This edition applies to Debug Tool for z/OS, Version 10.1 (Program Number 5655-V50), which supports the following compilers:

- AD/Cycle C/370 Version 1 Release 2 (Program Number 5688-216)
- C/C++ for MVS/ESA Version 3 (Program Number 5655-121)
- C/C++ feature of OS/390 (Program Number 5647-A01)
- C/C++ feature of z/OS (Program Number 5694-A01)
- OS/VS COBOL, Version 1 Release 2.4 (5740-CB1) - with limitations
- VS COBOL II Version 1 Release 3 and Version 1 Release 4 (Program Numbers 5668-958, 5688-023) - with limitations
- COBOL/370 Version 1 Release 1 (Program Number 5688-197)
- COBOL for MVS & VM Version 1 Release 2 (Program Number 5688-197)
- COBOL for OS/390 & VM Version 2 (Program Number 5648-A25)
- Enterprise COBOL for z/OS and OS/390 Version 3 (Program Number 5655-G53)
- Enterprise COBOL for z/OS, Version 4.2 and earlier (Program Number 5655-S71)
- PL/I for MVS & VM Version 1 Release 1 (Program Number 5688-235)
- VisualAge PL/I for OS/390 Version 2 Release 2 (Program Number 5655-B22)

- Enterprise PL/I for z/OS and OS/390 Version 3.9 or earlier (Program Number 5655-H31)

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

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

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

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR prefix (full-screen mode)</td>
<td>92</td>
</tr>
<tr>
<td>COMMENT command</td>
<td>93</td>
</tr>
<tr>
<td>COMPUTE command (COBOL)</td>
<td>93</td>
</tr>
<tr>
<td>CURSOR command (full-screen mode)</td>
<td>94</td>
</tr>
<tr>
<td>Declarations (assembler, disassembly, and</td>
<td>95</td>
</tr>
<tr>
<td>non-Language Environment COBOL)</td>
<td></td>
</tr>
<tr>
<td>Declarations (C and C++)</td>
<td>96</td>
</tr>
<tr>
<td>Declarations (COBOL)</td>
<td>98</td>
</tr>
<tr>
<td>DECLARE command (PL/I)</td>
<td>100</td>
</tr>
<tr>
<td>DESCRIBE command</td>
<td>102</td>
</tr>
<tr>
<td>DISABLE command</td>
<td>106</td>
</tr>
<tr>
<td>DISABLE prefix (full-screen mode)</td>
<td>108</td>
</tr>
<tr>
<td>DO command (assembler, disassembly, and</td>
<td>108</td>
</tr>
<tr>
<td>non-Language Environment COBOL)</td>
<td></td>
</tr>
<tr>
<td>do/while command (C and C++)</td>
<td>109</td>
</tr>
<tr>
<td>DO command (PL/I)</td>
<td>109</td>
</tr>
<tr>
<td>ENABLE command</td>
<td>112</td>
</tr>
<tr>
<td>ENABLE prefix (full-screen mode)</td>
<td>113</td>
</tr>
<tr>
<td>EVALUATE command (COBOL)</td>
<td>114</td>
</tr>
<tr>
<td>Expression command (C and C++)</td>
<td>116</td>
</tr>
<tr>
<td>FIND command</td>
<td>116</td>
</tr>
<tr>
<td>FINDBP command</td>
<td>121</td>
</tr>
<tr>
<td>for command (C and C++)</td>
<td>123</td>
</tr>
<tr>
<td>FREE command</td>
<td>124</td>
</tr>
<tr>
<td>GO command</td>
<td>124</td>
</tr>
<tr>
<td>GOTO command</td>
<td>125</td>
</tr>
<tr>
<td>GOTO LABEL command</td>
<td>127</td>
</tr>
<tr>
<td>%IF command (programming language neutral)</td>
<td>128</td>
</tr>
<tr>
<td>IF command (assembler, disassembly, and</td>
<td>129</td>
</tr>
<tr>
<td>non-Language Environment COBOL)</td>
<td></td>
</tr>
<tr>
<td>if command (C and C++)</td>
<td>130</td>
</tr>
<tr>
<td>IF command (COBOL)</td>
<td>131</td>
</tr>
<tr>
<td>Allowable comparisons for the IF command</td>
<td>132</td>
</tr>
<tr>
<td>(COBOL)</td>
<td></td>
</tr>
<tr>
<td>IF command (PL/I)</td>
<td>135</td>
</tr>
<tr>
<td>IMMEDIATE command (full-screen mode)</td>
<td>136</td>
</tr>
<tr>
<td>INPUT command (C, C++, and COBOL)</td>
<td>137</td>
</tr>
<tr>
<td>JUMPTO command</td>
<td>138</td>
</tr>
<tr>
<td>JUMPTO LABEL command</td>
<td>139</td>
</tr>
<tr>
<td>LIST command</td>
<td>141</td>
</tr>
<tr>
<td>LIST (blank) command</td>
<td>142</td>
</tr>
<tr>
<td>LIST AT command</td>
<td>142</td>
</tr>
<tr>
<td>LIST CALLS command</td>
<td>145</td>
</tr>
<tr>
<td>LIST CONTAINER command</td>
<td>146</td>
</tr>
<tr>
<td>LIST CURSOR command (full-screen mode)</td>
<td>147</td>
</tr>
<tr>
<td>LIST DTNC or CADP command</td>
<td>148</td>
</tr>
<tr>
<td>LIST expression command</td>
<td>149</td>
</tr>
<tr>
<td>LIST FREQUENCY command</td>
<td>153</td>
</tr>
<tr>
<td>LIST LAST command</td>
<td>154</td>
</tr>
<tr>
<td>LIST LINE NUMBERS command</td>
<td>155</td>
</tr>
<tr>
<td>LIST LINES command</td>
<td>155</td>
</tr>
<tr>
<td>LIST MONITOR command</td>
<td>155</td>
</tr>
<tr>
<td>LIST NAMES command</td>
<td>155</td>
</tr>
<tr>
<td>LIST ON (PL/I) command</td>
<td>157</td>
</tr>
<tr>
<td>LIST PROCEDURES command</td>
<td>158</td>
</tr>
<tr>
<td>LIST REGISTERS command</td>
<td>158</td>
</tr>
<tr>
<td>LIST STATEMENT NUMBERS command</td>
<td>159</td>
</tr>
<tr>
<td>LIST STATEMENTS command</td>
<td>160</td>
</tr>
<tr>
<td>LIST STORAGE command</td>
<td>160</td>
</tr>
<tr>
<td>LOAD command</td>
<td>162</td>
</tr>
<tr>
<td>LOADDEBUGDATA command</td>
<td>164</td>
</tr>
<tr>
<td>MEMORY command</td>
<td>165</td>
</tr>
<tr>
<td>MONITOR command</td>
<td>166</td>
</tr>
<tr>
<td>M prefix (full-screen mode)</td>
<td>169</td>
</tr>
<tr>
<td>MOVE command (COBOL)</td>
<td>171</td>
</tr>
<tr>
<td>Allowable moves for the MOVE command</td>
<td>172</td>
</tr>
<tr>
<td>NAMES command</td>
<td>175</td>
</tr>
<tr>
<td>NAMES DISPLAY command</td>
<td>175</td>
</tr>
<tr>
<td>NAMES EXCLUDE command</td>
<td>176</td>
</tr>
<tr>
<td>NAMES INCLUDE command</td>
<td>177</td>
</tr>
<tr>
<td>Null command</td>
<td>178</td>
</tr>
<tr>
<td>ON command (PL/I)</td>
<td>178</td>
</tr>
<tr>
<td>PANEL command (full-screen mode)</td>
<td>180</td>
</tr>
<tr>
<td>PERFORM command (full-screen mode)</td>
<td>182</td>
</tr>
<tr>
<td>PLAYBACK commands</td>
<td>184</td>
</tr>
<tr>
<td>PLAYBACK ENABLE command</td>
<td>184</td>
</tr>
<tr>
<td>PLAYBACK START command</td>
<td>185</td>
</tr>
<tr>
<td>PLAYBACK FORWARD command</td>
<td>187</td>
</tr>
<tr>
<td>PLAYBACK BACKWARD command</td>
<td>187</td>
</tr>
<tr>
<td>PLAYBACK STOP command</td>
<td>187</td>
</tr>
<tr>
<td>PLAYBACK DISABLE command</td>
<td>187</td>
</tr>
<tr>
<td>POPUP command</td>
<td>188</td>
</tr>
<tr>
<td>POSITION command</td>
<td>188</td>
</tr>
<tr>
<td>Prefix commands (full-screen mode).</td>
<td>188</td>
</tr>
<tr>
<td>PROCEDURE command</td>
<td>189</td>
</tr>
<tr>
<td>QUALIFY RESET command</td>
<td>190</td>
</tr>
<tr>
<td>QUERY command</td>
<td>190</td>
</tr>
<tr>
<td>QUERY prefix (full-screen mode)</td>
<td>195</td>
</tr>
<tr>
<td>QUIT command</td>
<td>196</td>
</tr>
<tr>
<td>QQUIT command</td>
<td>197</td>
</tr>
<tr>
<td>RESTORE command</td>
<td>198</td>
</tr>
<tr>
<td>RETRIEVE command (full-screen mode)</td>
<td>199</td>
</tr>
<tr>
<td>RUN command</td>
<td>200</td>
</tr>
<tr>
<td>RUNTO command</td>
<td>200</td>
</tr>
<tr>
<td>RUNTO prefix command (full-screen mode)</td>
<td>201</td>
</tr>
<tr>
<td>SCROLL command (full-screen mode)</td>
<td>201</td>
</tr>
<tr>
<td>SELECT command (PL/I)</td>
<td>204</td>
</tr>
<tr>
<td>SET command</td>
<td>205</td>
</tr>
<tr>
<td>SET ASSEMBLER ON/OFF command</td>
<td>207</td>
</tr>
<tr>
<td>SET AUTOMONITOR command</td>
<td>208</td>
</tr>
<tr>
<td>SET AUTOMONITOR command</td>
<td>209</td>
</tr>
<tr>
<td>SET CHANGE command</td>
<td>211</td>
</tr>
<tr>
<td>SET COLOR command (full-screen and line mode)</td>
<td>212</td>
</tr>
<tr>
<td>SET COUNTRY command</td>
<td>215</td>
</tr>
<tr>
<td>SET DBCS command</td>
<td>215</td>
</tr>
<tr>
<td>SET DEFAULT DBG command</td>
<td>216</td>
</tr>
<tr>
<td>SET DEFAULT LISTINGS command</td>
<td>217</td>
</tr>
<tr>
<td>SET DEFAULT MDBG command</td>
<td>218</td>
</tr>
<tr>
<td>SET DEFAULT SCROLL command (full-screen mode)</td>
<td>220</td>
</tr>
<tr>
<td>SET DEFAULT VIEW command</td>
<td>220</td>
</tr>
<tr>
<td>SET DEFAULT WINDOW command (full-screen mode)</td>
<td>221</td>
</tr>
<tr>
<td>SET DISASSEMBLY command</td>
<td>221</td>
</tr>
<tr>
<td>SET DYNDDEBUG command</td>
<td>224</td>
</tr>
<tr>
<td>SET ECHO command</td>
<td>224</td>
</tr>
<tr>
<td>SET EQUATE command</td>
<td>225</td>
</tr>
<tr>
<td>SET EXECUTE command</td>
<td>226</td>
</tr>
<tr>
<td>SET FIND BOUNDS command</td>
<td>226</td>
</tr>
<tr>
<td>SET FREQUENCY command</td>
<td>227</td>
</tr>
</tbody>
</table>
About this document

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

This document contains descriptions of the commands, functions, and variables available through Debug Tool, as well as the messages that you might see as you use Debug Tool. Many Debug Tool 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 non-Language Environment COBOL.

Who might use this document

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

Debug Tool 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 a dedicated terminal only
- WebSphere® in remote debug mode or full-screen mode using a dedicated terminal only

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

Accessing z/OS licensed documents on the Internet

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

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

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

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

http://www.ibm.com/servers/resourcelink
To register for access to the z/OS licensed documents:
1. Sign in to Resource Link using your Resource Link user ID and password.
2. Select User Profiles located on the left-hand navigation bar.

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

Printed licensed documents are not available from IBM.

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

---

**Using LookAt to look up message explanations**

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

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

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

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

---

**How this document is organized**

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

- Chapter 1 describes the syntax of the TEST run-time option.
- Chapters 2, 3, 4, and 5 describe the complete syntax of the Debug Tool commands.
- Chapters 6 and 7 describe the syntax of Debug Tool built-in functions and variables.
• Chapters 8, 9, 10, 11, 12, and 13 lists all the messages that Debug Tool and other tools shipped with Debug Tool might display.

• Appendix A, “Reference card: Frequently used Debug Tool commands,” on page 387 has a reference card that can be printed. The reference card contains commands that are frequently used.

• Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393 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 D, “Support resources and problem solving information,” on page 399 describes the resources available to help you solve any problems you might have with Debug Tool.

• Appendix E, “Accessibility,” on page 407 describes the features and tools available to people with physical disabilities that help them use Debug Tool and Debug Tool documents.

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

Terms used in this document

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

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

Notes:

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

**assembler**

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

**COBOL**

Refers to the all COBOL compilers supported by Debug Tool except the COBOL compilers described in the term non-Language Environment COBOL.

**disassembly or disassembled**

Refers to high-level language programs compiled without debug information or assembler programs without debug information. The debugging support Debug Tool provides for these programs is through the disassembly view.

**Enterprise PL/I**

Refers to the Enterprise PL/I for z/OS and OS/390® and the VisualAge® PL/I for OS/390 compilers.

**non-Language Environment COBOL**

Refers to any of the following COBOL programs:

- Programs compiled with the IBM OS/VS COBOL compiler.
- Programs compiled with the VS COBOL II compiler with the NOTEST compiler option and linked with a non-Language Environment library.

As you read through the information in this document, remember that OS/VS COBOL programs are non-Language Environment programs, even though you might have used Language Environment libraries to link and run your program.

VS COBOL II programs are non-Language Environment programs when you compile them with the NOTEST compiler option and link them with a non-Language Environment library. VS COBOL II programs are Language Environment programs when you compile them with the TEST compiler option and link them with the Language Environment library.

Read the information regarding non-Language Environment programs for instructions on how to start Debug Tool and debug non-Language Environment COBOL programs, unless information specific to non-Language Environment 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:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>/SM590000/</td>
<td>Indicates the beginning of the syntax diagram.</td>
</tr>
<tr>
<td>──/SM590000</td>
<td>Indicates that the syntax diagram is continued to the next line.</td>
</tr>
<tr>
<td>──/SM590000</td>
<td>Indicates that the syntax is continued from the previous line.</td>
</tr>
<tr>
<td>──/SM630000</td>
<td>Indicates the end of the syntax diagram.</td>
</tr>
</tbody>
</table>

Syntax items

Syntax diagrams contain many different items. Syntax items include:

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

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

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

Syntax examples

The following table provides syntax examples.
**Table 1. Syntax examples**

<table>
<thead>
<tr>
<th>Item</th>
<th>Syntax example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required item.</td>
<td>Required items appear on the main path of the horizontal line. You must specify these items.</td>
</tr>
<tr>
<td>Required choice.</td>
<td>A required choice (two or more items) appears in a vertical stack on the main path of the horizontal line. You must choose one of the items in the stack.</td>
</tr>
<tr>
<td>Optional item.</td>
<td>Optional items appear below the main path of the horizontal line.</td>
</tr>
<tr>
<td>Optional choice.</td>
<td>An optional choice (two or more items) appears in a vertical stack below the main path of the horizontal line. You may choose one of the items in the stack.</td>
</tr>
<tr>
<td>Default.</td>
<td>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.</td>
</tr>
<tr>
<td>Variable.</td>
<td>Variables appear in lowercase italics. They represent names or values.</td>
</tr>
<tr>
<td>Repeatable item.</td>
<td>An arrow returning to the left above the main path of the horizontal line indicates an item that can be repeated.</td>
</tr>
<tr>
<td>Fragment.</td>
<td>The fragment symbol indicates that a labelled group is described below the main syntax diagram. Syntax is occasionally broken into fragments if the inclusion of the fragment would overly complicate the main syntax diagram.</td>
</tr>
</tbody>
</table>
How to send your comments

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

- Use the Online Readers’ Comment Form at www.ibm.com/software/awdtools/rcf/. Be sure to include the name of the document, the publication number of the document, the version of Debug Tool, and, if applicable, the specific location (for example, page number) of the text that you are commenting on.

- Fill out the Readers’ Comment Form at the back of this document, and return it by mail or give it to an IBM representative. If the form has been removed, address your comments to:

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

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

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

This section lists the key changes made to Debug Tool for z/OS.

Changes introduced with Debug Tool V10.1

The following changes, if applicable, are marked with revision bars:

- A RESTful HTTP access interface to read, create, update, and delete profiles in the DTCN profile repository has been added. The interface is described in Debug Tool API User's Guide and Reference. An example of a GUI interface that uses the RESTful HTTP access interface to manipulate the profiles from the workstation is also available. How to download and install this example is described in Debug Tool API User's Guide and Reference.

- If you are using z/OS XLC C/C++, Version 1 Release 11, Debug Tool has added support for .mdbg files that contain source. The .mdbg file contains the debug information and a copy of the source needed in your debug session. You no longer need to have access to the source while debugging your program.

  If you are not familiar with .mdbg files and how to create them, see the topics "dbgld - Create a module map for debugging" or "CDADBGLD - Create a debug side file for the module map" in z/OS XL C/C++ User's Guide.

  You can indicate that Debug Tool always use .mdbg files to search for source and debug information by setting the EQAXOPT option MDBG to YES in the EQAOPTS options file. To learn how to set the MDBG option, see “Specifying whether Debug Tool searches through .mdbg files” in Debug Tool Customization Guide. In situations where you can specify environment variables, you can set the environment variable EQA_USE_MDBG to YES or NO, which overrides any setting (including the default setting) of the EQAXOPT MDBG option.

  To learn what compiler options to choose to create .dbg files, which the dbgld command or CDADBGLD utility use to build the .mdbg files, see the following topics in Debug Tool User's Guide:

  - "Choosing DEBUG compiler options for C programs"
  - "Choosing DEBUG compiler options for C++ programs"

  There are several different methods of specifying the location of .dbg and .mdbg files. The following list summarizes each method:

  - While you are debugging your program, you can use the following commands:

    - “SET DEFAULT DBG command” on page 216
    - “SET DEFAULT MDBG command” on page 218
    - Only for full screen, batch, and line mode: “SET MDBG command” on page 236

  After you specify the location of the .dbg and .mdbg files, you can use the following commands to verify the location:

    - The QUERY SET DEFAULT DBG and QUERY SET DEFAULT MDBG commands, which are described in “QUERY command” on page 190.
    - Only for full screen, batch, and line mode: QUERY SET DBG, which is described in “QUERY command” on page 190

  All of these commands are described in Debug Tool Reference and Messages.
In your JCL, you can add EQADBG and EQAMDBG DD statements and specify the data set name of the corresponding .dbg or .mdbg file. To learn more about using these DD statements, see “Compiling your program without using Debug Tool Utilities” in Debug Tool User’s Guide. If you are debugging in UNIX System Services on or in CICS, you cannot use these DD statements.

In UNIX System Services, you can use the following environment variables:
- EQA_DBG_PATH
- EQA_MDBG_PATH

The term full-screen mode through a VTAM® terminal has been changed to full-screen mode using a dedicated terminal. The term was changed to remove the implication that any instructions that referred to VTAM terminal applied only to those terminals connected through an SNA network.

You can now use IBM Session Manager while debugging in full-screen mode using a dedicated terminal and using the Terminal Interface Manager.

The following topics have been added to or updated in the Debug Tool Customization Guide:
- “Example: a debugging session using the Debug Tool Terminal Interface Manager”
- “Starting the Debug Tool Terminal Interface Manager”
- “Configuring Terminal Interface Manager as an IBM Session Manager application”

In the Debug Tool User’s Guide, the topic “Starting a debugging session in full-screen mode using a dedicated terminal” has been updated.

The CODEPAGE(ccsid) option has been added to the XML option of the LIST CONTAINER and LIST STORAGE commands to improve the display of character strings encoded in an alternate code page on a 3270 terminal.

In the Debug Tool Reference and Messages and Debug Tool Reference Summary, the descriptions and syntax diagrams of the following commands have been updated:
- “LIST CONTAINER command” on page 146
- “LIST STORAGE command” on page 160

You can now add all the variables in the Working-Storage Section of a COBOL program to the Monitor window with one command.

In Debug Tool User’s Guide, the topic “Displaying the Working-Storage Section of a COBOL program in the Monitor window” has been added to describe how to add these variables to the Monitor window.

In Debug Tool Reference and Messages, the topic “MONITOR command” on page 166 has been updated to describe the new suboption WSS.

In Debug Tool Reference Summary, the syntax diagram in “MONITOR command” has been updated to include the new suboption WSS.

In full screen mode, a new window called the Command pop-up window has been added that makes it easier to enter and edit long commands.

In Debug Tool User’s Guide, the following topics have been added or updated:
- “Command pop-up window”
- “Opening the Command pop-up window to enter long Debug Tool commands”
- “Entering multiline commands in full-screen”

In Debug Tool Reference Summary and Debug Tool Reference and Messages, the following topics have been added:
• You can now enter changes to multiple variables in the Monitor window at one time.

In Debug Tool User’s Guide, an item on the list in “Restrictions for modifying variables in the Monitor window” has been removed.

• Debug Tool now supports automatic saving and restoring of breakpoints and settings for IMS Transaction Manager (TM) programs.

In Debug Tool User’s Guide phrases that describe this limitation have been removed from the following topics:

  - “Restoring Manually”
  - “Data sets used by Debug Tool”

In Debug Tool Reference and Messages phrases that describe this limitation have been removed from the following topics:

  - “SET RESTORE command” on page 247
  - “SET SAVE command” on page 249

• In Debug Tool Reference and Messages, the syntax diagram for “LIST expression command” on page 149 has been updated to include the use of the GROUP option for COBOL programs.

• Two commands, POSITION and FINDBP, have been added to improve the ability to scroll to a specific line. You can use POSITION integer, which is similar to SCROLL TO integer, to scroll to a particular line or statement. FINDBP, which is similar to FIND, searches for line, statement, or offset breakpoints in the Source window.

