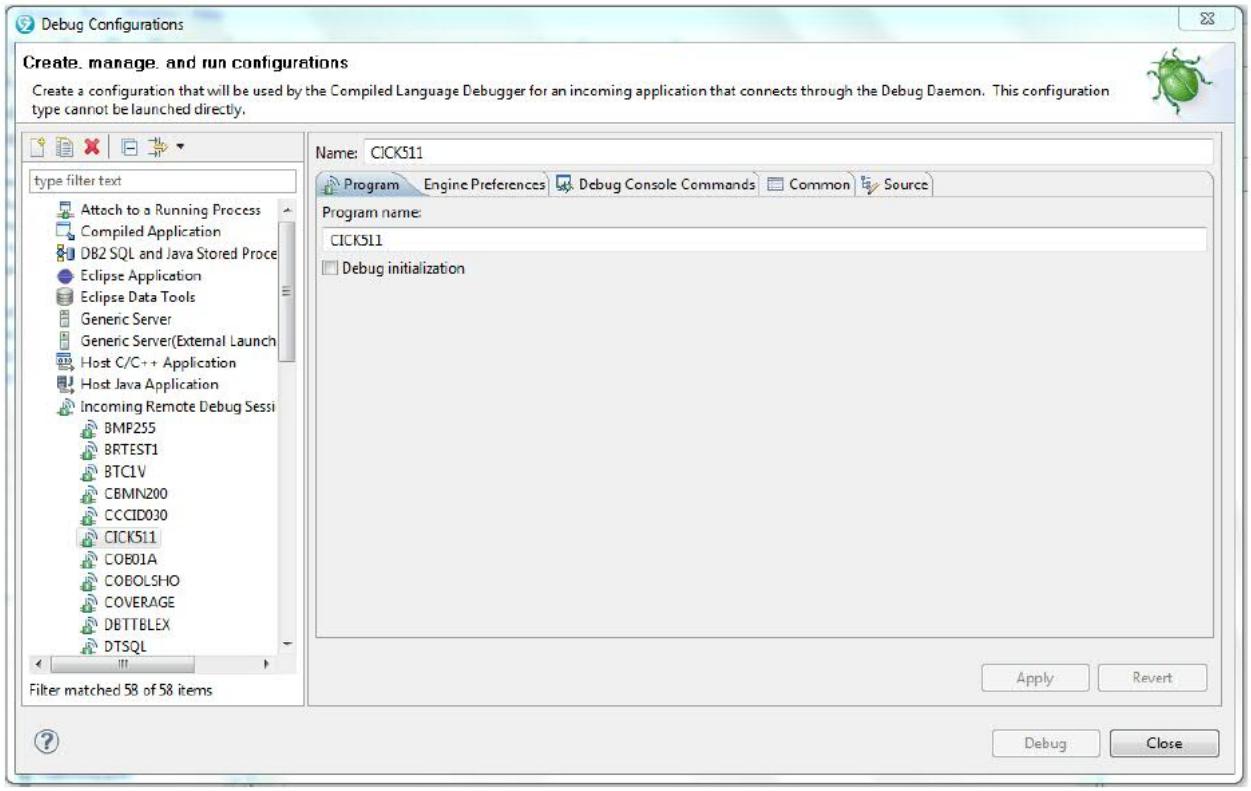
**Example**

If you want to know the value of the PSW when you debug an assembler program, add the following z/OS Debugger variable to the monitor view:

%PSW

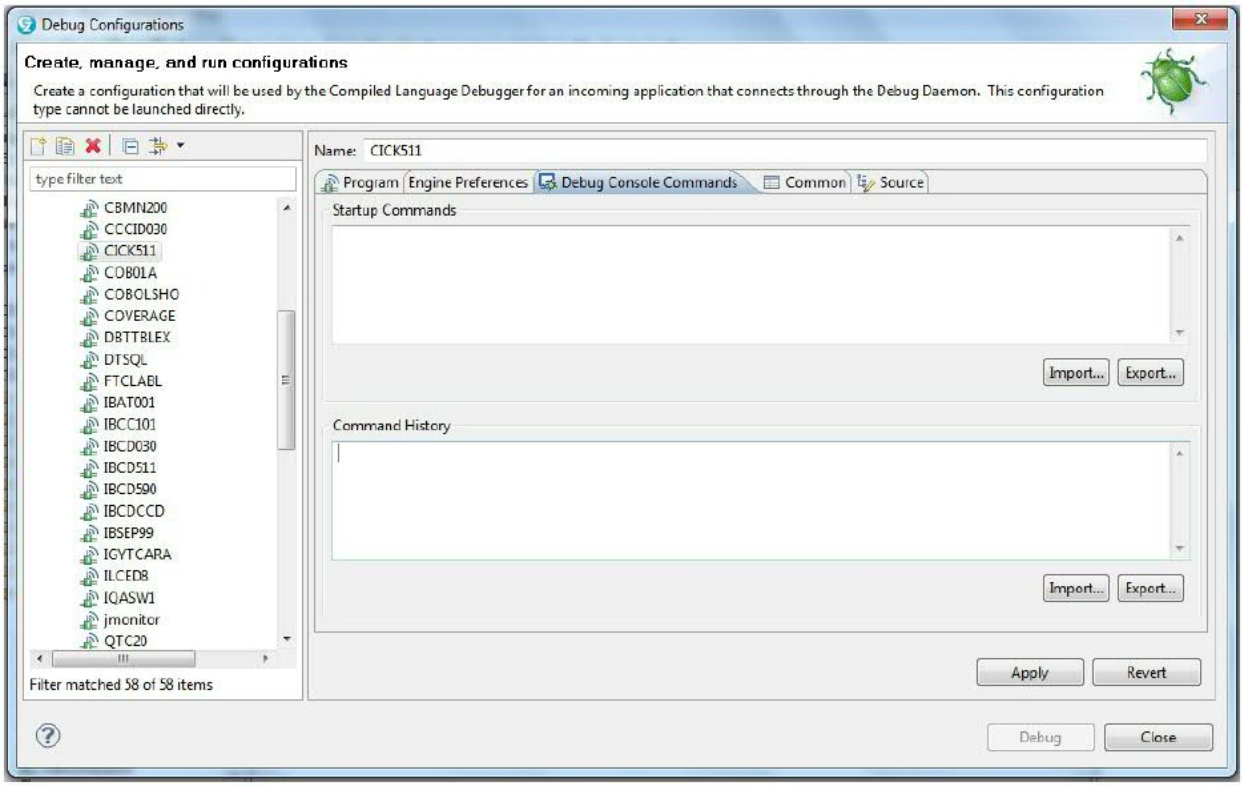
## Specifying z/OS Debugger commands in launch configuration

Access the Debug Configuration by selecting **RUN -> Debug Configurations**. After selecting this option you are presented with the following view:




On the left side, you will see a list of programs under Incoming Remote Debug Session. Pick the program that you are interested in debugging and for which you want to add z/OS Debugger commands. For this example CICK511 is selected.

Select the Debug Console Commands tab.