In Debug Tool User’s Guide the following topics have been updated to clarify how to scroll to a particular line:

  - “Scrolling to a particular line number”
  - “Displaying the line at which execution halted”

In Debug Tool Reference and Messages and Debug Tool Reference Summary, the commands FINDBP and POSITION are described in topics “FINDBP command” on page 121 and “POSITION command” on page 188.

• The way to identify, in DTCN profiles, the program you want to debug has changed.

Previously, you identified a program through the Program ID field. This has changed to two fields: LoadMod and CU.

In Debug Tool Reference Summary, the following syntax diagrams have been updated to describe the new options:

  - “DISABLE command”
  - “ENABLE command”
  - “LIST DTCN or CADP command”
  - DTCNFORCELOADMODID, which is described in “EQAOPTS options”
  - DTCNFORCECUID, which is described in “EQAOPTS options”

In Debug Tool Reference and Messages, the following syntax diagrams have been updated to describe the new options:

  - “DISABLE command”
  - “ENABLE command”
  - “LIST DTCN or CADP command”

In Debug Tool User’s Guide, the instructions in “Creating and storing a DTCN profile” have been updated to describe the new fields.
In *Debug Tool Customization Guide*, DTCNFORCELOADMODID and DTCNFORCECUID have been added to “Defining EQAOPTS options: checklist and instructions”.

- Debug Tool now supports running in browse mode. In this mode, you cannot make modifications to storage or registers, nor modify the control flow of a program with commands like GOTO and JUMPTO. You can debug a program, but you cannot change the behavior of a program. This might be useful when you want to debug a program running in a production environment but you want to prevent unauthorized changes to a program’s behavior or production data.

In *Debug Tool Reference Summary* and *Debug Tool Reference and Messages*, the description of the following commands have been updated to describe how you cannot use them in browse mode:

- “ALLOCATE command” on page 33
- “Assignment command (assembler and disassembly)” on page 35
- “Assignment command (non-Language Environment COBOL)” on page 38
- “Assignment command (PL/I)” on page 38
- “CALL %CECI command” on page 79
- “CALL entry_name command (COBOL)” on page 86
- “CALL %FM command” on page 85
- “CALL %HOGAN command” on page 85
- CLEAR LOG command, which is described in “CLEAR command” on page 88
- COMPUTE command (COBOL)” on page 93
- “FREE command” on page 124
- 60 BYPASS command, which is described in “GO command” on page 124
- “GOTO command” on page 125
- “GOTO LABEL command” on page 127
- “INPUT command (C, C++, and COBOL)” on page 137
- “JUMPTO command” on page 138
- “JUMPTO LABEL command” on page 139
- “MEMORY command” on page 165
- “MOVE command (COBOL)” on page 171
- “QUIT command” on page 196
- QUIT expression command, which is described in “QUIT command” on page 196
- “QQUIT command” on page 197
- “SET INTERCEPT command (C and C++)” on page 229
- “SET INTERCEPT command (COBOL, full-screen mode, line mode, batch mode)” on page 230
- “SET INTERCEPT command (COBOL, remote debug mode)” on page 231
- “SET command (COBOL)” on page 260
- “STORAGE command” on page 266
- “SYSTEM command (z/OS)” on page 270
- “TRIGGER command” on page 271
- “TSO command (z/OS)” on page 274

The topic “QUERY command” on page 190 has been updated to describe the new option BROWSE MODE.
In [Debug Tool User’s Guide](#), the topic “Choosing a debugging mode” has been updated to describe how browse mode works and how you control browse mode.

In [Debug Tool Customization Guide](#), the following topics have been added to describe the customization tasks you must do for this feature:

- “Installing the browse mode RACF® facility”
- “Enabling users to control browse mode”

- **Debug Tool** now supports displaying more than 1000 lines in the Monitor window.

In [Debug Tool Reference and Messages](#), the usage note that describes this limitation has been removed from “MONITOR command” on page 166.

In [Debug Tool Reference and Messages](#) and [Debug Tool Reference Summary](#), the following topics have been updated:

- The topic “MONITOR command” on page 166 has been updated to describe the new option LIMIT.
- The topic “QUERY command” on page 190 has been updated to describe the new option MONITOR LIMIT.

In the [Debug Tool User’s Guide](#), the topic “Monitor window” has been updated to describe how to increase the number of lines that the Monitor window displays and the implications of monitoring large volumes of data.

- **Debug Tool** now supports monitoring, by using the AT CHANGE command, of assembler variables with dynamically updated addresses such as those in a DSECT.

In [Debug Tool Reference and Messages](#), an existing usage note has been modified and a new usage note has been added to “AT CHANGE command (full screen mode, line mode, batch mode)” on page 48 that describes how Debug Tool monitors these variables.

- **Debug Tool** now supports debugging C and C++ programs that run in the Airline Control System (ALCS). This support is available only if you debug in remote debug mode.

In the [Debug Tool User’s Guide](#), the following topics have been updated:

- “A table that lists the supported subsystems” has been updated to indicate that Debug Tool supports the ALCS subsystem.
- “Choosing TEST or NOTEST compiler suboptions for C programs” has been updated to indicate that if you want to debug C and C++ programs running in ALCS, you must compile them with hooks.
- “Choosing TEST or NOTEST compiler options for C++ programs” has been updated to indicate that if you want to debug C and C++ programs running in ALCS, you must compile them with hooks.
- “Choosing a debugging mode” has been updated to indicate for the ALCS subsystem, you must choose remote debug mode.

- **Debug Tool** now supports using the L and M prefix commands for assembler and disassembly programs.

In the [Debug Tool User’s Guide](#), the following topics have been updated to describe how to use the L and M prefix commands on assembler and disassembly programs:

- “Displaying the value of a variable”
- “Entering prefix commands on specific lines or statements”
- “Displaying and monitoring the value of a variable”
- “One-time display of the value of variables”
Some of these topics also describe a slight change in terminology. These topics use the word "operand" to mean a variable in C, C++, COBOL, or PL/I, or the operand of an assembler instruction.

In *Debug Tool Reference and Messages*, the topics “L prefix command (full-screen mode)” on page 151 and “M prefix (full-screen mode)” on page 169 have been updated to describe how to identify operands or variables on a statement, describe the limitations of this support, and show a new example.

- In the *Debug Tool User’s Guide*, the topic “Quick Start guide for compiling and assembling programs for use with IBM Problem Determination Tools products” which helps you choose the compiler options that work for all Problem Determination Tools, has been added.

- In *Debug Tool Customization Guide*, all of the EQAOPTS options have been organized into one topic: “Defining EQAOPTS options: checklist and instructions” This will help you keep track of the changes you are making to EQAOPTS so that you can make those changes at one time.
Chapter 1. Debug Tool run-time options

This topic describes the run-time options that you can use to control the operation of Debug Tool.

"Table 10" in Debug Tool 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 Debug Tool User’s Guide to select the method that works best for your site.

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

When you specify run-time options for a Language Environment program, they are handled by Language Environment and the following rules apply:
- You can mix them with other Language Environment run-time options in any order.
- Separate them with either blanks or commas.
- Separate all run-time options from user-program options with a slash (’/’).
- The placement of these options (before or after the slash) depends on the programming language of the MAIN routine.

When you specify run-time options for a non-Language Environment program by using EQANMDBG under z/OS batch or TSO, Debug Tool processes the options and the following rules apply:
- You must specify the name of the program to be debugged as the first parameter; this is a positional parameter.
- Specify the run-time options in any order following the name of the program to be debugged.
- Separate all options with commas.
- Separate the run-time options from user-program options with a slash (’/’). If you do not specify any run-time options, the slash follows the name of the program.
- Specify any parameters to the user-program after the slash.
- If no user-program parameters are required, you can omit the slash.

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

Related tasks
"Planning your debug session" in Debug Tool User’s Guide

Related references
z/OS Language Environment Programming Reference
Non-Language Environment positional parameter

If you use EQANMDBG to start Debug Tool to debug MVS batch or TSO programs that do not run in Language Environment, the first positional parameter must be the name of the program you want to debug. This name must be immediately followed by one of the following options:

- one or more of the Debug Tool keyword run-time options described in the following sections of this chapter and then a slash (‘/’) and any user-program parameters
- a slash (‘/’) and any user-program parameters

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

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

**Related tasks**

- “Planning your debug session” in [Debug Tool User’s Guide](#)

COUNTRY run-time option

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

The syntax for this option is:

```
```

```country_code```

A valid country code, one of:

- US United States of America
- JP Japan

NATLANG run-time option

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

The syntax for this option is:

```
```

```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 Debug Tool displays messages in the correct format.
TEST run-time option

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

This topic describes the TEST run-time option and its suboptions. The suboptions of the TEST run-time option control how, when, and where Debug Tool gains control of your program. For a description of how to specify the TEST run-time option, refer to the "Planning your debug session" in Debug Tool User’s Guide.

Syntax of the TEST run-time option

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

The syntax for this option is:

```
NOTEST
TEST
test_level, commands_file,
prompt_level, preferences_file
```

test_level:

```
ALL
ERROR
NONE
```

commands_file:

```
*, commands_file_designator
VADSCPnnnnn
```

prompt_level:

```
PROMPT
NOPROMPT
```
preferences_file:

%

VTAM

user_id:

(1)

TCPIP

tcip_workstation_id:

(1)

VADTCP

%8001

VADTCP

port_id:

%8001

Notes:

1. Specifies remote debug mode.

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

NOTEST

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

TEST

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

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

test_level:

ALL (or blank)

Specifies that the occurrence of an attention interrupt, termination of your program (either normally or through an ABEND), or any program or Language Environment condition of Severity 1 and above causes Debug Tool to gain control, regardless of whether a breakpoint is defined for that type of condition.

When a FINISH, CEE066 or CEE067 thread termination condition is raised by Language Environment, your system administrator can prevent Debug Tool from stopping at this condition by specifying the THREADTERMCOND option in the EQAOPTS option file. Contact your system administrator to determine if this option was specified.
If a condition occurs and a breakpoint exists for the condition, the commands specified in the breakpoint are executed. If a condition occurs and a breakpoint does not exist for that condition, or if an attention interrupt occurs, Debug Tool does the following:

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

**ERROR**

Specifies that only the following conditions cause Debug Tool to gain control without a user-defined breakpoint.

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

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

**NONE**

Specifies that Debug Tool gains control from a condition only if a breakpoint is defined for that condition. If a breakpoint exists for the condition, the commands specified in the breakpoint are executed. An attention interrupt does not cause Debug Tool to gain control unless Debug Tool was started. To change the TEST level after you start your debug session, use the SET TEST command.

**commands_file:**

* (or blank)

Indicates that no commands file is supplied. The terminal, if available, is used as the source of Debug Tool commands.

**commands_file_designator**

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

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

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

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

The commands_file_designator has a maximum length of 80 characters.

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

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

The primary commands file is shared across multiple enclaves.

VADSCPyyyyy
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 VADSCPyyyyy suboption to override the Debug Tool default code page (037). Consult with your system programmer to determine if he implemented the CODEPAGE option 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 VADSCPyyyyy suboption specifying that code page.

The following examples show how to use VADSCPyyyyy:

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

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

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

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

If a CODEPAGE option exists, the code page specified in the CODEPAGE option overrides the CCSID specified in VADSCPyyyyy.

If neither the CODEPAGE option or the VADSCPyyyyy option are specified, the default code page is US code page (037).

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

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

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

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

If you include a STEP command or GO command in your command string, none of the subsequent commands are processed.

The use of a command in prompt_level is not supported in remote debug mode.

preferences_file:

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

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

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

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

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

Specifies the user ID that you used to log on to the Debug Tool Terminal Interface Manager. See the entry for VTAM_LU_id for more information.

INSPPREF **(or blank)**

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

preferences_file_designator

A valid DD name or data set designation specifying the preferences file to use. This file is read the first time Debug Tool is started and must contain a sequence of Debug Tool commands to be executed.

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

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

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

* Specifies that no preferences file is supplied.

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

TCPIP& or VADTCPPIP& **(remote debug mode only)**

Specifies that Debug Tool start in remote debug mode and connect with one of the following remote debuggers:

- Compiled Language Debugger component of Rational Developer for System z
- Compiled Language Debugger component of WebSphere Developer for zSeries
- Compiled Language Debugger component of WebSphere Developer for System z
- WebSphere Developer Debugger for zSeries
- WebSphere Developer Debugger for System z

tcpip_workstation_id **(remote debug mode only)**

TCP/IP name or address of the workstation 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 (remote debug mode only)

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

- Compiled Language Debugger component of Rational Developer for System z
- Compiled Language Debugger component of WebSphere Developer for zSeries
- Compiled Language Debugger component of WebSphere Developer for System z
- WebSphere Developer Debugger for zSeries
- WebSphere Developer Debugger for System z

If you changed the default TCP/IP port settings used by these remote debuggers, you must specify the new number as the port ID in your TEST run-time 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:')`.

**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**

- [Debug Tool User’s Guide](#)

---

**TRAP run-time option**

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

The syntax for this option is:

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

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

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

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

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

Related references

- “block_name” on page 12
- “block_spec” on page 12
- “compile_unit_name” on page 13
- “cu_spec” on page 14
- “expression” on page 14
- “load_module_name” on page 15
- “load_spec” on page 15
- “offset_spec” on page 16
- “references” on page 16
- “statement_id” on page 17
- “statement_id_range and stmt_id_spec” on page 17
- “statement_label” on page 18

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, Debug Tool assumes that address references 64-bit addressable storage. If address contains 7 or 8 significant hexadecimal digits, Debug Tool assumes that address references 31-bit addressable storage. Otherwise, Debug Tool 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 non-Language Environment 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 ("ặt") or apostrophes (') if it is case sensitive. If the name is not inside quotation marks or apostrophes, Debug Tool 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.

\[ block_spec \]

\[ block_name \]

\[ %BLOCK \]

\[ := \]

\[ block_name \]

\[ cu_spec \]

\[ := \]

\[ block_name \]

block_name
- Name of the block. See “block_name.”

%BLOCK
- Represents the currently qualified block. See Chapter 7, “Debug Tool variables,” on page 287.

cu_spec
- A valid compile unit specification; see “cu_spec” on page 14.

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
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 non-Language Environment COBOL program
- A COBOL program or class
- The external procedure name of a PL/I for MVS program
- The package statement or the name of the main procedure, for an Enterprise PL/I program compiled with 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 (‘), Debug Tool 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 (‘):
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 7, “Debug Tool variables,” on page 287

equation

An expression is a combination of references and operators that result in a value. For example, it can be a single constant, a program, session, or Debug Tool variable, a built-in function reference, or a combination of constants, variables, and built-in function references, or operators and punctuation (such as parentheses).
Particular rules for forming an expression depend on the current programming language setting and what release level of the language run-time library under which Debug Tool is running. For example, if you upgrade your version of the HLL compiler without upgrading your version of Debug Tool, certain application programming interface inconsistencies might exist.

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

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

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

---

**load_module_name**

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

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

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

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

---

**load_spec**

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

![load spec diagram]

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*”

%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*”
- Chapter 7, “Debug Tool variables,” on page 287
offset_spec

An offset_spec identifies an offset specification.

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 14

references

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

The specification of a qualified reference includes all containing structures and blocks as qualifiers, and can optionally begin with a load module name qualifier. For example, when the current programming language setting is C, mod::>cu::>proc::>struc1.struc2.array[23]. However, in assembler, disassembly, and non-Language Environment 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, Debug Tool does not support the use of a qualified reference that includes block_spec, cu_spec, or load_spec.

If you are debugging a program compiled with one of the following compilers and running in the following environment, Debug Tool does support the use of a qualified reference that includes block_spec, cu_spec, or load_spec:

- Enterprise PL/I 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 Debug Tool is at an entry to a block, you can not list or reference any variable or expression that includes variables declared in the block being entered.

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

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

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

**statement_id**

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

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

**statement_id_range and stmt_id_spec**

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

```
stmt_id_spec: [block_spec:] stmt_id [:%LINE | %STATEMENT] 
```
**block_spec**

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

**cu_spec**

A valid compile unit specification; see “cu_spec” on page 14. The default is the currently qualified compile unit.

**statement_id**

A valid statement identifier number; see “statement_id” on page 17.

**%LINE**

Represents the currently suspended source statement or line. See Chapter 7, “Debug Tool variables,” on page 287. %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 14
- “statement_id” on page 17
- Chapter 7, “Debug Tool variables,” on page 287

### 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

Debug Tool 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/1 or C/C++ program.
Chapter 3. Syntax for assembler and disassembly expressions

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

Assembler expressions can be written in the following forms:

- A standard assembler expression with an implied length. The following are three examples:
  - X
  - 133
  - X+15

- A standard assembler expression without an implied length. Expressions can be written in this form only if the length can be specified or derived from an operand. For example: R3->+X'2C'

- A conditional assembler expression which is written with conditional operators and can be used only as the operand of an IF command. For example: X+1=Y & Z=4

Common syntax elements

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

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

**dd.dd, dd.dEdd, dd.dE+dd, dd.dE-dd, dd.dE/dd, dd.dE-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'T$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, Debug Tool defines the symbol as a constant, not as a register. For example, if you define the symbol R7 by using the instruction R7 EQU 7 and you never reference R7 or the first reference is not as a register, Debug Tool defines the symbol R7 as the constant 7, not as register R7.
Debug Tool implicitly defines the following symbols in all disassembly compilation units and in any assembler compilation units where the symbol is not already defined:

- R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15. These symbols are implicitly defined as Debug Tool 32-bit 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.

- RG0, RG1, RG2, RG3, RG4, RG5, RG6, RG7, RG8, RG9, RG10, RG11, RG12, RG13, RG14, RG15. These symbols are implicitly defined as Debug Tool 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:

  ```
  _STORAGE(address::length)
  ```

  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 Debug Tool variable. For example: %ADDRESS

## Operators

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

### Operators that can be used in any expression

Use the operators defined in this section to write assembler expressions.

- **Addition**
- **Subtraction or prefix minus**
- **Multiplication**
- **Division**
- **Remainder**
- **Concatenation (C and X-type operands only)**
- **Bitwise AND**
- **Bitwise OR**
- **Parenthesis to control the order of operation, specify the subscript of an array, or select a substring.**
Symbol(subscript)
Parenthesis to specify a subscript for an array. For example, if an array is defined by the instruction X DS 5F, you can specify the first word in the array as X(1).

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

Symbol(substrstart:substrrend)
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,string), Symbol(subscript,substrstart:substrrend), or Symbol(subscript,substrstart:substrlen).

\texttt{\textasciitilde, \textasciitilde, \%, or \%}
Indirection operator. You can use an indirection operator as follows:

\texttt{operand1<indirection-operator>operand2}
Use the contents of operand1 as the base address of the DSECT which contains operand2. For example, R1->DCBDDNAME instructs Debug Tool to use the contents of register 1 as the base address of the DSECT which contains DCBDDNAME.

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

If the indirection operator is not followed by a symbol, no length is implied. This form 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:

\texttt{-}\texttt{> Use the current Amode specification.}
\texttt{==}\texttt{> Use a 64-bit address specification.}
\texttt{=> Use a 31-bit address specification.}
\texttt{\% Use a 24-bit address specification.}

\texttt{.}
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.
- **l** 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, Debug Tool evaluates the 64-bit arithmetic expressions but you can not access the 64-bit General Purpose Registers.
Chapter 4. Syntax for non-Language Environment COBOL expressions

You can use the syntax defined in this section to write expressions for Debug Tool commands while you debug non-Language Environment COBOL programs.

In general, whenever you enter a non-Language Environment 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 non-Language Environment 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 Debug Tool commands that can be used for debugging non-Language Environment 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 non-Language Environment COBOL program you might use the following command:

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

Restrictions on non-Language Environment COBOL expressions

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

- The following operators are supported by Debug Tool in non-Language Environment COBOL expressions:
  - IN or OF
  - Subscript / index
  - LENGTH OF
  - +, -, *, /
  - // (remainder)
  - || (concatenation)
  - ( )
- In a subscript or index list, the subscript or index expressions must be separated by a comma. A space is not sufficient for separating subscript or index expressions.
- Lower-case letters are accepted in contexts other than non-numeric literals as a substitute for (and equivalent to) upper case letters.
- Debug Tool does not support the use of COBOL special registers (for example, DAY, DATE, and TIME) in non-Language Environment 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 non-Language Environment COBOL.
- Only the following subset of figurative constants are supported in Debug Tool non-Language Environment COBOL expressions:
  - HIGH-VALUE, HIGH-VALUES
Common syntax elements

You can use the following syntax elements to write a non-Language Environment COBOL expression:

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

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

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

**symbol**
A valid symbol used in the non-Language Environment COBOL source program. Examples:
LASTNAME
USERVAR8
12CENTS

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

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

Operators

You can use the operators defined in this section to write non-Language Environment COBOL expressions and conditional non-Language Environment COBOL expressions.

Operators that can be used in any expression
Use the operators defined in this section to write non-Language Environment 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:substrtrend)
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:substrtrend), or symbol(subscript,substrstart::substrlen).

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

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

= Compare the two operands for equality.

≠= Compare the two operands for inequality.

< Determines whether the left operand is less than the right operand.

> Determines whether the left operand is greater than the right operand.