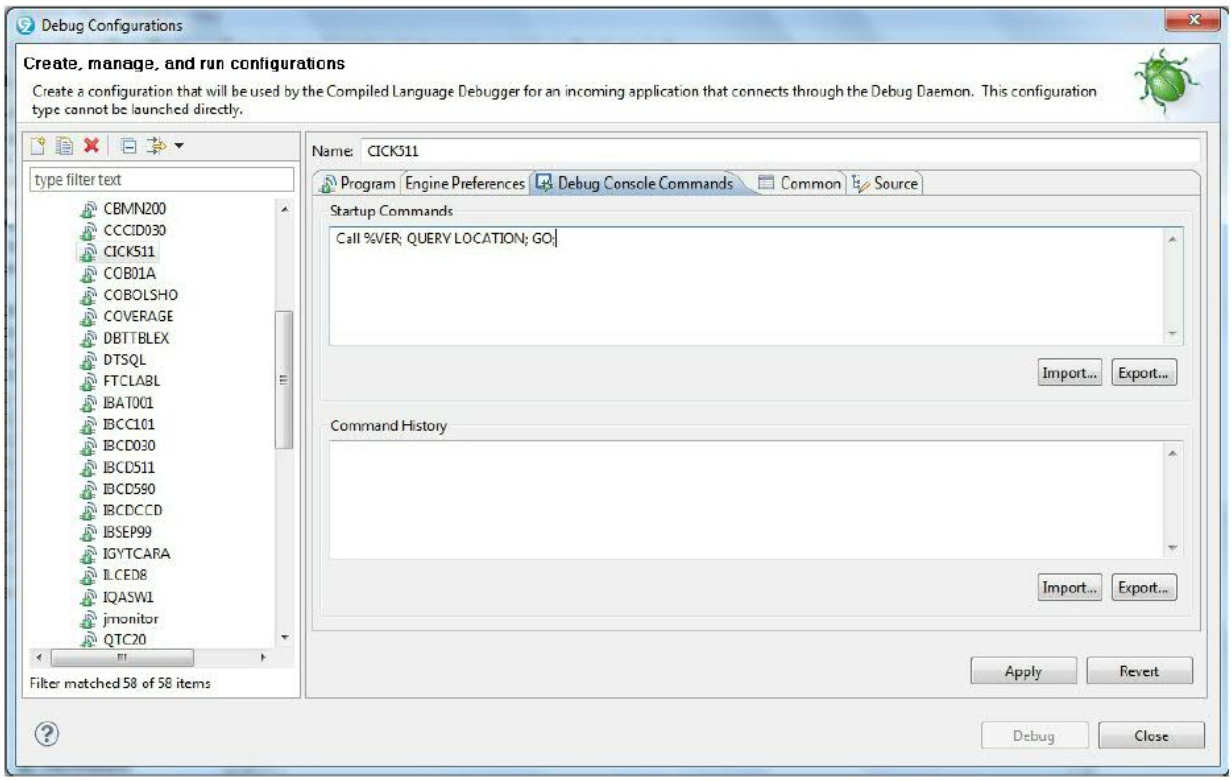
The top section labeled as Startup Commands is where you specify the z/OS Debugger commands that are to be executed at the start of the z/OS Debugger session.

All commands that you can execute in the Debug Console Commands can also be specified as Startup commands. The only two commands that you can only specify as Startup commands are STEP and GO.

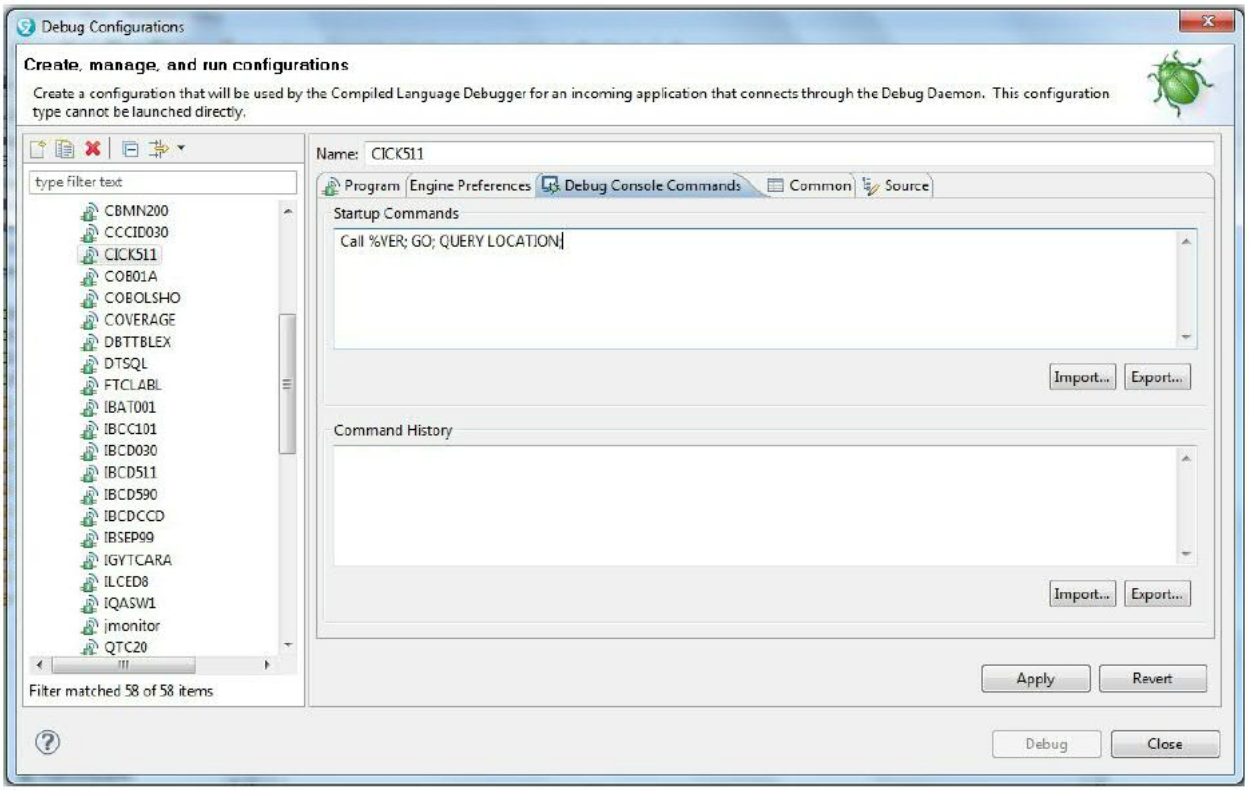
You can specify more than one command separated by a semicolon (;). If you specify STEP or GO, all the commands that follow are not executed, and z/OS Debugger returns back to the application in STEP or GO mode.

In the following example, three commands are specified: CALL %VER; QUERY LOCATION; GO;

All three commands are executed and because of the GO at the end z/OS Debugger runs back to the application until a breakpoint is found, a condition is raised, or the application terminates.




In the following example, the order of the commands is changed to CALL %VER; GO; QUERY LOCATION. In this example, the command QUERY LOCATION is not executed because the GO command tells z/OS Debugger to return to the application.




# Specifying the location of source, listing, or separate debug file in remote debug mode by using environment variables

z/OS Debugger retrieves the information it displays in the Source window from one of the following files:

- source or listing files
- separate debug files (.dbg files or SYSDEBUG)
- EQALANGX files
- .mdbg files (module map files)

If your build or run your applications on UNIX System Services and you move these files, you can use the following environment variables to specify the new location of these files:

**EQA_SRC_PATH**
Specifies the location of the source, listing, SYSDEBUG, or EQALANGX files.

**EQA_DBG_PATH**
Specifies the location of the .dbg file.

**EQA_MDBG_PATH**
Specifies the location of the .mdbg file.

**EQA_DBG_SYSDEBUG**
Specifies the location of SYSDEBUG

The following example shows you how to declare an environment variable that specifies the location of a listing file:

```
export EQA_SRC_PATH="/u/build1/try1:/u/build2/try3:evaf.test.listing"
```

You can have a combination of data sets and UNIX file system paths separated by a colon (:) and enclosed in quotation marks.

# Appendix B. Changes in behavior of some commands

This topic summarizes changes in the default behavior of some commands and in which version of z/OS Debugger those changes were introduced.

## Changes in the behavior introduced with Debug Tool for z/OS, Version 13.1

- The EQALANGP and EQALANGX modules are moved from Debug Tool's EQAW.SEQAMOD library to Common Component's IPV.SIPVMODA library. They are now aliases of IPVLANGP and IPVLANGX respectively. This removes duplication between the two tools.
- Appendix "Quick start guide for compiling and assembling programs for use with IBM Problem Determination Tools products" in the *Debug Tool User's Guide* is removed because this has been placed in the *IBM Problem Determination Tools for z/OS Common Component: Customization Guide and User Guide* instead.

**Note:** Debug Tool for z/OS is now named z/OS Debugger, and Problem Determination Tools for z/OS Common Component is now named IBM Application Delivery Foundation for z/OS Common Components.

## Changes in the behavior introduced with Debug Tool for z/OS, Version 12.1, with the PTF for APAR PM85967 for Enterprise COBOL for z/OS Version 5.1

- BINARY data items are treated the same as "native" binary data items, that is, `COMP-5`.
- Values for Debug Tool variables of string type are displayed with quotation marks around them. For example, `LIST %SYSTEM` is displayed as `%SYSTEM = 'MVS'`.
- The output of `LIST %RC` is changed to be consistent with the display of the program variables of type `S9(4) COMP`. For example, `LIST %RC` result is displayed as `%RC = +00000`.
- When displaying the value of a variable, Debug Tool does not show the block qualification.
- If a Debug Tool session variable has the same name as a program variable, the session variable takes precedence. For example, if you use a `LIST` or `DESCRIBE ATTRIBUTES` command for such a variable, the values that are displayed correspond to the session variable.
- The output of `AUTOMONITOR` for a scalar variable does not show the level number associated with the variable.
- If a statement uses `ADDRESS OF <reference>`, it is the address of the reference that is shown in the automonitor output, not the value of `<reference>`.
- With `SET MONITOR DATATYPE ON`, Debug Tool displays the attributes as declared in the program.
- With `SET MONITOR DATATYPE ON`, Debug Tool displays the resulting attributes of the expression.
- When referencing an array element, you can use an integer literal to increment or decrement an index value, for example, `LIST ARR4(IX3 + 5, 1)`.
- When using a subscript as part of an expression, you need to follow COBOL language rules that specify that the data name or index name should be followed by the operator + or - and a positive or unsigned integer literal.
- Simple assignment between compatible types can be done by using the `SET`, `MOVE`, or `COMPUTE` command.
- Object-oriented programs do not have Factory and Object blocks. Commands like `LIST NAMES`, `DESCRIBE CUS`, `LIST TITLED` will not show any information about these blocks.