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

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

& Logical "and" operation.

| Logical "or" operation.
Chapter 5. Debug Tool commands

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

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

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

The table below summarizes the Debug Tool commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“? command” on page 33</td>
<td>Displays all Debug Tool commands in the Log window.</td>
</tr>
<tr>
<td>“ALLOCATE command” on page 33</td>
<td>Allocates a file to an existing data set, a concatenation of existing data sets, or a temporary data set.</td>
</tr>
<tr>
<td>“ANALYZE command (PL/I)” on page 34</td>
<td>Displays the process of evaluating an expression and the data attributes of any intermediate results.</td>
</tr>
<tr>
<td>“Assignment command (assembler and disassembly)” on page 35</td>
<td>Assigns the value of an expression to a specified storage location or register.</td>
</tr>
<tr>
<td>“Assignment command (non-Language Environment COBOL)” on page 38</td>
<td>Assigns the value of an expression to a specified reference.</td>
</tr>
<tr>
<td>“Assignment command (PL/I)” on page 38</td>
<td>Assigns the value of an expression to a specified reference.</td>
</tr>
<tr>
<td>“AT command” on page 39</td>
<td>Defines a breakpoint (gives control of your program to Debug Tool under the specified circumstances).</td>
</tr>
<tr>
<td>“BEGIN command” on page 76</td>
<td>BEGIN and END delimit a sequence of one or more commands to form one longer command.</td>
</tr>
<tr>
<td>“block command (C and C++)” on page 77</td>
<td>Allows you to group any number of Debug Tool commands into one command.</td>
</tr>
<tr>
<td>“break command (C and C++)” on page 77</td>
<td>Allows you to terminate and exit a loop (that is, do, for, and while) or switch command from any point other than the logical end.</td>
</tr>
<tr>
<td>“CALL command” on page 78</td>
<td>The CALL command calls either a procedure, entry name, or program name, or it requests that a utility function be run.</td>
</tr>
<tr>
<td>“CLEAR command” on page 88</td>
<td>Removes the actions of previously issued Debug Tool commands (such as breakpoints).</td>
</tr>
<tr>
<td>“COMMENT command” on page 93</td>
<td>Used to insert commentary into the session log.</td>
</tr>
<tr>
<td>“COMPUTE command (COBOL)” on page 93</td>
<td>Assigns the value of an arithmetic expression to a specified reference.</td>
</tr>
<tr>
<td>“CURSOR command (full-screen mode)” on page 94</td>
<td>Moves the cursor between the last saved position on the Debug Tool session panel (excluding the header fields) and the command line.</td>
</tr>
<tr>
<td>Topic</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Assembler, disassembly, and non-Language Environment COBOL</td>
<td>Declares session variables that are effective during a Debug Tool session.</td>
</tr>
<tr>
<td>Declarations (C and C++)</td>
<td>Declares session variables and tags that are effective during a Debug Tool session.</td>
</tr>
<tr>
<td>Declarations (COBOL)</td>
<td>Declares session variables that are effective during a Debug Tool session.</td>
</tr>
<tr>
<td>DECLARE command (PL/I)</td>
<td>Declares session variables that are effective during a Debug Tool session.</td>
</tr>
<tr>
<td>DESCRIBE command</td>
<td>Displays the attributes of references, compile units, and the execution environment.</td>
</tr>
<tr>
<td>DISABLE command</td>
<td>Makes the AT breakpoint inoperative, but does not clear it; you can ENABLE it later without typing the entire command again.</td>
</tr>
<tr>
<td>do/while command (C and C++)</td>
<td>Performs a command before evaluating the test expression.</td>
</tr>
<tr>
<td>DO command (PL/I)</td>
<td>Allows one or more commands to be collected into a group which can (optionally) be run repeatedly.</td>
</tr>
<tr>
<td>ENABLE command</td>
<td>Makes AT breakpoints operative after they have been disabled by the DISABLE command.</td>
</tr>
<tr>
<td>EVALUATE command (COBOL)</td>
<td>Provides a shorthand notation for a series of nested IF statements.</td>
</tr>
<tr>
<td>Expression command (C and C++)</td>
<td>Evaluates the given expression which can be used to either assign a value to a variable or to call a function.</td>
</tr>
<tr>
<td>FIND command</td>
<td>Provides full-screen and line mode searching of source and listing files, and full-screen searching of Log and Monitor windows.</td>
</tr>
<tr>
<td>FINDBP command</td>
<td>Provides full-screen searching of the source for line, statement, and offset breakpoints.</td>
</tr>
<tr>
<td>for command (C and C++)</td>
<td>Provides iterative looping.</td>
</tr>
<tr>
<td>FREE command</td>
<td>Frees (deallocates) an allocated file.</td>
</tr>
<tr>
<td>GO command</td>
<td>Causes Debug Tool to start or resume running your program.</td>
</tr>
<tr>
<td>GOTO command</td>
<td>Causes Debug Tool to resume program execution at the specified statement id.</td>
</tr>
<tr>
<td>GOTO LABEL command</td>
<td>Causes Debug Tool to resume running program at the specified statement label.</td>
</tr>
<tr>
<td>%IF command (programming language neutral)</td>
<td>Lets you conditionally perform a command; use this syntax if you are constructing a command that might run in different programming languages.</td>
</tr>
<tr>
<td>IF command (assembler, disassembly, and non-Language Environment COBOL)</td>
<td>Lets you conditionally perform a command.</td>
</tr>
<tr>
<td>if command (C and C++)</td>
<td>Lets you conditionally perform a command.</td>
</tr>
<tr>
<td>IF command (COBOL)</td>
<td>Lets you conditionally perform a command.</td>
</tr>
<tr>
<td>IF command (PL/I)</td>
<td>Lets you conditionally perform a command.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&quot;IMMEDIATE command (full-screen mode)&quot; on page 136</td>
<td>Causes a command within a command list to be performed immediately. For use with commands assigned to a PF key.</td>
</tr>
<tr>
<td>&quot;INPUT command (C, C++, and COBOL)&quot; on page 137</td>
<td>Provides input for an intercepted read and is valid only when there is a read pending for an intercepted file.</td>
</tr>
<tr>
<td>&quot;JUMPTO command&quot; on page 138</td>
<td>Jumps to the specified statement and then stops the program at that statement.</td>
</tr>
<tr>
<td>&quot;LIST command&quot; on page 141</td>
<td>Displays information about your Debug Tool session.</td>
</tr>
<tr>
<td>&quot;LOAD command&quot; on page 162</td>
<td>Specifies that the named module should be loaded for debugging purposes.</td>
</tr>
<tr>
<td>&quot;LOADDEBUGDATA command&quot; on page 164</td>
<td>Specifies that a compile unit (CU) as an assembler CU and loads debug data.</td>
</tr>
<tr>
<td>&quot;MEMORY command&quot; on page 165</td>
<td>Identifies an address in memory to display in the Memory window.</td>
</tr>
<tr>
<td>&quot;MONITOR command&quot; on page 166</td>
<td>Defines or redefines a command whose output is displayed in the Monitor window (full-screen mode), terminal output (line mode), or log file (batch mode).</td>
</tr>
<tr>
<td>&quot;MOVE command (COBOL)&quot; on page 171</td>
<td>Transfers data from one area of storage to another.</td>
</tr>
<tr>
<td>&quot;NAMES command&quot; on page 175</td>
<td>Specify names of load modules or compile units to debug or ignore, and display the current setting of the NAMES command.</td>
</tr>
<tr>
<td>&quot;Null command&quot; on page 178</td>
<td>A semicolon written where a command is expected.</td>
</tr>
<tr>
<td>&quot;ON command (PL/I)&quot; on page 178</td>
<td>Establishes the actions to be executed when the specified PL/I condition is raised.</td>
</tr>
<tr>
<td>&quot;PANEL command (full-screen mode)&quot; on page 180</td>
<td>Displays special panels (for example, to customize your full-screen session).</td>
</tr>
<tr>
<td>&quot;PERFORM command (COBOL)&quot; on page 182</td>
<td>Identifies a series of commands to be run. The series of commands can be run repeatedly, if you use the UNTIL keyword of the command.</td>
</tr>
<tr>
<td>&quot;PLAYBACK commands&quot; on page 184</td>
<td>Commands to start and stop recording application execution states and replay the recorded execution states.</td>
</tr>
<tr>
<td>&quot;POPUP command&quot; on page 188</td>
<td>Displays the Command pop-up window, where you type in commands.</td>
</tr>
<tr>
<td>&quot;POSITION command&quot; on page 188</td>
<td>Positions the cursor to a specific line in the specified window.</td>
</tr>
<tr>
<td>&quot;Prefix commands (full-screen mode)&quot; on page 188</td>
<td>Apply only to source listing lines and are typed into the Source window.</td>
</tr>
<tr>
<td>&quot;PROCEDURE command&quot; on page 189</td>
<td>Allows the definition of a group of commands that can be accessed using the CALL procedure command.</td>
</tr>
<tr>
<td>&quot;QUALIFY RESET command&quot; on page 190</td>
<td>Resets qualification to the block of the suspended program and scrolls the Source window to display the current statement line.</td>
</tr>
<tr>
<td>&quot;QUERY command&quot; on page 190</td>
<td>Displays the current value of Debug Tool settings (such as the current location in the suspended program).</td>
</tr>
<tr>
<td>&quot;QUIT command&quot; on page 196</td>
<td>Ends a Debug Tool session (with a return code, if specified).</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&quot;QUIT command&quot; on page 197</td>
<td>Ends a Debug Tool session (without additional prompting)</td>
</tr>
<tr>
<td>&quot;RETRIEVE command (full-screen mode)&quot; on page 199</td>
<td>Displays the last command entered on the command line.</td>
</tr>
<tr>
<td>&quot;RESTORE command&quot; on page 198</td>
<td>Enables explicit restoring of settings, breakpoints, and monitor specifications.</td>
</tr>
<tr>
<td>&quot;RUN command&quot; on page 200</td>
<td>Causes Debug Tool to start or resume running your program.</td>
</tr>
<tr>
<td>&quot;RUNTO command&quot; on page 200</td>
<td>Causes Debug Tool to run your program to a specific point (without setting a breakpoint)</td>
</tr>
<tr>
<td>&quot;SCROLL command (full-screen mode)&quot; on page 201</td>
<td>Provides horizontal and vertical scrolling in full-screen mode.</td>
</tr>
<tr>
<td>&quot;SELECT command (PL/I)&quot; on page 204</td>
<td>Chooses one of a set of alternate commands.</td>
</tr>
<tr>
<td>&quot;SET command&quot; on page 205</td>
<td>Controls various Debug Tool settings.</td>
</tr>
<tr>
<td>&quot;SET command (COBOL)&quot; on page 260</td>
<td>Assigns a value to a COBOL reference.</td>
</tr>
<tr>
<td>&quot;SHOW prefix command (full-screen mode)&quot; on page 263</td>
<td>Specifies what relative statement (for C) or relative verb (for COBOL) within the line is to have its frequency count temporarily shown in the suffix area.</td>
</tr>
<tr>
<td>&quot;STEP command&quot; on page 263</td>
<td>Causes Debug Tool to dynamically step through a program, running one or more program statements.</td>
</tr>
<tr>
<td>&quot;STORAGE command&quot; on page 266</td>
<td>Enables you to alter up to eight bytes of storage.</td>
</tr>
<tr>
<td>&quot;switch command (C and C++)&quot; on page 268</td>
<td>Enables you to transfer control to different commands within the switch body, depending on the value of the switch expression.</td>
</tr>
<tr>
<td>&quot;SYSTEM command (z/OS)&quot; on page 270</td>
<td>Lets you issue TSO commands during a Debug Tool session.</td>
</tr>
<tr>
<td>&quot;TRIGGER command&quot; on page 271</td>
<td>Raises the specified AT condition in Debug Tool, or raises the specified programming language condition in your program.</td>
</tr>
<tr>
<td>&quot;TSO command (z/OS)&quot; on page 274</td>
<td>Lets you issue TSO commands during a Debug Tool session (this command is valid only in a TSO environment).</td>
</tr>
<tr>
<td>&quot;USE command&quot; on page 274</td>
<td>Causes the Debug Tool commands in the specified file or data set to be either performed or syntax checked.</td>
</tr>
<tr>
<td>&quot;while command (C and C++)&quot; on page 276</td>
<td>Enables you to repeatedly perform the body of a loop until the specified condition is no longer met or evaluates to false.</td>
</tr>
<tr>
<td>&quot;WINDOW command (full-screen mode)&quot; on page 276</td>
<td>Opens, close, resizes, or expands to full screen (zooms) the specified window on the Debug Tool session panel.</td>
</tr>
</tbody>
</table>

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

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

**Related references**
- Chapter 6, “Debug Tool built-in functions,” on page 281
- Chapter 7, “Debug Tool variables,” on page 287
? command

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

Usage note

In the following cases, Debug Tool does not display the syntax help after you enter the ? command:
- The Debug Tool SYSTEM and TSO commands followed by the ? command do not display the syntax help; instead the ? is sent to the host as part of the system command.
- The COMMENT command followed by the ? command does not display the syntax help.
- The SET PFx command accepts a ? as the “command” operand and, in this case, does not display syntax help.

ALLOCATE command

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

ALLOCATE FILE ddname attributes ;

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 102

---

**ANALYZE command (PL/I)**

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

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

**EXPRESSION**
Requests that the accompanying *expression* be evaluated from the following points of view:
- What are the attributes of each element during the evaluation of the expression?
- What are the dimensions and bounds of the elements of the expression, if applicable?
- What are the attributes of any intermediate results that will be created during the processing of the expression?

*expression*  
A valid Debug Tool PL/I expression.

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

**Example**
This example is based on the following program segment:

```plaintext
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:

```plaintext
ANALYZE EXPRESSION ( (hi_point - lo_point) + offset / percent )
```

<table>
<thead>
<tr>
<th>HI_POINT</th>
<th>LO_POINT</th>
<th>OFFSET</th>
<th>PERCENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIXED BINARY(31,3) REAL</td>
<td>FIXED BINARY(31,5) REAL</td>
<td>FIXED DECIMAL(12,2) REAL</td>
<td>CHARACTER(12)</td>
</tr>
<tr>
<td>HI_POINT - LO_POINT</td>
<td>OFFSET / PERCENT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIXED BINARY(31,17) REAL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 258
- “PLAYBACK commands” on page 184

---

### Assignment command (assembler and disassembly)

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

```plaintext
receiver = sourceexpr;
```

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

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

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

**Assignment rules**

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

The following table shows how the assignment type is determined from the source and receiver data types. In this table, the following definitions are used:

? Indicates an unknown type, for example, R1->+2.
* Indicates any type or length.

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

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

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

**Hex Float**
Hexadecimal floating point assignment.

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

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

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

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

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

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

All others Error

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 D$ 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 prog_name variable the value of the character string ’MYPROG’.
  prog_name = 'MYPROG' ;
- Assign the value of X to the 4 bytes at offset 8 from the contents of R8.
  %R8->+8 <1’x’> = x;
- Move a string of 14 bytes pointed to by the contents of R8 (where R8 was an equated register used in the program) to 6 bytes past the location pointed to by R2.
  %R2->+6 <14> = R8->+0;
- Set 32 bytes pointed to by R6 to zero.
  %R6->+0 <X’20’> = X’00’;
Assignment command (non-Language Environment 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 Debug Tool non-Language Environment COBOL reference enclosed in apostrophes (').

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

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

Assignment command (PL/I)

The Assignment command assigns the value of an expression to a specified reference.
reference
A valid Debug Tool PL/I reference.

eexpression
A valid Debug Tool PL/I expression.

Usage notes
• The PL/I repetition factor is not supported by Debug Tool.
  For example, the following is not valid: rx = (16)'01'B;
• If Debug Tool was started because of a computational condition or an attention
  interrupt, using an assignment to set a variable might not give the expected
  results. This is because Debug Tool cannot determine variable values within
  statements, only at statement boundaries.
• The PL/I assignment statement option BY NAME is not valid in the Debug Tool.
• If you are debugging a Enterprise PL/I program, the target of an assignment
  command can not be the variables %EPRn, %FPRn, %GPRn, or %LPRn.
• The Assignment command cannot be used while you replay recorded statements
  by using the PLAYBACK commands.

Examples
• Assign the value 6 to variable x.
  x = 6;
• Assign to the Debug Tool variable %GPR5 the address of name_table.
  %GPR5 = ADDR (name_table);
• Assign to the prg_name variable the value of Debug Tool 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 16
  "PLAYBACK commands" on page 184

AT command

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

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

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

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

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

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

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

The following table summarizes the forms of the AT command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT ALLOCATE (PL/I) command on page 43</td>
<td>Gives Debug Tool control when storage for a named controlled variable or aggregate is dynamically allocated by PL/I.</td>
</tr>
</tbody>
</table>
| AT APPEARANCE command on page 44 | Gives Debug Tool control:  
- For C and PL/I, when the specified compile unit is found in storage  
- For COBOL, the first time the specified compile unit is called |
<p>| AT CALL command on page 46 | Gives Debug Tool control on an attempt to call the specified entry point. |
| AT CHANGE command (full screen mode, line mode, batch mode) on page 48 | Gives Debug Tool control when either the specified variable value or storage location is changed. |</p>
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT CHANGE command (remote debug mode)* on page 53</td>
<td>Gives Debug Tool control when the specified variable value is changed.</td>
</tr>
<tr>
<td>AT CURSOR command (full-screen mode)* on page 55</td>
<td>Defines a statement breakpoint by cursor pointing.</td>
</tr>
<tr>
<td>AT DATE command (COBOL)* on page 56</td>
<td>For COBOL, gives Debug Tool control for each date processing statement within the specified block.</td>
</tr>
<tr>
<td>AT DELETE command on page 57</td>
<td>Gives Debug Tool control when a load module is deleted.</td>
</tr>
<tr>
<td>AT ENTRY command* on page 58 or &quot;AT ENTRY command (remote debug mode)* on page 59</td>
<td>Defines a breakpoint at the specified entry point.</td>
</tr>
<tr>
<td>AT EXIT command* on page 60</td>
<td>Defines a breakpoint at the specified exit point.</td>
</tr>
<tr>
<td>AT GLOBAL command* on page 61</td>
<td>Gives Debug Tool control for every instance of the specified AT-condition.</td>
</tr>
<tr>
<td>AT LABEL command on page 62</td>
<td>Gives Debug Tool control at the specified statement label.</td>
</tr>
<tr>
<td>AT LINE command on page 65</td>
<td>Gives Debug Tool control at the specified line.</td>
</tr>
<tr>
<td>AT LOAD command* on page 65 or &quot;AT LOAD command (remote debug mode)* on page 67</td>
<td>Gives Debug Tool control when the specified load module is loaded.</td>
</tr>
<tr>
<td>AT OCCURRENCE command* on page 67</td>
<td>Gives Debug Tool control on a language or Language Environment condition or exception.</td>
</tr>
<tr>
<td>AT OFFSET command (disassembly)* on page 70</td>
<td>Gives Debug Tool control at the specified offset in the disassembly view.</td>
</tr>
<tr>
<td>AT PATH command* on page 71</td>
<td>Gives Debug Tool control at a path point.</td>
</tr>
<tr>
<td>AT Prefix command (full-screen mode)* on page 72</td>
<td>Defines a statement breakpoint through the Source window prefix area.</td>
</tr>
<tr>
<td>AT STATEMENT command* on page 73 or &quot;AT STATEMENT command (remote debug mode)* on page 75</td>
<td>Gives Debug Tool control at the specified statement.</td>
</tr>
<tr>
<td>AT TERMINATION command* on page 75</td>
<td>Gives Debug Tool control when the application program is terminated.</td>
</tr>
</tbody>
</table>

**Usage notes**

- To set breakpoints at specific locations in a program, Debug Tool depends on that program being loaded into storage. If you issue an AT command for a specific EXIT, LABEL, LINE, or STATEMENT breakpoint and the program is not known by Debug Tool, a warning message is issued and the breakpoint is not set. For ENTRY, the breakpoint becomes a deferred breakpoint.
• To set a global breakpoint, you can specify an asterisk (*) with the AT command or you can specify an AT GLOBAL command. For example, if you want to set a global AT ENTRY breakpoint, specify:

    AT ENTRY *;
    or
    AT GLOBAL ENTRY;

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

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

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

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

• If you want breakpoints to 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**

- Debug Tool User’s Guide

**Related references**

- "LIST command" on page 141

### every_clause syntax

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

The syntax for `every_clause` is:

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

**EVERY integer**

Specifies how frequently the breakpoint is taken. For example, EVERY 5 means that Debug Tool is started every fifth time the AT-condition is met. The default is EVERY 1.
FROM integer
Specifies when Debug Tool invocations are to begin. For example, FROM 8 means that Debug Tool is not started until the eighth time the AT-condition is met. If the FROM value is not specified, its value is equal to the EVERY value.

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

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

Examples
- Break every third time statement 50 is reached, beginning with the 48th time and ending after the 59th time. The breakpoint action is performed the 48th, 51st, 54th, and 57th time statement 50 is reached.
  AT EVERY 3 FROM 48 TO 59 STATEMENT 50;
- At the fifth change of structure field member of the structure named mystruct, print a message saying that it has changed and list its new value. In addition, clear the CHANGE breakpoint. The current programming language setting is C.
  AT FROM 5 CHANGE mystruct.member {
    LIST ("mystruct.member has changed.
         It is now", mystruct.member);
    CLEAR AT CHANGE mystruct.member;
  }

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

AT every_clause ALLOCATE identifier command;

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

* Sets a breakpoint at every ALLOCATE.

command
A valid Debug Tool command.

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

- When the major structure area_name is allocated, display the address of the storage that was obtained.
  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 42
- "PLAYBACK commands" on page 184

AT APPEARANCE command

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

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

command
  A valid Debug Tool command.

Usage notes

- If this breakpoint is set in a parent enclave it can be triggered and operated on with breakpoint commands while the application is in a child enclave.
- If the compile unit is qualified with a load module name, the AT APPEARANCE breakpoint will only be recognized for the compile unit that is contained in the specified load module. For example, if a compile unit cux that is in load module loady appears, the breakpoint AT APPEARANCE loady::>: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.

• Debug Tool gains control when the specified compile unit is first recognized by Debug Tool. This can occur when a program is reached that contains a reference to that compile unit. This occurs late enough that the program can be operated on (setting breakpoints, for example), but early enough that the program has not yet been executed. In addition, for C, static variables can also be referenced.

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

• AT APPEARANCE is helpful when setting breakpoints in unknown compile units. You can set breakpoints at locations currently unknown to Debug Tool by using the proper qualification and embedding the breakpoints in the command list associated with an APPEARANCE breakpoint. However, there can be only one APPEARANCE breakpoint set at any time for a given compile unit and you must include all breakpoints for that unknown compile unit in a single APPEARANCE breakpoint.

• For a non-CICS application, the AT APPEARANCE breakpoint is cleared at the end of a process.

• Before you enter the AT APPEARANCE command while you debug an assembler or disassembled program, enter the SET ASSEMBLER ON or SET DISASSEMBLY ON command.

• For C and C++ only: AT APPEARANCE is not triggered for compile units that reside in a loaded module because the compile units are known at the time of the load.

• For C, C++, and PL/I only: An APPEARANCE breakpoint is triggered when Debug Tool finds the specified compile unit in storage. To be triggered, however, the APPEARANCE breakpoint must be set before the compile unit is loaded.

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

• For COBOL only: An APPEARANCE breakpoint is triggered when Debug Tool finds the specified compile unit in storage. To be triggered, however, the APPEARANCE breakpoint must be set before the compile unit is called.

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

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

Examples

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

  AT APPEARANCE cu {
    AT ENTRY a;
    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 42
  * "cu_spec” on page 14
  * “PLAYBACK commands” on page 184

AT CALL command

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

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

`entry_name`

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

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

`command`

A valid Debug Tool command.

Usage notes

* AT CALL intercepts the call itself, not the subroutine entry point. C, COBOL, and
  PL/I programs compiled with the 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 Debug Tool at compile time.
  Calls to entry variables are not intercepted, except when the current
  programming language setting is either C or COBOL (compiled with the TEST
  run-time option).

* AT CALL 0 intercepts calls to unresolved entry points when the current
  programming language setting is C or PL/I (compiled with the TEST run-time
  option).

* AT CALL allows you to intercept or bypass the target program by using GO
  BYPASS or GOTO. If resumed by a normal GO or STEP, execution resumes by
  performing the call.
• If you set a breakpoint in a parent enclave, the breakpoint can be triggered and operated on with breakpoint commands while the application is in a child enclave.

• While debugging a CICS application, the breakpoint is cleared at the end of the last process in the CICS application. While debugging a non-CICS application, the breakpoint is cleared at the end of a process.

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

• You cannot use the AT CALL command while you debug a disassembly program.

• Debug Tool does not support the AT CALL command while you debug a non-Language Environment 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 4.1
    - 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:
    - NOHOOK suboption of the TEST compiler option for Enterprise COBOL for z/OS, Version 4.1
    - 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 Debug Tool control at any SVC or PC instruction.

**Examples**

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

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

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

**Related tasks**

[Debug Tool User’s Guide](#)

**Related references**

"every_clause syntax" on page 42

“PLAYBACK commands” on page 184

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

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