- Reference modifications can be applied only to character types that have usage DISPLAY, DISPLAY-1, or NATIONAL. If a reference modification is applied to any other type, an error message is displayed.

# Changes in behavior introduced with Debug Tool for z/OS, Version 11.1

Beginning with Debug Tool for z/OS, Version 11.1, Debug Tool changed how it processed nested blocks in C and C++ programs, thereby improving performance.

The z/OS XL C/C++ compiler defines a block statement (or block) as all definitions, declarations, and statements enclosed with a single set of braces: {}. You can nest blocks and the compiler assigns names to these blocks using the following pattern: %BLOCK*n*, where *n* is a sequential number.

To describe the difference in behavior, review the following example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define NUM_INVOKE 5

main()
{                                   /* block 1 start */
  int elm1 = 5, qvar1, qvar2, i;

  qvar1 = 1;
  qvar2 = 9;

  for(i=0; i<NUM_INVOKE; ++i)
  {                                 /* block 2 start */
    qvar1 = qvar1 + 1;
    qvar2 = qvar2 + 1;
  }                                 /* block 2 end */

  qvar1 = qvar1 + 1;
  qvar2 = qvar2 + 1;

  if(elm1 = 5)
  {                                 /* block 3 start */
    qvar1 = qvar1 + 5;
    qvar1 = qvar1 + 5;
  }                                 /* block 3 end */
  else
  {                                 /* block 4 start */
    qvar1 = qvar1 - 5;
    qvar2 = qvar2 - 5;
  }                                 /* block 4 end */

  qvar1 = qvar1 - 1;
  qvar2 = qvar2 - 1;

  } /* block 1 end */
```

**Previous behavior**

After you enter the AT ENTRY, AT EXIT, AT GLOBAL, or AT PATH command, z/OS Debugger would stop at the start, end, or both (depending on which command you entered) of all the blocks.

**New behavior**

After you enter the AT ENTRY, AT EXIT, AT GLOBAL, or AT PATH command, z/OS Debugger stops only at the start, end, or both (depending on which command you entered) of block 1.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**

Description of block statements in *z/OS XL C/C++ Language Reference*

“AT ENTRY command” on page 54
“AT EXIT command” on page 56
“AT GLOBAL command” on page 58
“AT PATH command” on page 69

## Changes in behavior introduced with Debug Tool for z/OS, Version 10.1

Beginning with Debug Tool for z/OS, Version 10.1, the Debug Tool DTCN Primary Menu changed how to identify the program or programs you want to debug. Previously, you identified a program through the Program ID field. This changed to two fields: LoadMod and CU.

## Changes in behavior introduced with Debug Tool for z/OS, Version 9.1, with the PTF for APAR PK74749 applied

Beginning with Debug Tool for z/OS, Version 9.1, with the PTF for APAR PK74749 applied, Debug Tool changed how it handled pointers in C/C++ programs to better match the semantics of C/C++. The following commands were affected by this change:

- An `AT CHANGE` command that references a pointer. For example, `AT CHANGE p`.

  **Previous behavior**
  Debug Tool stops when p changes.

  **New behavior**
  Debug Tool stops when the value of what p points to changes.

- A `LIST STORAGE` command that references a pointer. For example, `LIST STORAGE(p,0,4)`.

  **Previous behavior**
  Debug Tool displays the contents of p.

  **New behavior**
  Debug Tool displays the contents of what p points to.

- A `MEMORY` command that references a pointer. For example, `MEMORY p`.

  **Previous behavior**
  Debug Tool displays the contents of p.

  **New behavior**
  Debug Tool displays the contents of what p points to.

- A `STORAGE` command that references a pointer. For example, `STORAGE(p,0,4)`.

  **Previous behavior**
  Debug Tool changes the contents of p

  **New behavior**
  Debug Tool changes the contents of what p points to.

Refer to the following topics for more information related to the material discussed in this topic.

**Related references**
"AT CHANGE command (full screen mode, line mode, batch mode)" on page 45
"LIST STORAGE command" on page 163
"MEMORY command" on page 169
"WINDOW SWAP command" on page 284
"STORAGE command" on page 271

# Appendix C. Limitations of 64-bit support in Debug Tool compatibility mode

Debug Tool compatibility mode, a subset of remote debug mode, supports 64-bit COBOL and C/C++ programs with some limitations. For 64-bit PL/I programs, use standard mode, another subset of remote debug mode, instead.

Subsystems IMS, Db2, and CICS are not supported.

The following functions are not supported for 64-bit COBOL and C/C++ programs in Debug Tool compatibility mode:

- The EQAUEDAT user exit, EQA_DBG_PATH environment variable, and EQA_SRC_PATH environment variable
- Language Environment user exit (CEEBXITA)
- Compiled in hooks

In addition to all the commands strictly related to the unsupported subsystems and functions mentioned above, the following commands are not supported for 64-bit COBOL and C/C++ programs in Debug Tool compatibility mode:

- z/OS Debugger commands:
  - CALL %FA
  - SET DYNDEBUG
- EQAOPTS commands:
  - EQAQPP
  - DLAYDBGCND
  - DLAYDBGXRF
  - DYNDEBUG
  - SVCSCREEN

# Appendix D. Support resources and problem solving information

This section shows you how to quickly locate information to help answer your questions and solve your problems. If you have to call IBM support, this section provides information that you need to provide to the IBM service representative to help diagnose and resolve the problem.

## Accessing the IBM Support portal

You must be a registered user on the IBM Support portal to download fixes and to submit a problem online to the IBM Support community.

If you need to look beyond IBM Documentation to answer your question or resolve your problem, you can use one or more of the following approaches:

- Open the IBM Support portal.
- Click to log in using your IBM.com username.
- On the IBM Support portal, you can do the following tasks:
  - Search for known issues, documentation, and support forums.
  - Open a case or view cases you opened.
  - Open a chat window with a Support representative.
  - Open Fix Central to view product downloads and updates.
  - Access product documentation and support forums.
  - Manage your support account, including notifications, invoices, orders, contracts, and warranties. For more information about notifications, see “Subscribing to support updates” on page 529.

## Getting fixes

A product fix might be available to resolve your problem. To determine what fixes and other updates are available, select a link from the following list:

- Latest PTFs for z/OS Debugger
- Latest PTFs for IBM Developer for z/OS Enterprise Edition
- Latest PTFs for ADFz Common Components

When you find a fix that you are interested in, click the name of the fix to read its description and to optionally download the fix.

Subscribe to receive e-mail notifications about fixes and other IBM Support information as described in Subscribing to Support updates.

## Subscribing to support updates

To receive automatic updates when IBM publishes new support content for your products, subscribe to weekly email updates or RSS feeds. Support content might include information about new releases, fixes, technotes, APARs, and support flashes.

To sign up for email updates, you must be a registered user on the IBM Support community website.

To subscribe to Support updates, follow the steps below.

1. Open the IBM Support portal website.
2. Click to log in using your IBM.com username.

3. Click **Manage support account** > **Notifications** to view your notifications.
4. Type the product name in the search field or click **Browse for a product**.
5. Type the product name in the **Product lookup** field, or click **Browse for a product**.
6. Click **Subscribe** beside your product, and in the **Select document types** window, select the types of documents for which you want to receive information. Click **Submit**.
7. Optionally, you can click the RSS/Atom feed by clicking **Links**. Then, copy and paste the link into your feeder.
8. To see any notifications that were sent to you, click **View**.