```
AT every_clause CHANGE
  reference
  WHEN condition
  %STORAGE (address, length)
  (reference)

condition
  A valid Debug Tool conditional expression.
```
A valid Debug Tool reference in the current programming language.

A valid Debug Tool reference when the current programming language is non-Language Environment COBOL.

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

The starting address of storage to be watched for changes.

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

A valid Debug Tool command. If you are using remote debug mode, you can specify only commands that are supported in remote debug mode.

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 Debug Tool variables %GPRn, %Rn, %FPRn, %LPRn, %EPRn, and any assembler or disassembly symbols representing registers cannot be watched.

Only entire bytes are watched; bits or bit strings within a byte cannot be singled out.

Because AT CHANGE breakpoints are identified by storage address and length, it is not possible to have two AT CHANGE breakpoints for the same area (address and length) of storage. That is, an AT CHANGE command replaces a previous AT CHANGE command if the storage address and length are the same. However, any other overlap is ignored and the breakpoints are considered to be for two separate variables. For example, if the storage address is the same, but the length is different, the AT CHANGE command does not replace the previous AT CHANGE.

When more than one AT CHANGE breakpoint is triggered at a time, AT CHANGE breakpoints are triggered in the order that they were entered. However, if the triggering of one breakpoint causes a variable watched by a different breakpoint to change, the ordering of the triggers will not necessarily be according to when they were originally entered. For example,

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

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

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

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

For the following items, Debug Tool recalculates the address of reference each time it monitors the storage location. If the address of reference changes, Debug Tool 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 level-88 variable on an AT CHANGE command, the current setting of the value is saved. Debug Tool stops at the breakpoint only if the setting of the level-88 variable changes from the saved value to a different value. For example, if the saved value was TRUE and the new value is FALSE, Debug Tool stops at the breakpoint.

• To use a level-88 variable with the AT CHANGE command, you (through a Debug Tool command) or the program must have previously set the variable to one of the values specified in the variable’s declaration. If you do not do this, Debug Tool behavior becomes unpredictable.

• When you use a condition, the variables used in the condition or the condition are not evaluated at the time the breakpoint is set but when the location associated with the AT CHANGE command changes.

• Only the following conditional operators can be used in a condition:
  = Compare the two operands for equality.
  ≠ Compare the two operands for inequality.
  < Determines whether the left operand is less than the right operand.
  > Determines whether the left operand is greater than the right operand.
<= Determines whether the left operand is less than or equal to the right operand.

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

& Logical "and" operation.

| Logical "or" operation.

- If you use the AT CHANGE command with a WHEN condition, every time the variable changes the condition is evaluated. If the condition evaluates to true, Debug Tool stops and runs the command associated with the breakpoint.
- When Debug Tool evaluates the condition and the condition is invalid, Debug Tool does one of the following actions:
  - If SET WARNING is set to ON, Debug Tool stops and displays a message that it could not evaluate the condition. You need to enter a command to indicate what action you want Debug Tool to take.
  - If SET WARNING is set to OFF, Debug Tool does not stop nor display a message that it could not evaluate the condition. Debug Tool continues running the program.
- If you specify address with more than 8 significant digits or if reference references 64-bit addressable storage, Debug Tool assumes that the storage location is 64-bit addressable storage. Otherwise, Debug Tool assumes that the storage location is 31-bit addressable storage.

Examples

- Identify the current location each time variable varb11 or varb12 is found to have a changed value. The current programming language setting is COBOL.
  
  AT CHANGE (varb11, varb12) PERFORM
  QUERY LOCATION;
  GO;
  END-PERFORM;

- When storage at the hex address 22222 changes, print a message in the log. Eight bytes of storage are to be watched. The current programming language setting is C.
  
  AT CHANGE %STORAGE (0x00022222, 8)
  LIST "Storage has changed at hex address 22222";

- Set two breakpoints when storage at the hex address 1000 changes. The variable x is defined at hex address 1000 and is 20 bytes in length. In the first breakpoint, 20 bytes of storage are to be watched. In the second breakpoint, 50 bytes of storage are to be watched. The current programming language setting is C.
  
  AT CHANGE %STORAGE (0x00010000, 20) /* Breakpoint 1 set */
  AT CHANGE %STORAGE (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
AT CHANGE command (remote debug mode)

Gives Debug Tool control when the program changes the specified variable value.

```
AT CHANGE "reference" ;
```

'reference' or "reference"

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

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

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

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

- 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, Debug Tool 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 level-88 variable on an AT CHANGE command, the current setting of the value is saved. Debug Tool stops at the breakpoint only if the setting of the level-88 variable changes from the saved value to a different value. For example, if the saved value was TRUE and the new value is FALSE, Debug Tool stops at the breakpoint.
- To use a level-88 variable with the AT CHANGE command, you (through a Debug Tool command) or the program must have previously set the variable to one of the values specified in the variable’s declaration. If you do not do this, Debug Tool behavior becomes unpredictable.
- If reference references 64-bit addressable storage, Debug Tool assumes that the storage location is 64-bit addressable storage. Otherwise, Debug Tool 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 16
“AT CHANGE command (full screen mode, line mode, batch mode)” on page 48
Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

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.

```
AT CURSOR
TOGGLE
```

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.

```plaintext
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 184
```

**AT DATE command (COBOL)**

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

```
AT DATE every_clause block_spec command
```

* Sets a breakpoint at every date processing statement.

**command**

A valid Debug Tool command.

**Usage notes**

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

**Examples**

- Each time a date processing statement is encountered in the nested subprogram 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 42
- “block_spec” on page 12
- “PLAYBACK commands” on page 184

### AT DELETE command

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

```
AT every_clause DELETE load_spec command;
```

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

**command**

A valid Debug Tool command.

**Usage notes**

- Debug Tool 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, Debug Tool is not notified of deletes affected by the MVS delete service. Refer to [Debug Tool 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 Debug Tool, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on Debug Tool, it is cleared at the end of a process.
- The AT 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 42
- “load_spec” on page 15
- “PLAYBACK commands” on page 184
AT ENTRY command

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

\[ \text{AT} \text{ every\_clause} \text{ ENTRY} \text{ block\_spec} \text{ WHEN} \text{ condition} \]

* Sets a breakpoint at every ENTRY of any block.

command

A valid Debug Tool command. If you are using remote debug mode, you can specify only commands that are supported in remote debug mode.

condition

A valid Debug Tool conditional expression.

Usage notes

- For VS COBOL II programs, Debug Tool supports only the AT ENTRY * command.
- To specify an AT ENTRY breakpoint for a program that is not currently known to Debug Tool, you must do one of the following:
  - If the name of the program is the same as the block_spec, you do not need to qualify the block_spec with the name of the program.
  - If the name of the program is not the same as the block_spec, you need to qualify the block_spec with a program name. When Debug Tool detects a program name that matches the one you specified, it sets the breakpoint.
- An ENTRY 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 Debug Tool, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on Debug Tool, it is cleared at the end of a process.
- 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, Debug Tool 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 Debug Tool reaches the entry, it evaluates the condition. If the condition evaluates to true, Debug Tool stops and runs the command associated with the breakpoint.

When Debug Tool evaluates the condition and the condition is invalid, Debug Tool does one of the following actions:

- If SET WARNING is set to ON, Debug Tool stops and displays a message that it could not evaluate the condition. You need to enter a command to indicate what action you want Debug Tool to take.

- If SET WARNING is set to OFF, Debug Tool does not stop nor display a message that it could not evaluate the condition. Debug Tool continues running the program.

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 42
- “condition” on page 13
- “block_spec” on page 12
- “AT APPEARANCE command” on page 44
- “PLAYBACK commands” on page 184
- Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

**AT ENTRY command (remote debug mode)**

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

```
 AT ENTRY block_spec;
```

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

**Related references**

- “block_spec” on page 12
**AT EXIT command**

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

```
AT every_clause EXIT block_spec command ;
```

* Sets a breakpoint at every EXIT of any block.

**command**

A valid Debug Tool command.

**Usage notes**

- For VS COBOL II programs, Debug Tool 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 Debug Tool, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on Debug Tool, it is cleared at the end of a process.
- EXIT breakpoints for blocks in a fetched or loaded program are removed when that program is released.
- 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 non-Language Environment 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

**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 42
- “block_spec” on page 12
- “PLAYBACK commands” on page 184

**AT GLOBAL command**

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

```
AT GLOBAL
  every_clause
    ALLOCATE
    APPEARANCE
    CALL
    DATE
    DELETE
    ENTRY WHEN condition
    EXIT
    LABEL
    LINE
    LOAD
    PATH
    STATEMENT WHEN condition
```

*command*

A valid Debug Tool command.

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

**Usage notes**

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

**Examples**

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

- At every statement or line, display a message identifying the statement or line.
  The current programming language setting is COBOL.
AT GLOBAL STATEMENT LIST (‘At Statement:’, %STATEMENT);
- If you enter (for COBOL):
  AT EXIT table1 PERFORM
  LIST TITLED (age, pay);
  GO;
  END-PERFORM;

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

only benefits and scale are listed when your program reaches the exit point of
block table1. The second AT EXIT replaces the first because the breakpoints are
defined for the same location. However, if you define the following GLOBAL
breakpoint 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 Debug Tool commands, specify:
  AT GLOBAL DATE QUERY LOCATION;

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

Referenced topics
"every_clause syntax" on page 42
"PLAYBACK commands" on page 184

AT LABEL command

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

**command**

A valid Debug Tool command.

**Usage notes**

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

Either form can be prefixed with the usual block, compile unit, and load module qualifiers.

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

You can set breakpoints for more than one label at the same location. Debug Tool is entered for each specified label.

To be able to set LABEL breakpoints in COBOL 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 Debug Tool sets a breakpoint for each label specified, unless there are several labels on the same statement. In this case, only the last LABEL breakpoint defined is set.

For COBOL, a reference to a label or a label constant can take either of the following forms:

- *name*
  This form is used to refer to a section name or the name of a paragraph contained in not more than one section of the block.

- *name1 OF name2* or *name1 IN name2*
  This form is used to refer to a paragraph contained within a section if the paragraph name exists in other sections in the same block. You can use either OF or IN, but Debug Tool 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.

**Examples**

- Set a breakpoint at label create in the currently qualified block.
  
  AT LABEL create;

- At program label para 0F sect1 display variable names x and y and their values, and continue program execution. The current programming language setting is COBOL.

  AT LABEL para 0F sect1 PERFORM
  LIST TITLED (x, y);
  GO;
  END-PERFORM;
• Set a breakpoint at labels label1 and label2, even though both labels are associated to the same statement. The current programming language setting is C.

AT LABEL label1 LIST 'Stopped at label1'; /* Label1 is first */
AT LABEL label2 LIST 'Stopped at label2'; /* Label2 is second */

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

Related references
“every_clause syntax” on page 42
“statement_label” on page 18
“PLAYBACK commands” on page 184

AT LINE command

Gives Debug Tool control at the specified line.

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

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

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

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

Related references
“AT OFFSET command (disassembly)” on page 70
“PLAYBACK commands” on page 184
“AT STATEMENT command” on page 73

AT LOAD command

Gives Debug Tool control when the specified load module is brought into storage. For example, Debug Tool gains control on completion of a successful C fetch(), a PL/I FETCH, during a COBOL dynamic CALL, MVS LOAD service, or EXEC CICS LOAD.

To stop at a compile unit or program in a COBOL DLL, use AT APPEARANCE. Once the breakpoint is raised for the specified load module, it is not raised again unless either the load module is released and fetched again or another load module with the specified name is fetched.

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

```
* Sets a breakpoint at every LOAD of any load module.
```
A valid Debug Tool command.

Usage notes
- Debug Tool gains control for loads that are affected by the Language Environment load service, the MVS LOAD service, or EXEC CICS LOAD. A LOAD breakpoint is triggered when a new enclave is entered. If the Dynamic Debug facility is deactivated (by entering the SET DYNDEBUG OFF command) or SVC screening is disabled, Debug Tool is not notified of any loads that are affected by the MVS LOAD service. Refer to [Debug Tool Customization Guide](#) for instructions on how to control SVC screening.
- AT LOAD can be used to detect the loading of specific language library load modules; however, the loading of language library load modules does not trigger an AT GLOBAL LOAD or AT LOAD *.
- AT LOAD cannot specify the initial load module because it is already loaded when Debug Tool is started.
- If this breakpoint is set in a parent enclave, it can be triggered and operated on with breakpoint commands while the application is in a child enclave.
- For a CICS application on Debug Tool, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on Debug Tool, it is cleared at the end of a process.
- AT LOAD on an implicitly or explicitly loaded DLL is not supported by Debug Tool.
- Depending on the version of the C or C++ compiler used, Debug Tool might recognize a compile unit in a DLL only after it has had a function in it called. For example, if a DLL contains a function fn1 in CU file1 and it contains a function fn2 in CU file2, a call to fn1 will not enable Debug Tool to recognize file2, only file1. Similarly, a call to fn2 will not enable Debug Tool to recognize file1.
- At the triggering of a LOAD breakpoint for C, C++, and PL/I, Debug Tool has enough information about the loaded module to set breakpoints and examine variables of static and extern storage classes.
- At the triggering of a LOAD breakpoint for COBOL, C, and C++ DLL’s, Debug Tool does not have enough information about the loaded module to set breakpoints in blocks contained within the module. At the triggering of an APPEARANCE breakpoint, however, you can set such breakpoints.
- The AT LOAD command cannot be used while you replay recorded statements by using the PLAYBACK commands.

Examples
- Print a message when load module mymod is loaded. The current programming language setting is either C, C++, or COBOL.
  AT LOAD mymod LIST ("Load module mymod has been loaded");
- Establish an entry breakpoint when load module a is fetched and then resume execution. The current programming language setting is C.
  AT LOAD a {
    AT ENTRY a;
    GO;
  }

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

Related references
AT LOAD command (remote debug mode)

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

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

```
AT LOAD module_name;
```

Related references

- “AT LOAD command” on page 65
- Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

AT OCCURRENCE command

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

```
AT every_clause OCCURRENCE condition command;
```

**condition**

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

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

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

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

When a C++ user specifies AT CONDITION THROWOBJ, Debug Tool transfers control to the user at the point of the throw in C++ code.
PL/I condition constants can be used. However, FILE condition constants and CONDITION condition constants cannot be used while debugging Enterprise PL/I programs.

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

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

**command**

A valid Debug Tool command.

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

**For C and C++:**

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

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

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

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

**Usage notes**

- If the application program also has established an exception handler for the condition then that handler is entered when Debug Tool releases control, unless return is by use of 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 Debug Tool, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on Debug Tool, it is cleared at the end of a process.
- For COBOL, Debug Tool detects Language Environment conditions. If a Language Environment condition occurs during your session, the following series of events takes place:
  1. Debug Tool is started before any condition handler.
  2. If you set an OCCURRENCE breakpoint for that condition, Debug Tool processes that breakpoint and executes any commands you have specified. If you have not set an OCCURRENCE breakpoint for that condition, and:
     - If the current test-level setting is ALL, Debug Tool prompts you for commands or reads them from a commands file.
     - If the current test-level setting is ERROR, and the condition has a severity level of 2 or higher, Debug Tool gets commands by prompting you or by reading from a commands file.
     - If the current test-level setting is NONE, Debug Tool ignores the condition and returns control to the program.

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

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

- When a data exception occurs, query the current location. The current programming language setting is either C or COBOL.
  AT OCCURRENCE CEE347 QUERY LOCATION;
- When you are running in MVS without the Language Environment run time, that is under EQANMDBG, when a System 0C1 ABEND occurs, list information about the current CUs with the following command:
  AT OCCURRENCE 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).

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

Related references

"every_clause syntax" on page 42
“ON command (PL/I)” on page 178
“PLAYBACK commands” on page 184
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 Debug Tool control at the specified offset in the disassembly view.

\[\text{AT OFFSET} \quad \text{offset_spec} \quad \text{command} ;\]

\(\text{command}\)

A valid Debug Tool 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';

Gives Debug Tool control at the specified offset in the disassembly view.
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 184
- "offset_spec" on page 16

**AT PATH command**

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

```
AT PATH command;
```

*command*

A valid Debug Tool command.

**Usage notes**

- For a CICS application on Debug Tool, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on Debug Tool, it is cleared at the end of a process.
- For C, to be able to set PATH breakpoints, you must compile your program in one of the following ways:
  - With either the PATH or ALL suboption of the TEST compiler option.
  - 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.
- Debug Tool does not support the AT PATH command while you debug a disassembled program or a VS COBOL II program.

**Examples**

- Whenever a path point has been reached, display the five most recently processed breakpoints and conditions.
  ```
  AT PATH LIST LAST 5 HISTORY;
  ```
Whenever a path point has been reached, display a message and query the current location. The current programming language setting is COBOL.

```
AT PATH PERFORM
  LIST "Path point reached";
  QUERY LOCATION;
  GO;
END-PERFORM;
```

Whenever a path point has been reached, the value of %PATHCODE contains the code representing the type of path point stopped at. If the program is stopped at the entry to a block, display the %PATHCODE.

```
AT PATH LIST %PATHCODE;
```

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

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

**Related references**
- “every clause syntax” on page 42
- “%PATHCODE” on page 296
- “PLAYBACK commands” on page 184

### 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.

```
AT [integer] ;
```

**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 184
AT STATEMENT command

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

```
AT all_clause every_clause LINE statement_id_range ...
```

* Sets a breakpoint at every STATEMENT or LINE.

**command**

A valid Debug Tool command. If you are using remote debug mode, you can specify only commands that are supported in remote debug mode.

**condition**

A valid Debug Tool conditional expression.

Usage notes

- You cannot use the AT STATEMENT command (except for the AT STATEMENT * form) while you debug a disassembled program. Instead, use the AT OFFSET command.
- A STATEMENT breakpoint set for a nonactive compile unit (one that is not in the current enclave), is suspended until the compile unit becomes active. A STATEMENT breakpoint set for a compile unit that is deleted from storage is suspended until the compile unit is reloaded. A suspended breakpoint cannot be triggered until it is reactivated.
- For a CICS application on Debug Tool, this breakpoint is cleared at the end of the last process in the application. For a non-CICS application on Debug Tool, it is cleared at the end of a process.
- You can specify the first relative statement on each line in any 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, Debug Tool logs them differently according to the current programming language as follows:
  - **For C and C++**
    The first relative statement on a line is specified with "0". All of the above breakpoints are logged as AT 3.0.
  - **For COBOL or PL/I**
    The first relative statement on a line is specified with "1". All of the above breakpoints are logged as AT 3.1. For optimized COBOL programs, the first relative statement is the first executable statement. This might not be the first statement if the optimizer discarded the first statement.
- When the STORAGE run-time option is in effect, the AT STATEMENT command cannot be used to set a breakpoint in the prologue of an assembler compile unit between the first BALR 14,15 instruction and the following LR 13,x instruction.
- The AT STATEMENT command cannot be used while you replay recorded statements by using the PLAYBACK command.
- You can restrict the circumstances under which the AT STATEMENT breakpoint is raised by specifying a WHEN condition. If a WHEN condition is specified, Debug...
Tool 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 Debug Tool reaches the statement, it evaluates the condition. If the condition evaluates to true, Debug Tool stops and runs the command associated with the breakpoint.
- Debug Tool evaluates references in a WHEN condition before it runs a statement.
- When Debug Tool evaluates the condition and the condition is invalid, Debug Tool does one of the following actions:
  - If SET WARNING is set to ON, Debug Tool stops and displays a message that it could not evaluate the condition. You need to enter a command to indicate what action you want Debug Tool to take.
  - If SET WARNING is set to OFF, Debug Tool does not stop nor display a message that it could not evaluate the condition. Debug Tool 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 42
  "statement_id_range and stmt_id_spec" on page 17
  "AT OFFSET command (disassembly)" on page 70
  "PLAYBACK commands" on page 184
Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

AT STATEMENT command (remote debug mode)
Gives Debug Tool control at the specified statement or line.

\[ \begin{array}{l}
\text{AT} \quad \text{LINE} \quad \text{STATEMENT} \quad \text{statement_id} \quad ;
\end{array} \]

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 17
"AT STATEMENT command" on page 73
Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

AT TERMINATION command
Gives Debug Tool control when the application program is terminated.

\[ \begin{array}{l}
\text{AT—TERMINATION—command—;}
\end{array} \]

command
A valid Debug Tool command.

Usage notes
- The setting of the current programming language when the application program terminates might be unpredictable.
- AT TERMINATION does not allow specification of an every_clause because termination can only occur once.
- If this breakpoint is set in a parent enclave, it can be triggered and operated on with breakpoint commands while the application is in a child enclave.
- When Debug Tool gains control, normal execution of the program is complete; however, a CALL or function invocation from Debug Tool can continue to perform program code. When the AT TERMINATION breakpoint gives control to Debug Tool:
  - Fetched load modules have not been released
  - Files have not been closed
  - Language-specific termination has been started yet no action has been taken
In C, the user atexit() lists have already been called.
In PL/I, the FINISH condition was already raised.
You are allowed to enter any command with AT TERMINATION. However, normal error messages are issued for any command that cannot be completed successfully because of lack of information about your program.

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

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

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

Examples

- When the program ends, check the Debug Tool environment to see what files have not been closed.
  
  AT TERMINATION DESCRIBE ENVIRONMENT;

- When the program ends, display the message "Program has ended" and end the Debug Tool session. The current programming language setting is C.
  
  AT TERMINATION {
    LIST "Program has ended";
    QUIT;
  }

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

Related references

“PLAYBACK commands” on page 184

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.

```
BEGIN ; command ; END ;
```

command

A valid Debug Tool command.

Usage notes

- The BEGIN command is most helpful when used in AT or PROCEDURE commands.
- 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 Debug Tool commands into one command. When you enclose Debug Tool commands within a single set of braces ({}), everything within the braces is treated as a single command. You can place a block anywhere a command is allowed.

```
{ command ; command }
```

**command**

A valid Debug Tool command.

**Usage notes**

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

**Example**

Establish an entry breakpoint when load module \( a \) is fetched.

```
AT LOAD a { 
  AT ENTRY a;
  GO;
}
```

**break command (C and C++)**

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

```
break ;
```

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

**Usage notes**

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

**Examples**

- The following example shows a `break` command in the action part of a `for` command. If the i-th element of the array `string` is equal to `'\0'`, the `break` command causes the `for` command to end.

```c
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.

```c
char key;

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

**CALL command**

The `CALL` command calls either a procedure, entry name, or program name, or it requests that a utility function be run. The C and C++ equivalent for `CALL` is a function reference. PL/I subroutines or functions cannot be called dynamically during a Debug Tool session. The `CALL` keyword cannot be abbreviated.

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

```c
extern "C"
```

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

The following table summarizes the forms of the `CALL` command.

<table>
<thead>
<tr>
<th>Form of <code>CALL</code> command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL %CEBR command on page 79</td>
<td>Starts the CICS Temporary Storage Browser Program.</td>
</tr>
<tr>
<td>CALL %CECI command on page 79</td>
<td>Starts the CICS Command Level Interpreter Program.</td>
</tr>
<tr>
<td>CALL %DUMP command on page 80</td>
<td>Calls a dump service to obtain a formatted dump.</td>
</tr>
<tr>
<td>CALL %FA command on page 84</td>
<td>Calls IBM Fault Analyzer to provide a formatted dump of the current machine state.</td>
</tr>
</tbody>
</table>
CALL %HOGAN command
Starts Computer Sciences Corporation’s KORE-HOGAN application.

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

CALL entry_name command (COBOL)
Calls an entry name in the application program (COBOL).

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

CALL %CEBR command
Starts the CICS Temporary Storage Browser Program.

```
CALL %CEBR;
```

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

Refer to the following 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.

```
CALL %CECI;
```

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

Refer to the following 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.

 CALL %DUMP (options_string, title)

**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 Debug Tool issues an EXEC CICS DUMP TRANSACTION.

**TRACEBACK**

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

TRACEBACK can be abbreviated as TRACE.

**NOTRAZEBACK**

Suppresses traceback.

NOTRAZEBACK can be abbreviated as NOTRAZ.

**FILES**

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

FILES can be abbreviated as FILE.

**NOFILES**

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

VARIABLES
Requests a symbolic dump of all variables, arguments, and registers.

Variables include arrays and structures. Register values are those saved in
the stack frame at the time of call. There is no way to print a subset of this
information.

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

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

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

VARIABLES can be abbreviated as VAR.

NOVARIABLES
Suppresses dump of variables, arguments, and registers.

NOVARIABLES can be abbreviated as NOVAR.

BLOCKS
Produces a separate hexadecimal dump of control blocks.

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

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

BLOCKS can be abbreviated as BLOCK.

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

NOBLOCKS
Suppresses the hexadecimal dump of control blocks.

NOBLOCKS can be abbreviated as NOBLOCK.

STORAGE
Dumps the storage used by the program.

The storage is displayed in hexadecimal and character format. Global
storage and storage associated with each routine on the call chain is
printed. Storage is dumped for Debug Tool. The dump proceeds up the call
chain for the number of routines specified by the STACKFRAME option.
Storage for all file buffers is also dumped if the FILES option is specified.
When the Dynamic Debug facility is activated, some of the original application instructions are not displayed because they are replaced by '0A91'x instructions.

STORAGE can be abbreviated as STOR.

NOSTORAGE
Suppresses storage dumps.

NOSTORAGE can be abbreviated as NOSTOR.

STACKFRAME(n|ALL)
Specifies the number of stack frames dumped from the call chain.

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

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

STACKFRAME can be abbreviated to SF.

PAGESIZE(n)
Specifies the number of lines on each page of the dump.

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

PAGESIZE can be abbreviated to PAGE.

FNAME(s)
Specifies the ddname of the file where the dump report is written.

The default ddname CEEEDUMP 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 Debug Tool routine that called the dump service and the contents of the registers at the point of the call. For the currently supported programming languages, ENTRY is extraneous and will be ignored.

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

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

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

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

Usage notes
- If incorrect options are used, a default dump is written.
- The service used to format the dump is determined by the following conditions:
  - Language Environment is active
    Language Environment dump service: Debug Tool does not analyze any of the CALL %DUMP options, but just passes them to the Language Environment dump service. Some of these options might not be appropriate, because the call is being made from Debug Tool rather than from your program.

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

  Language Environment not active and you are not running under CICS
  The MVS SNAP dump service
- When you use CALL %DUMP, one of the following ddnames must be allocated for you to receive a formatted dump:
  - CEEDUMP (default)
  - SYSPRINT.

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

CICS: You do not need this allocation when you are running without Language Environment under CICS. Under those conditions, EXEC CICS DUMP TRANSACTION is issued, and a transaction dump with a code of $DT$ is written to the CICS dump data set.
- COBOL does not do anything if the FILES option is specified; the BLOCKS option gives the file information instead.

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

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

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

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

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

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

Related references
- "PLAYBACK commands" on page 184
- 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.
CALL %FA command

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.

CALL %FM command

Starts IBM File Manager for z/OS.

CALL %FM userID [BACKGROUND];

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).

CALL %HOGAN ;

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 AEIO exception.

CALL %VER command

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

CALL %VER ;
You can use this command in remote debug mode.

**Example**

You have Debug Tool for z/OS, Version 10.1, installed on your system. Enter the CALL %VER command to display the following information in the Log window:

IBM Debug Tool Version 10 Release 1 Mod 0
11/06/2009 4:11:41 PM Level: V10R1 PKnnnnn
5655-V50: Copyright IBM Corp. 1992, 2009

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 B, “Debug Tool commands supported in remote debug mode,” on page 393

**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).

\[\text{CALL identifier literal USING identifier_clause;}\]

**identifier_clause:**

\[\text{REFERENCE ADDRESS OF identifier}\]
\[\text{BY CONTENT identifier}\]
\[\text{ADDRESS OF identifier}\]
\[\text{LENGTH OF literal}\]

<table>
<thead>
<tr>
<th>identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>A valid Debug Tool COBOL identifier.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>literal</th>
</tr>
</thead>
<tbody>
<tr>
<td>A valid COBOL literal.</td>
</tr>
</tbody>
</table>

**Usage notes**

- If you have a COBOL entry point name that is the same as a Debug Tool procedure name, the procedure name takes precedence when using the CALL command. If you want the entry name to take precedence over the Debug Tool procedure name, you must qualify the entry name when using the CALL command.
- You can use the CALL `entry_name` command to change program flow dynamically. You can pass parameters to the called module.
- The CALL follows the same rules as calls within the COBOL language.
The COBOL ON OVERFLOW and ON EXCEPTION phrases are not supported, so END-CALL is not supported.

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

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

- 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 184
- Enterprise COBOL for z/OS Language Reference

CALL procedure command

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

CALL procedure_name;

procedure_name

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

Usage notes

- Because the Debug Tool procedure names are always uppercase, the procedure name is converted to uppercase even for programming languages that have mixed-case symbols.

- The CALL keyword is required even for programming languages that do not use CALL for subroutine invocations.

- The CALL command is restricted to calling procedures in the currently executing enclave.

Example

Create and call the procedure named proc1.

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

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

CHKSTGV ;

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 B, “Debug Tool commands supported in remote debug mode,” on page 393

Related tasks

“Detecting CICS storage violations early” in Debug Tool User’s Guide

CLEAR command

The CLEAR command removes the actions of previously entered Debug Tool commands. Some breakpoints are removed automatically when Debug Tool determines that they are no longer meaningful. For example, if you set a breakpoint in a fetched or loaded compile unit, the breakpoint is discarded when the compile unit is released.
AT  Removes all breakpoints, including GLOBAL breakpoints, set by previously entered AT commands, except for AT TERMINATION breakpoints.

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

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

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

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

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

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

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

LOG
Erases the log file and clears out the data being retained for scrolling. In line mode, CLEAR LOG clears only the log file.

If the log file is directed to a SYSOUT type file, CLEAR LOG will not clear the log contents in the file.

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.

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

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

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

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

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

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

### Usage notes

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

### Examples

- Remove the LABEL breakpoint set in the program at label create.
  
  \[ \text{CLEAR AT LABEL create;} \]
- Remove previously defined variables \( x, y, \) and \( z \).
  
  \[ \text{CLEAR DECLARE \((x, y, z)\);} \]
- Remove the effect of the ninth command defined for MONITOR.
  
  \[ \text{CLEAR MONITOR 9;} \]
- 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.
  
  \[ \text{CLEAR VARIABLES struct tagone;} \]
- Establish some breakpoints with the AT command and then remove them with the CLEAR command (checking the results with the LIST command).
  
  \[
  \begin{align*}
  \text{AT 50;} \\
  \text{AT 56;} \\
  \text{AT 55 LIST \((r, c)\);} \\
  \text{LIST AT;} \\
  \text{CLEAR AT 50;} \\
  \text{LIST AT;} \\
  \text{CLEAR AT;} \\
  \text{LIST AT;} \\
  \end{align*}
  \]
- If you want to clear an AT ENTRY * breakpoint, specify:
CLEAR AT ENTRY *;  
or  
CLEAR AT GLOBAL ENTRY;  

- If you want to remove the DATE breakpoint for block MYBLOCK, specify:  
  CLEAR AT DATE MYBLOCK;  

- If you want to remove a generic DATE breakpoint, specify:  
  CLEAR AT DATE *;  

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

**Related references**  
[
“CLEAR prefix (full-screen mode)”  
“AT command” on page 39  
“LIST command” on page 141  
“PLAYBACK commands” on page 184  
Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393  
]

**CLEAR prefix (full-screen mode)**  
Clears a breakpoint when you enter this command through the Source window prefix area or clears a selected member of the current set of MONITOR commands when you enter this command through the Monitor window prefix area.  

```
  CLEAR [integer];
```

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

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

**Examples**  
- In the Source window, clear a breakpoint at the third statement or verb in the line (typed in the prefix area of the line where the statement is found).  
  CLEAR 3  
  No space is needed as a delimiter between the keyword and the integer; hence, CLEAR 3 is equivalent to CLEAR3.  
- In the Monitor window, type CL in the prefix area to on the line that displays the entry you want to remove, then press Enter.
COMMENT command

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

```
COMMENT [commentary];
```

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

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

**Examples**
- Comment that `varblxx` seems to have the wrong value.
  ```
  COMMENT At this point varblxx seems to have the wrong value;
  ```
- Combine a commentary with valid Debug Tool commands.
  ```
  COMMENT Entering subroutine testrun; LIST (x); GO;
  ```

COMPUTE command (COBOL)

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

```
COMPUTE reference = expression;
```

**reference**
A valid Debug Tool COBOL numeric reference.

**expression**
A valid Debug Tool COBOL numeric expression.

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

**Examples**

- Assign to variable x the value of \( a + 6 \).
  ```cobol
  COMPUTE x = a + 6;
  ```

- Assign to the variable mycode the value of the Debug Tool variable \( \%\text{PATHCODE} + 1 \).
  ```cobol
  COMPUTE mycode = \%\text{PATHCODE} + 1;
  ```

- Assign to variable xx the result of the expression \( (a + e(1)) / c * 2 \).
  ```cobol
  COMPUTE xx = (a + e(1)) / c * 2;
  ```
  You can also use table elements in such assignments as shown in the following example.
  ```cobol
  COMPUTE itm-2(1,2) = (a + 10) / e(2);
  ```

- To assign a value to a session variable named TSO or SYSTEM, append the "=" to the reference as shown in the following example.
  ```cobol
  COMPUTE tso= 5;
  ```

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

**Related references**

- “MOVE command (COBOL)” on page 171
- “PLAYBACK commands” on page 184
- “SET WARNING command (C, C++, COBOL, and PL/I)” on page 258

---

**CURSOR command (full-screen mode)**

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

```cobol
CURSOR —;
```

**Usage notes**

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

Example

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

CURSOR;

Declarations (assembler, disassembly, and non-Language Environment COBOL)

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

_identifier—DS_F

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

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

Usage note

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

**Declarations (C and C++)**

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

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

---

**scalar_def:**

- char
- signed
- unsigned
- double
- long
- float
- int
- signed
- unsigned
- long
- short
- long
- signed
- unsigned
- int
- long
- short
- char
- signed
- unsigned
- long
- int
- unsigned
- long
- int
- void
- char

**declarator:**

- identifier
- (identifier)
- [integer]
enum_def:

```
enum
    identifier
        constant_expr
```

struct_def:

```
_Packed
struct
    identifier
```

union_def:

```
_Packed
union
    identifier
```

*  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:
• 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:

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Constant</th>
<th>Integer Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>run</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>create</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>delete</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>suspend</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

  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
Debug Tool User’s Guide

Declarations (COBOL)

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

\[ \text{PIC} \quad \text{IS} \quad \text{picture} \quad \text{usage_attribute} \]

usage_attribute:

\[ \text{USAGE} \quad \text{IS} \quad \text{POINTER} \]

\[ \begin{align*}
\text{USAGE} & \quad \text{IS} \quad \text{BINARY} \\
& \quad \text{COMP} \\
& \quad \text{COMPUTATIONAL} \\
& \quad \text{COMP-1} \\
& \quad \text{COMPUTATIONAL-1} \\
& \quad \text{COMP-2} \\
& \quad \text{COMPUTATIONAL-2}
\end{align*} \]

level

1 or 77.

identifier

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

picture

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

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

Usage notes

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

Examples

- Define a variable named floatmp to hold a floating-point number.
  \[
  01 \text{ floatmp USAGE COMP-1;}
  \]
- Define an integer variable name temp.
  \[
  77 \text{ temp PIC S9(9) USAGE COMP;}
  \]

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

Related tasks

- Debug Tool 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 Debug Tool session. Variables declared this way can be used in other Debug Tool commands as if they were declared to the compiler. They are removed with the CLEAR command or when your Debug Tool session ends. The keywords cannot be abbreviated.

```
DCL DECLARE ...
```

### major_structure:

```
level name ...
```

### scalar:

```
name ...
```

**level**

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

**name**

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

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

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

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

**attribute**

A PL/I data or storage attribute.

Acceptable PL/I data attributes include:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>BINARY</th>
<th>CPLX</th>
<th>FIXED</th>
<th>LABEL</th>
<th>PTR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BIT</td>
<td>DECIMAL</td>
<td>FLOAT</td>
<td>OFFSET</td>
<td>REAL</td>
</tr>
<tr>
<td></td>
<td>CHARACTERS</td>
<td>EVENT</td>
<td>GRAPHIC</td>
<td>POINTER</td>
<td>VARYING</td>
</tr>
</tbody>
</table>
Acceptable PL/I storage attributes include:

```
BASED    ALIGNED    UNALIGNED
```

Pointers cannot be specified with the BASED option.

Only simple factoring of attributes is allowed. DECLAREs such as the following are not allowed:

```
DCL (a(2), b) PTR;
DCL (x REAL, y CPLX) FIXED BIN(31);
```

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

**Usage notes**

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

**Examples**

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

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

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

In COBOL, this can be any user-defined name appearing in the DATA DIVISION. Names can be subscripted or 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 Debug Tool non-Language Environment COBOL reference. This form must be used for non-Language Environment 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 non-Language Environment 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, Debug Tool 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, assembler, and disassembly do not provide any information for DESCRIBE ENVIRONMENT.

LOADMOS

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

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

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

load_spec

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

Usage notes

- You can use the DESCRIBE CUS, DESCRIBE CHANNEL, and DESCRIBE LOADMOS 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.

• Non-Language Environment COBOL PIC attributes might not match the original PIC specification in the following situations:
  – A COMP-3 variable will always have an odd number of digits in its PIC value.
  – All non-numeric strings will have a PIC value of X’s.

• If the DATA option of the PLAYBACK ENABLE command is in effect for the current compile unit, the DESCRIBE ATTRIBUTES and DESCRIBE CURSOR commands can be used while you replay recorded statements by using the PLAYBACK commands.

• The DESCRIBE ENVIRONMENT command cannot be used while you replay recorded statements by using the PLAYBACK commands.

• The DESCRIBE LOADMODS command does not display information about load modules or compile units provided by operating system, subsystem, or run-time software (for example: MVS, CICS, DB2, IMS, and Language Environment) because Debug Tool 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)

• 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, 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, Debug Tool 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>
<thead>
<tr>
<th>If the container name is...</th>
<th>If the programming language is PL/I, COBOL, assembler or disassembly, type in...</th>
<th>If the programming language is C or C++, type in...</th>
</tr>
</thead>
<tbody>
<tr>
<td>chname</td>
<td>DESCRIBE CHANNEL 'chname'</td>
<td>DESCRIBE CHANNEL chname</td>
</tr>
<tr>
<td>conNAME</td>
<td>DESCRIBE CHANNEL 'conNAME'</td>
<td>DESCRIBE CHANNEL conNAME</td>
</tr>
</tbody>
</table>

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 16
“cu spec” on page 14
“LIST CONTAINER command” on page 146
Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

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 Debug Tool 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, Debug Tool 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, 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++, Debug Tool 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;

• Debug Tool 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. If you do not want Debug Tool 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 Debug Tool 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 Debug Tool 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 Debug Tool is started every time a program matches one of these profiles. If you do not want Debug Tool 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 [Debug Tool User’s Guide](#)

**Related references**

- “ENABLE command” on page 112
- “DISABLE prefix (full-screen mode)”
- “LIST DTCN or CADP command” on page 148
- Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

**DISABLE prefix (full-screen mode)**

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

```
DISABLE integer;
```

**integer**

Selects a relative statement (for C and C++ or PL/I) or a relative verb (for COBOL) within the line. The default value is 1.

**Example**

Disable the breakpoint at the third statement or verb in the line by entering the following command in the prefix area of the line where the statement is found.

```
DIS 3
```

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

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

**Related tasks**

[Debug Tool User’s Guide](#)

**DO command (assembler, disassembly, and non-Language Environment COBOL)**

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

```
DO command;
```

**command**

A valid Debug Tool command.
do/while command (C and C++)

The do/while command performs a command before evaluating the test expression. Due to this order of execution, the command is performed at least once. The do and while keywords must be lowercase and cannot be abbreviated.

\[
\text{do-command-while-\{expression\}-};
\]

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

- **expression**
  A valid Debug Tool C and C++ expression.

The body of the loop is performed before the while clause (the controlling part) is evaluated. Further execution of the do/while command depends on the value of the while clause. If the while clause does not evaluate to false, the command is performed again. Otherwise, execution of the command ends.

A break command can cause the execution of a do/while command to end, even when the while clause does not evaluate to false.

**Usage note**

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

**Example**

The following command prompts you to enter a 1. If you enter a 1, the command ends execution. Otherwise, the command displays another prompt.

```c
int reply1;

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

DO command (PL/I)

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

**Simple**

\[
\text{DO-command-END};
\]

- **command**
  A valid Debug Tool 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 Debug Tool PL/I Boolean expression.

UNTIL

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

command

A valid Debug Tool command.

Iterative

reference

A valid Debug Tool PL/I reference.

expression

A valid Debug Tool PL/I expression.
BY Specifies that expression is evaluated at entry to the DO specification and saved. This saved value specifies the increment to be added to the control variable after each execution of the DO group.

If BY expression is omitted from a DO specification and if TO expression is specified, expression defaults to the value of 1.

If BY 0 is specified, the execution of the DO group continues indefinitely unless it is halted by a WHILE or UNTIL option, or control is transferred to a point outside the DO group.

The BY option allows you to vary the control variable in fixed positive or negative increments.

TO Specifies that expression is evaluated at entry of the DO specification and saved. This saved value specifies the terminating value of the control variable.

If TO expression is omitted from a DO specification and if BY expression is specified, repetitive execution continues until it is terminated by the WHILE or UNTIL option, or until some statement transfers control to a point outside the DO group.

The TO option allows you to vary the control variable in fixed positive or negative increments.

REPEAT Specifies that expression is evaluated and assigned to the control variable after each execution of the DO group. Repetitive execution continues until it is terminated by the WHILE or UNTIL option, or until some statement transfers control to a point outside the DO group.

The REPEAT option allows you to vary the control variable nonlinearly. This option can also be used for nonarithmetic control variables, such as pointers.

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

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

command

A valid Debug Tool command.

Usage note

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

Examples

- At statement 25, initialize variable a and display the values of variables x, y, and z.
  
  AT 25 DO; %BLOCK:>a = 0; LIST (x, y, z); END;

- Execute the DO group until ctr is greater than 4 or less than 0.
  
  DO UNTIL (ctr > 4) WHILE (ctr >= 0); END;
• Execute the DO group with i having the values 1, 2, 4, 8, 16, 32, 64, 128, and 256.
  DO i = 1 REPEAT 2*i UNTIL (i = 256); END;
• Repeat execution of the DO group with j having values 1 through 20, but only if k has the value 1.
  DO j = 1 TO 20 BY 1 WHILE (k = 1); END;

**ENABLE command**

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

```
AT_command
  ENABLE
  CADP
    PROGRAM prog_id
    CU cu_id
  DTCN
    LOADMOD loadmod_id
    CU cu_id
```

**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_, Debug Tool 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, Debug Tool 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, 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++, Debug Tool 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 Debug Tool 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 Debug Tool 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 Debug Tool User’s Guide

Related references
“DISABLE prefix (full-screen mode)” on page 108
“ENABLE prefix (full-screen mode)”
“LIST DTCN or CADP command” on page 148
Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

ENABLE prefix (full-screen mode)
Enables a disabled statement breakpoint or a disabled offset breakpoint when you issue this command through the Source window prefix area.

```
ENABLE [integer] ;
```

integer
Selects a relative statement (for C and C++ or PL/I) or a relative verb (for COBOL) within the line. The default value is 1. For optimized COBOL programs, the default value is the first executable statement which was not discarded by the optimizer.