# Contacting IBM Support

To submit your problem to IBM Support, you must have an active Passport Advantage® software maintenance agreement. Passport Advantage is the IBM comprehensive software licensing and software maintenance (product upgrades and technical support) offering. You can enroll online on the Passport Advantage website.

- To learn more about Passport Advantage, see the Passport Advantage FAQs.
- For further assistance, contact your IBM representative.

To submit your problem online (from the IBM website) to IBM Support:

- Be a registered user on the IBM Support website. For details about registering, see Registering on the IBM Support website.
- Be listed as an authorized caller in the service request tool.

## Determine the business impact of your problem

When you report a problem to IBM, you are asked to supply a severity level. Therefore, you must understand and assess the business impact of the problem that you are reporting.

**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**
This 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.

## Gather diagnostic information

To save time, if there is a MustGather document available for the product, refer to the MustGather document and gather the information specified. MustGather documents contain specific instructions for submitting your problem to IBM and gathering information needed by the IBM support team to resolve your problem. To determine if there is a MustGather document for this product, go to the product support page and search on the term MustGather. At the time of this publication, the following MustGather documents are available:

- MustGather: Read first for problems encountered with z/OS Debugger: https://www.ibm.com/support/pages/node/89125
- MustGather: Read first for problems encountered with code coverage: https://www.ibm.com/support/pages/node/6561317

If the product does not have a MustGather document, provide answers to the following 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.

## Submit the problem to IBM Support

You can submit your problem to IBM Support in the following ways:

- Online: Open the IBM Support community website. Click **Open a case** to open a service request and describe the problem in detail.
- By phone: For the phone number to call in your country or region, see the IBM Directory of worldwide contacts and click the name of your country or geographic region.
- Through your IBM Representative: If you cannot access IBM Support online or by phone, contact your IBM Representative. If necessary, your IBM Representative can open a service request for you. You can find complete contact information for each country at IBM Directory of worldwide contacts.

If the problem you submit is for a software defect or for missing or inaccurate documentation, IBM Support creates an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Support provides a workaround that you can implement until the APAR is resolved and a fix is delivered. IBM publishes resolved APARs on the IBM Support website daily, so that other users who experience the same problem can benefit from the same resolution.

# Appendix E. 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 z/OS Debugger.

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

IBM Documentation, and its related publications, are accessibility-enabled. The accessibility features of the information center are described at https://www.ibm.com/docs.

## 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 format of this document is accessible to visually impaired individuals who use a screen reader:

- HTML format when viewed from IBM Documentation
- PDF format

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.

# 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:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711

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 illustrate 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 function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:
© (your company name) (year).

Portions of this code are derived from IBM Corp. Sample Programs.


## Privacy policy considerations

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, or to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies", and "the IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

## Programming interface information

This document 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 z/OS Debugger.

## Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Other company, product, or service names may be trademarks or service marks of others.

# Glossary

This glossary defines technical terms and abbreviations used in *IBM z/OS Debugger 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**
z/OS Debugger 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.

## E

**eXtra Performance LINKage (XPLINK)**
A new call linkage between functions that has the potential for a significant performance increase when used in an environment of frequent calls between small functions. XPLINK makes subroutine calls more efficient by removing nonessential instructions from the main path. When all functions are compiled with the XPLINK option, pointers can be used without restriction, which makes it easier to port new applications to z/OS.

## 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.

## N

**network identifier**
In TCP/IP, that part of the IP address that defines a network. The length of the network ID depends on the type of network class (A, B, or C).

## 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.

**statement**
An instruction in a program or procedure.

In programming languages, a language construct that represents a step in a sequence of actions or a set of declarations.

## X

**XPLINK**
See eXtra Performance LINKage (XPLINK).

# IBM z/OS Debugger publications

You can access the IBM z/OS Debugger publications by visiting any of the following pages:

- IBM Debug for z/OS:
  - IBM Documentation: https://www.ibm.com/docs/SSVSZX_15.0.0/com.ibm.debug.z.doc/topics/pdf.html
  - Library page: https://www.ibm.com/support/pages/node/713283
- IBM Developer for z/OS:
  - IBM Documentation: https://www.ibm.com/docs/SSQ2R2_15.0.0/com.ibm.debug.z.doc/topics/pdf.html
  - Library page: https://www.ibm.com/support/pages/node/713179
- IBM Z and Cloud Modernization Stack:
  - IBM Documentation: https://www.ibm.com/docs/SSV97FN_latest/com.ibm.debug.z.doc/topics/pdf.html

# Index

## Special Characters



## Numerics



## A

## B



## C

## D

## E



## F

## G



## H



## I



## J



## K



## L

## M



## N



## O



## P

## Q



## R



## S

## T

## U



## V



## W



## Z

Product Number: 5724-T07