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

ENABLE 3

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

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

```
EVALUATE constant expression reference TRUE FALSE
WHEN any_clause command
END-EVALUATE;
WHEN OTHER command
```

**any_clause:**

```
ANY condition TRUE FALSE
NOT constant reference THROUGH constant THRU reference
```

**constant**
A valid Debug Tool COBOL constant.

**expression**
A valid Debug Tool COBOL arithmetic expression.

**reference**
A valid Debug Tool COBOL reference.

**condition**
A simple relation condition.

**command**
A valid Debug Tool command.

**Usage notes**

- Only a single subject is supported.
- Consecutive WHENs without associated commands are not supported.
- THROUGH/THRU ranges can be specified as constants or references.
- See [Enterprise COBOL for z/OS Language Reference](#) for an explanation of the following COBOL keywords:

  ANY
  FALSE
  NOT
  OTHER
• Debug Tool implements the EVALUATE command as a series of IF commands.
• If the DATA option of the PLAYBACK ENABLE command is in effect for the current compile unit, the EVALUATE command can be used while you replay recorded statements by using the PLAYBACK commands.
• For optimized COBOL programs, the value of reference cannot refer to any variables discarded by the optimizer.
• If a COBOL variable is defined as National and it is an operand in a relation condition with an alphabetic, alphanumeric operand, or National numeric, the operand that is not National is converted to Unicode before that comparison is done, except for Group items. See “Allowable comparisons for the IF command (COBOL)” in 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 Debug Tool C and C++ expression. Assignment is affected by including one of the C and C++ assignment operators in the expression. No use is made of the value resulting from a stand-alone expression.

Usage notes

- Function invocations in expressions are restricted to functions contained in the currently executing enclave.
- The Expression command cannot be used while you replay recorded statements by using the PLAYBACK commands.

Examples

- Initialize the variables $x$, $y$, $z$. You can use functions to provide values for variables.
  
  ```
  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, assembler, disassembly, or PL/I, use quotation marks (""") or apostrophes (').

Table 5. Examples of how to specify quotation marks ("") and apostrophes ('') for strings in a FIND command.

<table>
<thead>
<tr>
<th>C</th>
<th>C++</th>
<th>COBOL or non-Language Environment COBOL</th>
<th>Assembler or disassembly</th>
<th>PL/I</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ABC&quot;</td>
<td>&quot;IntLink:::*&quot;</td>
<td>&quot;A5&quot; or 'A5'</td>
<td>'ABC' or &quot;ABC&quot; or C'ABC'</td>
<td>'ABC' or &quot;ABC&quot;</td>
</tr>
</tbody>
</table>

- 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, 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 Debug Tool 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, Debug Tool displays an error message.
   – If the previous FIND command that you entered specified or implied the FIRST or NEXT parameter, Debug Tool uses the NEXT parameter.
   – If the previous FIND command that you entered specified the LAST or PREV parameter, Debug Tool uses the PREV parameter.
   – If the previous FIND command that you entered specified a leftcolumn parameter, Debug Tool uses that leftcolumn parameter.
   – If the previous FIND command that you entered specified a rightcolumn parameter, Debug Tool uses that rightcolumn parameter.
   – If a repeat FIND immediately follows an unsuccessful FIND or repeat FIND, Debug Tool continues searching, wrapping from the last line to the first line. If the original direction of the FIND was backward to the beginning of the object, Debug Tool wraps from the first line to the last line.
   – If the cursor is not in a window, Debug Tool uses the same window that was used for the previous FIND command.

• In full-screen mode, Debug Tool chooses the window it searches through in the following ways:
   – If you specify a string and you do not place the cursor in a window nor specify an object on the command, Debug Tool searches the object in the window specified by the SET DEFAULT WINDOW command or the Default window entry in your Profile Settings panel.
   – If you place the cursor in a window and do not specify a different window on the command, Debug Tool searches the object in the window where you placed the cursor.

• If you specify a string without a direction keyword, forward is the default direction.
• FIND can be made immediately effective in full-screen mode with the IMMEDIATE command.
• If the current programming language setting is C or C++, the search is case-sensitive. Otherwise, the search is not case-sensitive.
• In full-screen mode, searches 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, Debug Tool wraps to the end of the object and continues from the end of the last line in the object.
  – If Debug Tool finds the string, the window for the object being searched is scrolled until the string is visible. If the string is DBCS, it is displayed without alteration. If the string is not DBCS, the string is highlighted as specified by the SET COLOR command and the cursor is placed at the beginning of the string.
  – If Debug Tool does not find the string, the screen does not change and the cursor is not moved. If you specified NEXT or PREV or the command defaults to NEXT or PREV and Debug Tool searched only part of the object, then Debug Tool displays the message 'Bottom of data reached' or 'Top of data reached', as appropriate. If Debug Tool searched through the entire object, then it displays the message 'Search target not found'.

• In 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, Debug Tool begins searching at the first character of the first line of the source or, if a previous FIND command was done in the same compile unit, at the location after the last string that was successfully found by a FIND command.
  – If you specify PREV or the command defaults to PREV, Debug Tool begins searching at the last character of the last line of the source, or if a previous FIND command was done in the same compile unit, at the location before the last string that was successfully found by a FIND command.
  – If you specify NEXT or PREV or the command defaults to NEXT or PREV and Debug Tool searched only part of the source and did not find the string, then Debug Tool displays the message 'Bottom of data is reached' or 'Top of data is reached', as appropriate. If Debug Tool searched through the entire source without finding the string, then it displays the message 'Search target not found'.
  – If Debug Tool finds the string, the line that contains the string is displayed and marked with a vertical bar character (|) beneath the string.

• The 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 Debug Tool 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, Debug Tool 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 Debug Tool remembers the string you last searched for. Again, the Source window is scrolled forward, var1 is highlighted, and the cursor points to the variable.
- If you want to find a question mark (?) in the Source window and you are debugging a PL/I program, enter the following command: FIND '?';
- If you want to find the string User's in the Source window and you are debugging a PL/I program, enter the following command:
  FIND User''s;
- If you want to find the string User's in the Source window and you are debugging a C program, enter the following command:
  FIND User's;
- If you want to find the string User's Guide in the Source window and you are debugging a PL/I program, enter the following command:
  FIND 'User''s Guide';
- If you want to find the string User's Guide in the Source window and you are debugging a C program, enter the following command:
  FIND "User's Guide";
- If you entered the command FIND xyz LAST; or FIND xyz PREV; and the cursor is on the found string ("xyz"), then press the PF key assigned to the FIND command to repeat the search. Debug Tool runs the command FIND xyz PREV;.
• If you entered the command `FIND xyz;`, Debug Tool searches in the forward direction. To find the string "xyz" in the backward direction, enter the command `FIND * PREV;`.

• 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.

```
  FINDBP
      ]FIRST   [ENABLED |
      ]LAST    [DISABLED |
      ]NEXT    |
      ]PREV    |
```

**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, Debug Tool uses the `NEXT` parameter.
  – If the previous `FINDBP` command that you entered specified or implied the `LAST` or `PREV` parameter, Debug Tool uses the `PREV` parameter.
– If a repeat FINDBP immediately follows an unsuccessful FINDBP or repeat FINDBP, Debug Tool 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, Debug Tool 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, Debug Tool 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 241.

• 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, Debug Tool wraps to the end of the source object and continues with the last source line.
  – If Debug Tool finds the breakpoint, Debug Tool scrolls the Source window so that you can see the breakpoint. Debug Tool places the cursor at the beginning of the prefix area for the source line that contains the breakpoint.
  – If Debug Tool 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 Debug Tool searched only part of the source object, then Debug Tool displays the message “Bottom of data reached” or “Top of data reached”, as appropriate. If Debug Tool 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.

• Debug Tool 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

- “AT LINE command” on page 65
- “AT OFFSET command (disassembly)” on page 70
- “AT STATEMENT command” on page 73
- “LIST AT command” on page 142, with the LINE, OFFSET, or STATEMENT options
- “POSITION command” on page 188
- “SCROLL command (full-screen mode)” on page 201, with the TO option
- “SET PFKEY command” on page 241

for command (C and C++)

The for command provides iterative looping similar to the C and C++ for statement. It enables you to do the following:

- Evaluate an expression before the first iteration of the command ("initialization").
- Specify an expression to determine whether the command should be performed again ("controlling part").
- Evaluate an expression after each iteration of the command.
- Perform the command, or block, if the controlling part does not evaluate to false.

The for keyword must be lowercase and cannot be abbreviated.

```plaintext
for (expression; expression; expression) command
```

expression
A valid Debug Tool C and C++ expression.

command
A valid Debug Tool command.

Debug Tool evaluates the first expression only before the command is performed for the first time. You can use this expression to initialize a variable. If you do not want to evaluate an expression before the first iteration of the command, you can omit this expression.

Debug Tool evaluates the second expression before each execution of the command. If this expression evaluates to false, the command does not run and control moves to the command following the for command. Otherwise, the command is performed. If you omit the second expression, it is as if the expression has been replaced by a nonzero constant and the for command is not terminated by failure of this expression.
Debug Tool evaluates the third expression after each execution of the command. You might use this expression to increase, decrease, or reinitialize a variable. If you do not want to evaluate an expression after each iteration of the command, you can omit this expression.

A break command can cause the execution of a for command to end, even when the second expression does not evaluate to false. If you omit the second expression, you must use a break command to stop the execution of the for command.

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

Examples
- The following for command lists the value of count 20 times. The for command initially sets the value of count to 1. After each execution of the command, count is incremented.
  ```c
  for (count = 1; count <= 20; count++)
      LIST TITLED count;
  ```
  Alternatively, the preceding example can be written with the following sequence of commands to accomplish the same task.
  ```c
  count = 1;
  while (count <= 20) {
      printf("count = %d\n", count);
      count++;
  }
  ```
- The following for command does not contain an initialization expression.
  ```c
  for (; index > 10; --index) {
      varlist[index] = var1 + var2;
      printf("varlist[%d] = %d\n", index, varlist[index]);
  }
  ```

FREE command
The FREE command frees a file that is currently allocated.

```
FREE FILE ddname;
```

ddname
Name of the file to free.

GO command
The GO command causes Debug Tool to start or resume running your program.

```
GO BYPASS;
```

BYPASS
Bypasses the user or system action for the condition that caused the breakpoint. It is valid only when Debug Tool is entered for an:
- AT CALL breakpoint
- HLL or Language Environment condition
Condition raised by an MVS or CICS ABEND when running without the Language Environment run time

Usage notes

- If GO is specified in a command list (for example, as the subject of an IF command or WHEN clause), all subsequent commands in the list are ignored.
- If GO is specified within the body of a loop, it causes the execution of the loop to end.
- To suppress the logging of GO commands, use the SET ECHO command.
- GO with no operand specified does not actually resume the program if there are additional AT-conditions that have not yet been processed.
- The GO command cannot be used while you replay recorded statements by using the PLAYBACK commands by using the PLAYBACK command.
- You can use the GO command in remote debug mode only by entering it in the Action field, which is in the Optional Parameters section of the Add a Breakpoint task.

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 39

GOTO command

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

```
GOTO statement_id;
```

Usage notes

- You cannot use the GOTO command while you debug a disassembled program.
- If GOTO is specified in a command list (for example, as the subject of an IF command or WHEN clause), all subsequent commands in the list are ignored.
- Statement GOTO’s are not restricted if the program is compiled with minimum optimization.
- The GOTO command cannot be used while you replay recorded statements by using the PLAYBACK command.
For C, C++, and PL/I, statements can be removed by the compiler during optimization, specify a reference or statement with the GOTO command that can be reached during program execution. You can issue the LIST STATEMENT NUMBERS command to determine the reachable statements.

PL/I allows GOTO in a command list on a call to 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 4.1, 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 4.1, 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 4.1, and specify the EJPD and NOHOOK suboption of the TEST compiler option. Specifying the EJPD suboption might cause some loss of optimization.

Examples

- Resume execution at statement 23, where statement 23 is in a currently active block.
  GOTO 23;
If there’s no breakpoint at statement 23, Debug Tool will run from statement 23 until a breakpoint is hit.

- Resume execution at statement 45, where statement 45 is in a currently active block.
  AT 45
  GOTO 45

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

Related tasks
Debug Tool User’s Guide

Related references
“statement_id” on page 17
GOTO LABEL command

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

```
GOTO statement_label
```

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

**Usage notes**
- Use the syntax of `statement_label` enclosed in apostrophes (') only for non-Language Environment COBOL programs. It is not supported in any other programming language.
- In PL/I, out-of-block GOTOs are allowed. However, qualification might be needed.
- The LABEL keyword is optional when either the target `statement_label` is nonnumeric or if it is qualified (whether the actual label was nonnumeric or not).
- A COBOL `statement_label` can have either of the following forms:
  - `name`
    This form can be used in COBOL for reference to a section name or for a COBOL paragraph name that is not within a section or is in only one section of the block.
  - `name1 OF name2` or `name1 IN name2`
    This form must be used for any reference to a COBOL paragraph (name1) that is within a section (name2), if the same name also exists in other sections in the same block. You can specify either `OF` or `IN`, but Debug Tool always uses `OF` for output.

Either form can be prefixed with the usual block, compile unit, and load module qualifiers.
- For C, to be able to use the GOTO LABEL command, you must compile your program in one of the following ways:
  - With either the PATH or ALL suboption and the SYM suboption of the TEST compiler option.
  - 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 4.1

- 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

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 Debug Tool 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.

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**

- **Debug Tool User’s Guide**

**Related references**

- “statement_label” on page 18

---

### %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.

```
%IF condition THEN command [ELSE command];
```

**condition**

A simple relation condition valid for all supported programming languages.
A valid Debug Tool command or a BEGIN-END group containing one or more valid Debug Tool commands. The Debug Tool 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 76
- “IF command (assembler, disassembly, and non-Language Environment COBOL)” on page 129
- “if command (C and C++)” on page 130
- “IF command (COBOL)” on page 131
- “IF command (PL/I)” on page 135

### IF command (assembler, disassembly, and non-Language Environment COBOL)

The IF command lets you conditionally perform a command. You can optionally specify an ELSE clause on the IF command. If the test expression evaluates to false and the ELSE clause exists, the command associated with the ELSE clause is performed. The IF and ELSE keywords cannot be abbreviated.

```
IF condition THEN command ELSE command ;
```

**condition**
An assembler conditional expression.

`'condition'`
A non-Language Environment COBOL conditional expression enclosed in apostrophes (').

**command**
A valid Debug Tool command or a DO group containing one or more valid Debug Tool 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 non-Language Environment 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.

```
   if(expression)command  [else command]:
```

expression

A valid Debug Tool C and C++ expression.

command

A valid Debug Tool command.

When if commands are nested and else clauses are present, a given else is associated with the closest preceding if clause within the same block.

Usage notes

- An else clause should always be included if the if clause causes Debug Tool to get more input (for example, an if containing USE or other commands that cause Debug Tool to be restarted because an AT-condition occurs).
- The if command cannot be used while you replay recorded statements by using the PLAYBACK commands by using the PLAYBACK command.

Examples

- The following example causes grade to receive the value "A" if the value of score is greater than or equal to 90.

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

- The following example shows a nested if command.

  ```
  if (paygrade == 7) {
    if (level >= 0 && level <= 8)
      salary *= 1.05;
    else
      salary *= 1.04;
  }
  else
    salary *= 1.06;
  ```
IF command (COBOL)

The IF command lets you conditionally perform a command. You can optionally specify an ELSE clause on the IF command. If the test expression evaluates to false and an ELSE clause exists, the command associated with the ELSE clause is performed. The keywords cannot be abbreviated.

```
IF condition
  THEN
    command
  ELSE
    command
END-IF
```

**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 Debug Tool command.

When IF commands are nested and ELSE clauses are present, a given ELSE or END-IF is associated with the closest preceding IF clause within the same block.

Unlike COBOL, Debug Tool requires terminating punctuation (;) after commands. The END-IF keyword is required.

**Usage notes**

- An ELSE clause should always be included if the IF clause causes Debug Tool to get more input (for example, an IF containing USE or other commands that cause Debug Tool to be restarted because an AT-condition occurs).
- The COBOL NEXT SENTENCE phrase is not supported.
- Comparison combinations with windowed date fields are not supported.
- Comparisons between expanded date fields with different DATE FORMAT clauses are not supported.
- If the DATA option of the PLAYBACK ENABLE command is in effect, the IF command can be used while you replay recorded statements by using the PLAYBACK commands.
- For optimized COBOL programs, the IF clause cannot reference any variables discarded by the optimizer.
- If a COBOL variable is defined as National and it is an operand in a relation condition with an alphabetic, alphanumeric operand, or National numeric, the operand that is not National is converted to Unicode before that comparison is
done, except for Group items. See Enterprise COBOL for z/OS Language Reference for more information about using COBOL variables in conditional expressions.

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

**Related references**

“Allowable comparisons for the IF command (COBOL)”

---

**Allowable comparisons for the IF command (COBOL)**

The following table shows the allowable comparisons for the Debug Tool IF command. A description of the codes follows the table.
<table>
<thead>
<tr>
<th>OPERAND</th>
<th>Group (GR)</th>
<th>Alphabet (AL)</th>
<th>Alpha numeric (ANE)</th>
<th>External Decimal (ED)</th>
<th>Binary</th>
<th>Numeric Edited (NE)</th>
<th>Alphanumeric Edited (ANE)</th>
<th>FIGCON ZERO</th>
<th>FIGCON</th>
<th>National Data Item (NDI)</th>
<th>National Numeric Data Item (NNDI)</th>
<th>Index Name (IN)</th>
<th>Index Data Item (IDI)</th>
<th>Pointer Data Item (PTR)</th>
<th>Address of (@)</th>
<th>Floating Point Literal</th>
<th>Internal Floating Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NU</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NU</td>
</tr>
<tr>
<td>AL</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NU</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NU</td>
</tr>
<tr>
<td>AN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NU</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NU</td>
</tr>
<tr>
<td>NE</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NU</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NU</td>
</tr>
<tr>
<td>BI</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NU</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NU</td>
</tr>
<tr>
<td>ED</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NU</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NU</td>
</tr>
<tr>
<td>ANE</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NU</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NU</td>
</tr>
<tr>
<td>NDI</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NU</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NU</td>
</tr>
<tr>
<td>NNDI</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NU</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
<td>NU</td>
<td>NU</td>
</tr>
<tr>
<td>OPERAND</td>
<td>GR</td>
<td>AL</td>
<td>AN</td>
<td>ED</td>
<td>BI</td>
<td>NE</td>
<td>ANE</td>
<td>NDI</td>
<td>NNDI</td>
<td>ID</td>
<td>IN</td>
<td>IDI</td>
<td>PTR</td>
<td>@</td>
<td>IF</td>
<td>EF</td>
<td>D1</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>----</td>
<td>----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>External Floating Point (EF)</td>
<td>NN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NU</td>
<td></td>
</tr>
<tr>
<td>DBCS data item (D1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NU</td>
<td></td>
</tr>
<tr>
<td>DBCS Literal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NU</td>
<td></td>
</tr>
<tr>
<td>Address hex Literal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NU</td>
<td></td>
</tr>
<tr>
<td>National Literal DBCS</td>
<td>NN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NU</td>
<td></td>
</tr>
<tr>
<td>National Hex Literal</td>
<td>NN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NU</td>
<td></td>
</tr>
</tbody>
</table>
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.

```
IF expression THEN command ELSE command ;
```

equation
A valid Debug Tool PL/I expression.

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

command
A valid Debug Tool command.
When IF commands are nested and ELSE clauses are present, a given ELSE is associated with the closest preceding IF clause within the same block.

**Usage notes**
- An ELSE clause should always be included if the IF clause causes Debug Tool to get more input (for example, an IF containing USE or other commands that cause Debug Tool to be restarted because an AT-condition occurs).
- The if command cannot be used while you replay recorded statements by using the LAYBACK commands.

**Examples**
- If the value of array1 is equal to the value of array2, go to the statement with label constant label_1. Execution of the user program continues at label_1. If array1 does not equal array2, the GOTO is not performed and control is passed to the user program.
  
  IF array1 = array2 THEN GOTO LABEL label_1; ELSE GO;
- Set a breakpoint at statement 23, which will test if variable j is equal to 10, display the names and values of variables rmdr, totodd, and terms(j). If variable j is not equal to 10, continue program execution.
  
  AT 23 IF j = 10 THEN LIST TITLED (rmdr, totodd, terms(j)); ELSE GO;

**IMMEDIATE command (full-screen mode)**

The IMMEDIATE command causes a command within a command list to be performed immediately. It is intended for use with commands assigned to a PF key.

IMMEDIATE can only be entered as an unnested command or within a compound command.

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.

```
IMMEDIATE—command—;
```

**command**

One of the following Debug Tool commands:

- FIND
- FINDBP
- RETRIEVE
- SCROLL commands
  - BOTTOM
  - DOWN
  - LEFT
  - NEXT
  - RIGHT
  - TO
  - TOP
  - UP
- WINDOW commands
  - CLOSE
  - OPEN
  - SIZE
Usage notes

- The IMMEDIATE command is not logged.

Examples

- Specify that the WINDOW OPEN LOG command be immediately effective.
  IMMEDIATE WINDOW OPEN LOG;
- Specify that the SCROLL BOTTOM command be immediately effective.
  IMMEDIATE SCROLL BOTTOM;

### INPUT command (C, C++, and COBOL)

The INPUT command provides input for an intercepted read and is valid only when there is a read pending for an intercepted file. The INPUT keyword cannot be abbreviated.

```
INPUT text ;
```

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

**Usage notes**

- The text consists of everything between the INPUT keyword and the semicolon (or end-of-line). Any leading or trailing blanks are removed by Debug Tool.
- If a semicolon (;) is included as part of the 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 Debug Tool prompt you for input to a sequential file. The prompt and the file’s name appears in the Command Log.

To substitute the input that would have come from the DD name specified by the SET INTERCEPT ON command with your desired input, enter:

```
INPUT text you want to input ;
```

Program input is recorded in your Log window.

A closing semicolon (;) is required for this command. Everything between the INPUT keyword and the semicolon is considered input text. If you want to include a semicolon, 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 229
- "SET INTERCEPT command (COBOL, full-screen mode, line mode, batch mode)" on page 230

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.

\[\text{JUMPTO } \text{statement_id};\]

Usage notes
- You cannot use the JUMPTO command while you debug a disassembled program.
- If you specify the JUMPTO command in a command list (for example, as the subject of an IF command or WHEN clause), all subsequent commands in the list are ignored.
- If the program is compiled with minimum optimization, the JUMPTO command is not restricted to specific statements.
- You cannot use the JUMPTO command while you replay recorded statements by using the PLAYBACK command.
- For C, C++, and PL/I programs, statements can be removed by the compiler during optimization. Specify a reference or statement for the JUMPTO command that can be reached while the program is running. You can use the LIST STATEMENT NUMBERS command to determine the statements that can be reached.
- For PL/I programs, you can use JUMPTO in a command list on a call to PLITEST or CEETEST.
- For PL/I programs, you cannot specify a statement that is out of the currently active block. However, you might have to qualify the statement.
- For COBOL programs, the 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 4.1, 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...
4.1, 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 4.1, and specify the EJPD and NOHOOK suboption of the TEST compiler option. Specifying the EJPD suboption might cause some loss of optimization.

- You can use the JUMPTO command in remote debug mode only by entering it in the Action field, which is in the Optional Parameters section of the Add a Breakpoint task.

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
Debug Tool User’s Guide

Related references
“statement_id” on page 17

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.

JUMPTO LABEL 'statement_label'

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

Usage notes
- Use the syntax of statement_label enclosed in apostrophes (’) only for non-Language Environment COBOL programs. It is not supported in any other programming language.
- In PL/I, out-of-block JUMPTOs are allowed. However, qualification might be needed.
- The LABEL keyword is optional when either the target statement_label is nonnumeric or if it is qualified (whether the actual label was nonnumeric or not). A COBOL statement_label can have either of the following forms:
– name
This form can be used in COBOL for reference to a section name or for a COBOL paragraph name that is not within a section or is in only one section of the block.

– name1 OF name2 or name1 IN name2
This form must be used for any reference to a COBOL paragraph (name1) that is within a section (name2), if the same name also exists in other sections in the same block. You can specify either OF or IN, but Debug Tool always uses OF for output.

Either form can be prefixed with the usual block, compile unit, and load module qualifiers.

• For C, to be able to use the JUMPTO LABEL command, you must compile your program in one of the following ways:
  – With either the PATH or ALL suboption and the SYM suboption of the TEST compiler option.
  – 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 4.1
  – 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

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

**Related references**
- "statement_label" on page 18

---

**LIST command**

The LIST command displays information about a program such as values of specified variables, structures, arrays, registers, statement numbers, frequency information, and the flow of program execution. The LIST command can be used to display information in any enclave. All information displayed will be saved in the log file.

The following table summarizes the forms of the LIST command.

<table>
<thead>
<tr>
<th>LIST command format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;LIST (blank) command&quot; on page 142</td>
<td>Displays Source Identification panel</td>
</tr>
<tr>
<td>&quot;LIST AT command&quot; on page 142</td>
<td>Lists the currently defined breakpoints.</td>
</tr>
<tr>
<td>&quot;LIST CALLS command&quot; on page 145</td>
<td>Displays the dynamic chain of active blocks.</td>
</tr>
<tr>
<td>&quot;LIST CONTAINER command&quot; on page 146</td>
<td>Displays the contents of a container.</td>
</tr>
<tr>
<td>&quot;LIST CURSOR command (full-screen mode)&quot; on page 147</td>
<td>Displays the variable pointed to by the cursor.</td>
</tr>
<tr>
<td>&quot;LIST DTCN or CADP command&quot; on page 148</td>
<td>Lists the load modules, programs, and compile units that were disabled by the DISABLE command.</td>
</tr>
<tr>
<td>&quot;LIST expression command&quot; on page 149</td>
<td>Displays values of expressions.</td>
</tr>
<tr>
<td>&quot;LIST FREQUENCY command&quot; on page 153</td>
<td>Lists statement execution counts.</td>
</tr>
<tr>
<td>&quot;LIST LAST command&quot; on page 154</td>
<td>Displays a list of recent entries in the history table.</td>
</tr>
<tr>
<td>&quot;LIST LINE NUMBERS command&quot; on page 155</td>
<td>Lists all line numbers that are valid locations for an AT LINE breakpoint.</td>
</tr>
<tr>
<td>&quot;LIST LINES command&quot; on page 155</td>
<td>Lists one or more lines from the current listing or source file displayed in the Source window.</td>
</tr>
<tr>
<td>&quot;LIST MONITOR command&quot; on page 155</td>
<td>Lists the current set of MONITOR commands.</td>
</tr>
<tr>
<td>&quot;LIST NAMES command&quot; on page 155</td>
<td>Lists the names of variables, programs, or Debug Tool procedures.</td>
</tr>
<tr>
<td>&quot;LIST ON (PL/I) command&quot; on page 157</td>
<td>Lists the action (if any) currently defined for the specified PL/I conditions.</td>
</tr>
<tr>
<td>&quot;LIST PROCEDURES command&quot; on page 158</td>
<td>Lists the commands contained in the specified Debug Tool procedure.</td>
</tr>
</tbody>
</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 Debug Tool 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 180](#)

### LIST AT command

Lists the currently defined breakpoints, including the action taken when the specified breakpoint is activated. If no action is defined, Debug Tool displays the NULL command.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST REGISTERS command</td>
<td>Displays the current register contents.</td>
</tr>
<tr>
<td>LIST STATEMENT NUMBERS command</td>
<td>Lists all statement numbers that are valid locations for an AT STATEMENT breakpoint.</td>
</tr>
<tr>
<td>LIST STATEMENTS command</td>
<td>Lists one or more statements from the current listing or source file displayed in the Source window.</td>
</tr>
<tr>
<td>LIST STORAGE command</td>
<td>Provides a dump-format display of storage.</td>
</tr>
</tbody>
</table>
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 39

**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 every active enclave in the process.

**Usage notes**
• For programs containing interlanguage communication (ILC), routines from previous enclaves are only listed if they are written in a language that is active in the current enclave.
• If the enclave was created with the system() function, compile units in parent enclaves are not listed.
• If you are debugging a program that does not follow the standard linkage conventions for R13, R14, and R15, the output of the LIST CALLS command can be incorrect or incomplete.
• If you are debugging a disassembled program and you encounter one of the following situations:
  – The registers’ save area has not been created.
  – The registers are not chained to the other save areas.
Some of the programs or CSECTs in the call chain are not displayed.
• The LIST CALLS command cannot be used while you replay recorded statements by using the PLAYBACK commands.

**Example**
Display the current dynamic chain of active blocks.
LIST CALLS;
**LIST CONTAINER command**

Displays the contents of a container.

```plaintext
<ListContainer><channel_name> container_name</channel_name></ListContainer>
```

- `channel_name`: The name of the channel that Debug Tool searches through to find a container. If you do not provide a channel name, Debug Tool 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 Debug Tool log. Otherwise, Debug Tool displays a formatted version of the XML document in the Debug Tool 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. Debug Tool uses the z/OS Unicode Services to convert the characters in the XML from
this code page to the code page specified in EQAOPTS 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, 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, Debug Tool 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 Debug Tool.

Examples

- For PL/I, 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, 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 Debug Tool User’s Guide

Related references

- “DESCRIBE command” on page 102
- Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

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.

```
LIST CURSOR;
```

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.

```
LIST DTCN ;
LIST CADP
```

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 112
- “DISABLE command” on page 106
- “LIST DTCN or CADP command”
- Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393
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 non-Language Environment COBOL, this is the default for expressions that are valid as receivers of a Debug Tool assembler assignment statement.

If you specify TITLED with no keyword, all variables in the currently qualified block are listed. If you specify TITLED with an asterisk (*) and you are debugging a C, C++, or COBOL program, all variables in the currently qualified compile unit are listed.

If you are debugging a COBOL program, the following additional options are available with TITLED:

FS Lists all variables defined in the COBOL File Section in the currently qualified compile unit.

WSS Lists all variables defined in the COBOL Working-Storage Section in the currently qualified compile unit.

LS Lists all variables defined in the COBOL Linkage Section in the currently qualified compile unit.

LOS List all variables defined in the COBOL Local-Storage Section in the currently qualified compile unit.

* (C, C++, and COBOL)
Lists all variables in the currently qualified compile unit.

UNTITLED
Lists expression values without displaying the expressions themselves. For C and C++, this is the default for expressions that are not lvalues. For COBOL, this is the default for expressions consisting of only a single constant. For
assembler, disassembly, and non-Language Environment COBOL, this is the
default for expressions that are not valid as receivers of a Debug Tool
assembler assignment statement. For the LIST command, an expression also
includes character strings enclosed in either 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 can not specify
expressions.

expression
An expression valid in the current programming language other than
non-Language Environment COBOL.

'expression'
A valid non-Language Environment COBOL expression enclosed in
apostrophes (').

Usage notes
• If you want to use the LIST TITLED with the parameters FS, WSS, LS or LOS, the
  PTF for Language Environment APAR PK12834 must be installed on z/OS
• For COBOL programs, if you want to use the LIST TITLED command with a
  variable that is named FS, WSS, LS, or LOS, you must enclose the name of the
  variable in parenthesis. For example, the command LIST TITLED (FS) lists the
  variable FS; the command LIST TITLED FS lists the variables in the File Section.
• Debug Tool allows you to abbreviate many commands. This might result in
  unexpected results when you use the LIST command with a single-letter
  expression. For example, LIST A can be interpreted as the LIST AT command,
  which lists all breakpoints. However, if you wanted to display the value of a
  variable labeled A in your program, you need to use parenthesis: LIST (A).
• If LIST TITLED * is specified and your compile unit is large, slow performance
  might result.
• For COBOL, if LIST TITLED * is specified and your compile unit is large, you
  might receive an out of storage error message.
• For COBOL, the LIST command can reference a condition name, a file name, or
  an expression.
• For optimized COBOL programs, the LIST command cannot reference a variable
  that was discarded by the optimizer.
• When using LIST TITLED with no parameters within the PL/I compile unit, only
  the first element of any array will be listed. If the entire array needs to be listed,
  use LIST and specify the array name (i.e., LIST array where array is the name of
  an array).
• If a character variable contains character data that cannot be displayed in its
displayed data type, Debug Tool displays the data with a special character. The
  topic "How Debug Tool handles characters that can't be displayed in their
  declared data type" in Debug Tool User's Guide describes what Debug Tool 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.

If your program is compiled with Enterprise PL/I or Enterprise COBOL, you can enter the L prefix command through the Source window prefix area to display the value of the variables on that line.

Examples

- Display the values for variables size and r and the expression c + r, with their respective names.
  
  ```lang
  LIST TITLED (size, r, c + r);
  ```

- Display the COBOL references as if they were elementary items. The current programming language setting is COBOL.
  
  ```lang
  LIST (GROUP x OF z(1,2), GROUP a, w);
  ```

- Display the value of the Debug Tool variable %ADDRESS.
  
  ```lang
  LIST %ADDRESS;
  ```

- In the disassembly view, display the value of register 1 (R1), which is the value of Debug Tool variable %R1.
  
  ```lang
  LIST R1;
  ```

- In COBOL, display the names and values of variables defined in the File Section.
  
  ```lang
  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 233
- "L prefix command (full-screen mode)"

**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.

```lang
L
integer
integer
integer
integer
```

**integer**

Identifies specific operands to be listed. If you do not specify an integer, Debug Tool lists all operands. If you use a single number or the form 1,2,3, Debug Tool lists the specified operand or operands. If you use the form 1-4, Debug Tool 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. Debug Tool 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`, Debug Tool lists all operands referenced explicitly or implicitly by the instruction. If you specify any form of `integer`, Debug Tool lists only the operands explicitly referenced by the specified operand or operands.

**Usage notes**

- The `L` prefix command can be entered only on lines that have valid 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
- 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
    ```
  - Debug Tool 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 Debug Tool 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, Debug Tool does number these fields when it numbers the operands.
  - For instructions that might be coded using extended mnemonics (BC, BCR, and BRC), Debug Tool 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:

```plaintext
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 IN0 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 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 L on line 201. Debug Tool 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. Debug Tool lists R1 and R4.

• Enter L1 on line 202. Debug Tool 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.

```
LIST FREQUENCY [LINES | STATEMENTS] statement_id_range...
```

* Lists frequency for all statements in the currently qualified compile unit. If currently executing at the AT TERMINATION breakpoint where there is no qualification available, it will list frequency for all statements in all the compile units in the terminating enclave where frequency data exists.

LINES
Displays the source line after the frequency count.

STATEMENT
Equivalent to LINES.

Usage notes
• In the disassembly view, LIST FREQUENCY and LIST FREQUENCY * are not supported.
• When you replay recorded statements by using the PLAYBACK commands, the frequency count is not updated.
Examples

- List frequency for statements 1-20.
  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 17
- “SET FREQUENCY command” on page 227

LIST LAST command

Displays a list of recent entries in the history table.

```
LIST LAST integer HISTORY LINES PATHS STATEMENTS;
```

integer

Specifies the number of most recently processed breakpoints and conditions displayed.

HISTORY

Displays all processed breakpoints and conditions.

LINES

Displays processed statement or line breakpoints. LINES is equivalent to STATEMENTS.

PATHS

Displays processed path breakpoints.

STATEMENTS

Is equivalent to LINES.

Usage notes

- The LAST keyword is provided to make the LIST command readable. It does not perform any function.
- In the disassembly view, LIST LAST is not supported.

Examples

- Display all processed path breakpoints in the history table.
  LIST PATHS;
- Display all program breakpoints and conditions for the last five times Debug Tool gained control.
  LIST LAST 5 HISTORY;
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
“SET HISTORY command” on page 228

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 160

LIST MONITOR command

Lists all or selected members of the current set of MONITOR commands and any suspended MONITOR LOCAL commands.

\[ \text{LIST MONITOR } \text{integer} \quad \text{or} \quad \text{LIST MONITOR integer - integer} \]

\[ \text{integer} \]

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 Debug Tool procedures. If LIST NAMES is issued with no keyword specified, the names of all program and session variables that can be referenced in the current programming language and that are visible to the currently qualified block are displayed. A subset of the names can be specified by supplying a pattern to be matched.
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:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>^ABC^</td>
<td>&quot;ABC&quot;</td>
</tr>
<tr>
<td>'A5'</td>
<td>&quot;A5&quot;</td>
</tr>
<tr>
<td>'MY'</td>
<td>'MY'</td>
</tr>
</tbody>
</table>

Pattern matching is not case-sensitive outside of DBCS. Both the pattern and potential names outside of shift codes are effectively uppercased, except when the current programming language setting is C. Letters in the pattern must be the correct case when the current programming language setting is C.

**BLOCK**
Displays variable names that are defined within one or more specified blocks.

**CUS**
Displays the compile unit names. **CUS** is equivalent to **PROGRAMS**.

**PROCEDURES**
Displays the Debug Tool procedure names.

**PROGRAMS**
Is equivalent to **CUS**.

**TEST**
Displays the Debug Tool session variable names.

**Usage notes**
- **LIST NAMES CUS** applies to compile unit names.
- **LIST NAMES TEST** shows only those session variable names that can be referenced in the current programming language.
- The output of **LIST NAMES** without any options depends on both the current qualification and the current programming language setting. If the current programming language differs from the programming language of the current
qualification, the output of the command shows only those session variable names that can be referenced in the current programming language.

- For structures, the pattern is tested against the complete name, hence "B" is not satisfied by "C OF B OF A" (COBOL).
- If the DATA option of the PLAYBACK ENABLE command is in effect for the current compile unit, you can use the LIST NAMES command while you replay recorded statements by using the PLAYBACK commands.
- For optimized COBOL programs, the LIST NAMES command does not display variables discarded by the optimizer.

Examples

- Display all compile unit names that begin with the letters "MY" and end with "5". The current programming language setting is either C or COBOL.

  ```plaintext
  LIST NAMES "MY*5" PROGRAMS;
  ```

- Display the names of all the Debug Tool procedures that can be called.

  ```plaintext
  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.

  ```plaintext
  LIST NAMES 'R*' BLOCK (mainprog);
  ```

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 14

LIST ON (PL/I) command

Lists the action (if any) currently defined for the specified PL/I conditions.

```plaintext
LIST ON [pli_condition];
```

**pli_condition**

A valid PL/I condition specification. If omitted, all currently defined ON command actions are listed.

Usage notes

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

Example

List the action for the ON ZERO DIVIDE command.

```plaintext
LIST ON ZERO DIVIDE;
```

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

Related references

- "ON command (PL/I)" on page 178
LIST PROCEDURES command

Lists the commands contained in the specified Debug Tool PROCEDURE definitions.

```
LIST PROCEDURES <name> ;
```

`name`
A valid Debug Tool procedure name. If no procedure name is specified, the commands contained in the currently running procedure are displayed. If no procedure is currently running, an error message is issued.

Usage note

Examples

- Display the commands in the Debug Tool procedure p2.
  ```
  LIST PROC p2;
  ```
- List the procedures abc and proc7.
  ```
  LIST PROCEDURES (abc, proc7);
  ```

LIST REGISTERS command

Displays the current register contents.

```
LIST <32BIT|64BIT|LONG|FLOATING> REGISTERS ;
```

REGISTERS
Displays the General Purpose Registers (%GPRn). When this command is issued when you are qualified to an Assembler or Disassembly CU other than the CU where execution was suspended, it also displays the values of the %Rn symbols.

- **32BIT**
  Displays the 32-bit decimal General Purpose Registers (%GPRn).

- **64BIT**
  Displays the 64-bit decimal General Purpose Registers (%GPRGn).

- **LONG**
  Displays the decimal value of the long-precision floating-point registers.

- **SHORT**
  Displays the decimal value of the short-precision floating-point registers.

- **FLOATING**
  Displays the long-precision floating-point registers.

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, only the 32-bit general-purpose registers are displayed.

Examples
- Display the General Purpose Registers at the point of a program interruption:
  ```plaintext
  LIST REGISTERS;
  ```
- Display the floating-point registers.
  ```plaintext
  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.

```
LIST STATEMENT NUMBERS [block_spec, cu_spec, statement_id_range];
```

**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.
  ```plaintext
  LIST STATEMENT NUMBERS;
  ```
- Display the statement or line number of every statement in block earnings.
  ```plaintext
  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 14
- “statement_id_range and stmt_id_spec” on page 17
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.

```
LIST LINE statement_id_range;
```

Usage notes

- The specified lines are displayed in the same format as they would appear in the full-screen Source window, except that wide lines are truncated.
- You might need to specify a range of line numbers to ensure that continued statements are completely displayed.
- This command is not to be confused with the LIST LAST STATEMENTS command.
- In the disassembly view, LIST STATEMENTS is not supported.

Examples

- List lines 25 through 30 in the source file associated with the currently qualified compile unit.
  LIST LINES 25 - 30;
- List statement 100 from the current program listing file.
  LIST STATEMENT 100;

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

**Related references**

- "statement_id_range and stmt_id_spec" on page 17

LIST STORAGE command

Displays the contents of storage at a particular address in hexadecimal or XML format.

```
LIST STORAGE address
    reference
    reference
    length
    offset
XML
    EBCDIC
    ASCII
    CODEPAGE(ccsid)
```

**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.
A non-Language Environment COBOL variable whose storage location is to be listed. A non-Language Environment 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 Debug Tool log. Otherwise, Debug Tool displays a formatted version of the XML document in the Debug Tool 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. Debug Tool uses the z/OS Unicode Services to convert the characters in the XML from this code page to the code page specified in EQAOPTS 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, Debug Tool displays the contents at the address given by that pointer.
- Using Debug Tool, cursor pointing can be used by typing the LIST STORAGE command on the command line and moving the cursor to a variable in the Source window before pressing Enter, or by moving the cursor and pressing a PF key with the LIST STORAGE command assigned to it.
- When using the LIST STORAGE command in Debug Tool for a variable that is located by the cursor position, the variable’s name cannot be split across different lines of the source listing.
- If the referenced variable is a General Purpose Register (GPR) such as %GPR1, the result depends on the programming language that is in effect:
  - For all languages except assembler and disassembly, Debug Tool displays the storage at the address contained in the referenced GPR.
  - For assembler and disassembly, you must use the indirection notation (%GPR1->) to instruct Debug Tool to display the storage at the address contained in the referenced register.
- If no operand is specified with LIST STORAGE, the command is cursor-sensitive.
If you are replaying recorded statements by using PLAYBACK commands, the LIST STORAGE command displays the contents of storage at the point where you entered the PLAYBACK START command.

For optimized COBOL programs, LIST STORAGE cannot display variables that were discarded by the optimizer.

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, Debug Tool 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 Debug Tool.

If you specify address with more than 8 significant digits or if reference references 64-bit addressable storage, Debug Tool assumes that the storage location is 64-bit addressable storage. Otherwise, Debug Tool 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 16

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.

```
LOAD module_name

(module_name)

LE
NONLE
```

*module_name*

The name of one or more load modules to be loaded by Debug Tool.

**LE** Use the Language Environment enclave-level load service to load the load module or modules. The load module or modules remain active until the current enclave terminates or you enter a CLEAR LOAD command for the load module or modules.

**NONLE**
Use non-Language Environment services to load the load module or modules. The load module or modules remain active until the debugging task terminates or you enter a CLEAR LOAD command for the load module or modules. For CICS programs, the load module or modules are loaded by using EXEC CICS LOAD. For all other programs, the load module or modules are loaded by using the MVS LOAD services.

**Usage notes**

• You can use this command in remote debug mode.
• You can enter the QUALIFY CU command for a program or CSECT in the load module or load modules that you just loaded unless the program is COBOL.
• If you set breakpoints in the programs or CSECTS in the module and then the same load module is loaded again, the breakpoints might not work because location of the load module has changed.
• If the module to be debugged is RESIDENT or was loaded before Debug Tool was started, you can use the LOAD command to make the module known to Language Environment.
• 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 B, “Debug Tool commands supported in remote debug mode,” on page 393
LOADDEBUGDATA command

Specifies that a compile unit (CU) is an assembler or non-Language Environment COBOL CU and loads debug data from the default data set name, userid.EQALANGX(cu_name). If the debug data is stored in a different data set, you can specify that data set name by using the SET SOURCE, SET DEFAULT LISTINGS command, or the EQADEBUG DD statement. In remote debug mode, you can specify the data set name by using the EQADEBUG DD statement or let the remote debugger prompt you for the data set name.

You can generate the required debug data by using the EQALANGX program or, if you are debugging an assembler program, by assembling your program through Debug Tool Utilities. Both methods are described in Debug Tool User’s Guide.

```
 LOADDDEBUGDATA
   LDD
   load_module_name::>cu_name
   (load_module_name::>cu_name)
```

**load_module_name**

The name of the load module containing the specified compile unit (cu_name). If the corresponding load module is known to Debug Tool, the specified compile unit must be a disassembly compile unit within the specified load module. If the load module is not known to Debug Tool, Debug Tool defers the LOADDEBUGDATA command until a load module by the specified name and containing the specified compile unit is loaded.

If you do not specify load_module_name, Debug Tool 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 non-Language Environment COBOL compile unit. If this compile unit is currently known to Debug Tool, it must be a disassembly compile unit. If it is not currently known to Debug Tool, the LOADDEBUGDATA command is deferred until a disassembly compile unit by the specified name becomes known to Debug Tool.

**Usage notes**

- When you use the SET SAVE command to save breakpoints or monitor specifications or you use the RESTORE command to restore breakpoints or monitor specifications, all LDD settings including the data set name of the data set from which the debug data was loaded is saved and restored.
- For CICS only: When a DTCN profile is active for a full screen mode debugging session, Debug Tool preserves all LDD settings, including the data set name of the data set from which the debug data was loaded, until the DTCN profile is deleted or the terminal session is terminated.
- You can use this command for assembler CUs (but not for non-Language Environment COBOL CUs) in remote debug mode.
- After Debug Tool successfully processes a LOADDEBUGDATA command for a CU, if the CU is deleted and then appears later, an implicit LDD command is run for the CU using the same EQALANGX data set that was used initially.
- You cannot enter the LDD command for the same compile unit more than once.
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, Debug Tool displays an error message.

The MEMORY command cannot be saved and restored.

```
MEMORY address

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 non-Language Environment 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, Debug Tool assumes that the storage location is 64-bit addressable storage. Otherwise, Debug Tool assumes that the storage location is 31-bit addressable storage.
• For C and C++, if reference is a pointer, Debug Tool 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**
- "Debug Tool session panel" in [Debug Tool User's Guide](#)
- "Switching between the Memory window and Log window" in [Debug Tool User's Guide](#)
- "Displaying the Memory window" in [Debug Tool User's Guide](#)
- "Displaying memory through the Memory window" in [Debug Tool 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**

Debug Tool maintains a list of your most recently entered MONITOR commands. Each command entered is assigned a number between 1 and 99 or you can assign it a number. Use these numbers to indicate to Debug Tool which MONITOR command you want to redefine.

```
GLOBAL Specify that the monitor definition is global. That is, it is not associated with a particular compile unit.
LOCAL Specify that the monitor definition is local to a specific compile unit. Using Debug Tool, the specified output is displayed only when the current qualification is within the associated compile unit.
```

---

MONITOR command definitions:
cu_spec
   A valid compile unit specification. This specifies the compile unit
   associated with the monitor definition.

integer
   An integer in the range 1 to 99, indicating what command in the list is
   replaced with the specified command and the order that the monitored
   commands are evaluated. If omitted, the next monitor integer is assigned. An
   error message is displayed if the maximum number of monitoring commands
   already exists.

command
   A DESCRIBE, LIST, Null, or QUERY command whose output is displayed in the
   monitor window or log file.

HEX
   Specifies that the value of the variable be displayed in hexadecimal format.
   You can specify the HEX parameter only with a MONITOR LIST expression
   command or the MONITOR n command where n is the nth command in the
   MONITOR list and it must be a LIST expression command.

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 nth 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 Debug Tool, monitor output is presented in monitor number sequence.
   • If a number is provided and a command omitted, a Null command is inserted
     on the line corresponding to the number in the monitor window. This reserves
     the monitor number.
   • You can only specify a monitor number that is at most one greater than the
     highest existing monitor number.
   • To clear a command from the monitor, use the CLEAR MONITOR command.
   • Replacement only occurs if the command identified by the monitor number
     already exists.
   • When SET AUTOMONITOR ON is in effect, Debug Tool 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, Debug Tool inserts the
     new MONITOR command in the last active entry and uses the next higher entry for
     SET AUTOMONITOR ON.
   • 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 values) display identifying information with the values, whereas expressions and literals do not.

The GLOBAL and LOCAL keywords also affect the default qualification for evaluation of an expression. GLOBAL indicates that the default qualification is the currently executing point in the program. LOCAL indicates that the default qualification is to the compile unit specified.

LOCAL monitors are 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.

If your program is compiled with Enterprise PL/I or Enterprise COBOL, you can enter the M prefix command through the Source window prefix area to add the variables on that line to the Monitor window.

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 14
- “CLEAR command” on page 88
- “DESCRIBE command” on page 102
- “LIST command” on page 141
- “M prefix (full-screen mode)” on page 169
- “QUERY command” on page 190
- “SET MONITOR command” on page 237
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, Debug Tool monitors all operands. If you enter a single number or the form 1,2,3, Debug Tool monitors the specified operand or operands. If you use the form 1-4, Debug Tool 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. Debug Tool 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, Debug Tool adds all operands referenced explicitly or implicitly by the instruction to the Monitor window. If you specify any form of integer, Debug Tool adds only the operands explicitly referenced by the specified operand or operands to the Monitor window.

Usage notes

- The M prefix command can be entered only on lines that have valid 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
- 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
  - Debug Tool 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 Debug Tool stopped the
program at the instruction you referenced.

- When you specify an explicit base or index register in an operand, Debug
  Tool computes the effective address of the storage location when you enter
  the M prefix command. Debug Tool does not recompute the effective address
  while it monitors the operand.

- When you specify a single symbol as a machine instruction operand, Debug
  Tool uses the current value of any base register and the currently active
  USING as Debug Tool monitors the operand.

- The M prefix command cannot access mask fields, immediate data fields, and
  any other constants imbedded in the machine instructions. However, Debug
  Tool does number these fields when it numbers the operands.

- For instructions that might be coded using extended mnemonics (BC, BCR,
  and BRC), Debug Tool 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. Debug Tool 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. Debug Tool adds R1 and R4 to the Monitor window.
- Enter M1 on line 202. Debug Tool 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 166]
MOVE command (COBOL)

The MOVE command transfers data from one area of storage to another. The keywords cannot be abbreviated.

```
MOVE reference TO reference; literal
```

**reference**
A valid Debug Tool COBOL reference.

**literal**
A valid COBOL literal.

**Usage notes**

- If Debug Tool was started because of a computational condition or an attention interrupt, using an assignment to set a variable might not give expected results. This is due to the uncertainty of variable values within statements as opposed to their values at statement boundaries.
- MOVE assigns a value only to a single receiver; unlike COBOL, multiple receiver variables are not supported.
- The COBOL CORRESPONDING phrase is not supported.
- MOVE does not support date windowing. Therefore, you cannot use the MOVE command to assign the value of a windowed date field to an expanded date field or to a 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.
- If the DATA parameter of the PLAYBACK ENABLE command is in effect for the current compile unit, the MOVE command can be used while you replay recorded statements by using the PLAYBACK commands. The target of the MOVE command must be a session variable, not a program variable.
- If you are debugging an optimized COBOL program, you can use the MOVE command to assign a value to a program variable only if you first enter the SET WARNING OFF command.
- If you are debugging a COBOL program that was compiled with the OPTIMIZE compiler option, neither operand of the MOVE command can be a variable that was discarded by the optimizer.
- If a COBOL variable defined as National is used as the receiving field in a MOVE command with an alphabetic or alphanumeric operand, the operand that is not National is converted to Unicode before that move is done, except for Group items. See [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 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)"
"SET WARNING command (C, C++, COBOL, and PL/I)" on page 258

Allowable moves for the MOVE command (COBOL)
The following table shows the allowable moves for the Debug Tool MOVE command.
<table>
<thead>
<tr>
<th>Source Field</th>
<th>GR</th>
<th>AL</th>
<th>AN</th>
<th>ED</th>
<th>NE</th>
<th>BI</th>
<th>D1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group (GR)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Alphabetic (AL)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Alphanumeric (AN)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>External Decimal (ED)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Binary (BI)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Externally Edited (ED)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Alphabetical (AL)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Alphanumeric Edition (ANE)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Figcon Zero</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Spaces (AL)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>High-Value, Low-Value, Quotes</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>National Data Item (NDI)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Numerical Literal</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Alphanumeric Literal</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Internal Decimal (ID)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Floating Point</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Internal Floating Point (IF)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Externally Edited (EED)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>DBCS Data Item (D1)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Source field</td>
<td>GR</td>
<td>AL</td>
<td>AN</td>
<td>ED</td>
<td>BI</td>
<td>NE</td>
<td>ANE</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>-----</td>
</tr>
<tr>
<td>DBCS LITERAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NATIONAL LITERAL (NL)</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NATIONAL HEX LITERAL (NHL)(^7)</td>
<td>Y(^l)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Notes:
1 Move without conversion (like AN to AN)
2 Numeric move
3 Decimal-aligned and truncated, if necessary
4 MOVE does not support date windowing. For example, the MOVE statement cannot be used to move a windowed date field to an expanded date field, or to a nondate field.
5 The MOVE command cannot be used to move one windowed date field to another windowed date field with a different DATE FORMAT clause, or to move one expanded date field to another expanded date field with a different DATE FORMAT clause.
6 Must be hexadecimal characters only, delimited by either 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.

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 171

**NAMES command**

Use the NAMES command only as instructed in “Debugging user programs that use system prefixed names” in Debug Tool 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 Debug Tool excludes by default are not included in the list that is displayed.

```
USER
ALL
INCLUDED
EXCLUDED
LOADMODS
CUS
```

**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 Debug Tool 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 Debug Tool User’s Guide

Related references

- Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

NAMES EXCLUDE command

The NAMES EXCLUDE command enables you to indicate to Debug Tool the names of load modules or compile units that you do not need to debug. If these are data-only modules, Debug Tool does not process them. If they contain executable code, Debug Tool might process them in some cases. See "Optimizing the debugging of large applications" in the Debug Tool User’s Guide for more information about these situations.

```
NAMES EXCLUDE LOADMOD pattern ;
```

<table>
<thead>
<tr>
<th>LOADMOD</th>
<th>Indicates that you do not want to debug the specified load module.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CU</td>
<td>Indicates that you do not want to debug the specified compile unit.</td>
</tr>
<tr>
<td>NOTEST</td>
<td>Indicates that you do not want to debug any compile units that were not compiled with debug data.</td>
</tr>
</tbody>
</table>

**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 Debug Tool.
  
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 Debug Tool 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 code control statements into the EQAOPTS Debug Tool customization module with the equivalent NAMES EXCLUDE command. See “Using EQAOPTS to implement NAMES commands” in the Debug Tool User’s Guide for instructions.

• 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 Debug Tool User’s Guide
“Debugging programs containing data-only modules” in Debug Tool User’s Guide

Related references
Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

NAMES INCLUDE command

Use the NAMES INCLUDE command to indicate to Debug Tool 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 Debug Tool User’s Guide for more information.

```
  NAMES INCLUDE LOADMOD (name)
  NAMES INCLUDE CU (name)
```

**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 Debug Tool.
• You cannot use the NAMES INCLUDE command to indicate to Debug Tool 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 code control statements into the EQAOPTS Debug Tool customization module with the equivalent NAMES INCLUDE command. See “Using EQAOPTS to implement NAMES commands” in Debug Tool User’s Guide for instructions.
• Do not use the NAMES INCLUDE command to debug system components (for example, Debug Tool, 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 Debug Tool recognizes as system components.
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.

```
ON condition_name command;
ENDFILE (file_reference);
ENDPAGE
NAME PENDING
RECORD TRANSMIT
UNDEFINEDFILE
AREA ATTENTION
CONVERSION ERROR
FINISH FIXEDOVERFLOW
OVERFLOW SIZE
STRING RANGE STRINGSIZE
SUBSCRIPT RANGE UNDERFLOW
OVERFLOW ZERODIVIDE
```

condition_name
A valid PL/I CONDITION condition name.

file_reference
A valid PL/I file constant or file variable (can be qualified).
A valid Debug Tool command.

Usage notes

- You must abide by the PL/I restrictions for the particular condition.
- An ON action for a specified PL/I condition remains established until:
  - Another ON command establishes a new action for the same condition. In other words, the breakpoint is replaced.
  - A CLEAR command removes the ON definition.
- The ON command occurs before any existing ON-unit in your application program. The ON-unit is processed after Debug Tool returns control to the language.
- The following are accepted PL/I abbreviations for the PL/I condition constants:

  - ATTENTION or ATTN
  - FIXEDOVERFLOW or FOFL
  - OVERFLOW or OFL
  - STRINGRANGE or STRG
  - STRINGSIZE or STRZ
  - SUBSCRIPTRANGE or SUBRG
  - UNDEFINEDFILE([file_reference]) or UNDF([file_reference])
  - UNDERFLOW or UFL
  - ZERODIVIDE or ZDIV

- The preferred form of the ON command is AT OCCURRENCE. For compatibility with PLITEST and INSPECT, however, it is recognized and processed. ON should be considered a synonym of AT OCCURRENCE. Any ON commands entered are logged as AT OCCURRENCE commands.
- The ON command cannot be used while you replay recorded statements by using the PLAYBACK commands.

Examples

- Display a message if a division by zero is detected.

  ```pli
  ON ZERODIVIDE BEGIN;
  LIST 'A zero divide has been detected';
  END;
  ```

- Display and patch the error character when converting character data to numeric.

  Given a PL/I program that contains the following statements:

  ```pli
  DECLARE i FIXED BINARY(31,0);
  ...
  i = '1s3';
  ```

  The following Debug Tool command would display and patch the error character when converting the character data to numeric:

  ```pli
  ON CONVERSION
  BEGIN;
  LIST (%STATEMENT, ONCHAR);
  ONCHAR = '0';
  GO;
  END;
  ```

  '1s3' cannot be converted to a binary number so CONVERSION is raised. The ON CONVERSION command lists the offending statement number and the offending character: 's'. The data will be patched by replacing the 's' with a character zero, 0, and processing will continue.
Related references
“AT OCCURRENCE command” on page 67
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 Debug Tool session panel.

LAYOUT
Displays the Window Layout Selection panel that controls the configuration of the windows on the Debug Tool 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.

Debug Tool 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 Debug Tool by entering their names on the Source Identification panel.

The Source Identification panel associates compile units with the names of their respective listing, source, separate debug files and controls what appears in the Source window. To explicitly name the compile units being displayed in the Source window, access the Source Identification panel (shown below) by entering the PANEL LISTINGS or PANEL SOURCES command.
## Source Identification Panel

**Command ==>**

<table>
<thead>
<tr>
<th>Compile Unit</th>
<th>Listings/Source File</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBKP515</td>
<td>TS64081.TEST.LISTING(IBM63)</td>
<td>Y</td>
</tr>
</tbody>
</table>

Enter **QUIT** to return with current settings saved.
CANCEL to return without current settings saved.
UP/DOWN to scroll up and down.

### Compile Unit

Is the name of a valid compile unit currently known to Debug Tool. New compile units are added to the list as they become known.

### Listing/Source File

Is the name of the listing, source, 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.dsnname.Listing(membername)` (COBOL), or `userid.dsnname.List(membername)` (PL/I) for partitioned data sets.

### Display

Is a flag that specifies whether the listing or source is to be displayed in the Source window.

To display a listing view, take the following steps:

- Compile the program with the proper option to generate a source or source listing file.
- Make sure the file is available and accessible on your host operating system.
- Set the **Display** field on the Source Identification panel to **Y** for the compile unit. To save time and avoid displaying listings or source you do not want to see, specify **N**.

If any of these conditions are not satisfied, the Source window remains empty until control reaches a compile unit where the conditions are satisfied.

You can change the 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 Debug Tool session panel either by issuing the **QUIT** command, or by pressing the **QUIT PF** key.

**PROFILE**

Displays the Profile Settings panel, where parameters of a full-screen Debug Tool session can be set.

**SOURCES**

Is equivalent to **LISTINGS**.

### Usage notes
• All information 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 Debug Tool session with the current settings in effect. In addition, CANCEL can be used to leave the panels without saving the changes.

• The PANEL command is not logged.

Examples
• Display the color and attribute panel.
  PANEL COLORS;
• Reset the relative sizes of the windows for the current layout configuration.
  PANEL LAYOUT RESET;

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

Related tasks
  Debug Tool User’s Guide

PERFORM command (COBOL)
The PERFORM command transfers control explicitly to one or more statements and implicitly returns control to the next executable statement after execution of the specified statements is completed. The keywords cannot be abbreviated.

Simple:

```
   PERFORM command END-PERFORM;
```

*command*  
A valid Debug Tool command.

Repeating:

```
   PERFORM command BEFORE TEST WITH AFTER 
   VARYING reference FROM reference BY reference UNTIL condition 
   END-PERFORM;
```

*reference*  
A valid Debug Tool COBOL reference.
A simple relation condition.

A valid Debug Tool command.

Usage notes

- A constant as a reference is allowed only on the right side of the FROM and BY keywords.
- Index-names and floating point variables cannot be used as the VARYING references.
- Index-names are not supported in the BY phrase.
- Only inline PERFORMs are supported (but the performed command can be a Debug Tool procedure invocation).
- The COBOL AFTER phrase is not supported.
- Windowed date fields cannot be used as the VARYING reference, the FROM reference, or the BY reference.
- See [Enterprise COBOL for z/OS Language Reference](#) for an explanation of the following COBOL keywords:
  ```
  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.

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;PLAYBACK ENABLE&quot;</td>
<td>Informs Debug Tool to record all subsequent statements that you run and other information about your program.</td>
</tr>
<tr>
<td>&quot;PLAYBACK START command&quot; on page 185</td>
<td>Informs Debug Tool to suspend normal debugging and to prepare to replay recorded statements.</td>
</tr>
<tr>
<td>&quot;PLAYBACK FORWARD command&quot; on page 187</td>
<td>Informs Debug Tool to replay recorded statements in forward direction.</td>
</tr>
<tr>
<td>&quot;PLAYBACK BACKWARD command&quot; on page 187</td>
<td>Informs Debug Tool to replay recorded statements in backward direction.</td>
</tr>
<tr>
<td>&quot;PLAYBACK STOP command&quot; on page 187</td>
<td>Informs Debug Tool to stop replaying statements, resume normal debugging, and continue recording the statements that you run and other information about your program.</td>
</tr>
<tr>
<td>&quot;PLAYBACK DISABLE command&quot; on page 187</td>
<td>Informs Debug Tool to stop recording the statements that you run and discard the information about your program that it recorded.</td>
</tr>
</tbody>
</table>

### Usage note

You cannot use the PLAYBACK commands while you debug a disassembled program.

---

### PLAYBACK ENABLE command

The PLAYBACK ENABLE command informs Debug Tool to begin recording the statements that you run and information about your program. If Debug Tool is already recording the statements that you run, you can use the PLAYBACK ENABLE command to inform Debug Tool to record the statements that you run in other compile units or to change the effect of the DATA option.

```
PLAYBACK ENABLE options ;
```

**options:**

- `cuname`
- `integer` (optional) `NODATA` (optional)
- `DATA` (optional)

---

184 Debug Tool V10 Reference and Messages
**cuname**

Name of the compile unit or compile units where Debug Tool is to record the statements that you run. You can specify only the names of the compile units currently known.

* Specifies that Debug Tool is to record the statements that you run in all compile units. This is the default.

**integer**

Specifies the maximum amount of memory to use to store data that is collected. The integer value specifies a unit of K (1024) bytes. For example, an integer value of 2000 indicates 2,048,000 bytes. The default value is 8000.

**DATA**

Specifies that Debug Tool is to save information about your program, such as the value of variables and registers. Debug Tool saves this information for the compile units that you specify in the cuname parameter or, if you specified the * parameter, for all compile units. The DATA parameter is effective only for compile units compiled with the following compilers:

- 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 Debug Tool 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 Debug Tool to prepare to replay the statements it recorded. When normal debugging is suspended, all breakpoints are disabled and many commands are unavailable. Use the STEP and RUNTO commands to navigate through recorded statements in a forward or backward direction. Backward is the initial direction of the navigation.

```
$PLAYBACK—START—;
```

**Usage notes**
The following commands are available while you replay recorded statements:

<table>
<thead>
<tr>
<th>Command (COBOL)</th>
<th>LIST</th>
<th>Command (COBOL)</th>
<th>LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPUTE</td>
<td>LIST</td>
<td>MOVE</td>
<td>LIST</td>
</tr>
<tr>
<td>DECLARE</td>
<td>LIST</td>
<td>IF</td>
<td>LIST</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>LIST</td>
<td>SET AUTOMONITOR</td>
<td>LIST</td>
</tr>
</tbody>
</table>

1 Refer to “PERFORM command (COBOL)” on page 182 for restrictions.

If the DATA option is in effect and the compile unit supports the DATA option, the following commands are available:

<table>
<thead>
<tr>
<th>Command (COBOL)</th>
<th>LIST</th>
<th>Command (COBOL)</th>
<th>LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER</td>
<td>LIST</td>
<td>BREAK</td>
<td>LIST</td>
</tr>
<tr>
<td>ALTER</td>
<td>LIST</td>
<td>BREAK</td>
<td>LIST</td>
</tr>
<tr>
<td>ALTER</td>
<td>LIST</td>
<td>BREAK</td>
<td>LIST</td>
</tr>
<tr>
<td>ALTER</td>
<td>LIST</td>
<td>BREAK</td>
<td>LIST</td>
</tr>
<tr>
<td>ALTER</td>
<td>LIST</td>
<td>BREAK</td>
<td>LIST</td>
</tr>
</tbody>
</table>

2 The target must be session variable.

The following commands are not available while you replay recorded statements:

<table>
<thead>
<tr>
<th>Command (PL/I)</th>
<th>LIST</th>
<th>Command (PL/I)</th>
<th>LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT</td>
<td>LIST</td>
<td>RUN</td>
<td>LIST</td>
</tr>
<tr>
<td>AT</td>
<td>LIST</td>
<td>RUN</td>
<td>LIST</td>
</tr>
<tr>
<td>AT</td>
<td>LIST</td>
<td>RUN</td>
<td>LIST</td>
</tr>
<tr>
<td>AT</td>
<td>LIST</td>
<td>RUN</td>
<td>LIST</td>
</tr>
<tr>
<td>AT</td>
<td>LIST</td>
<td>RUN</td>
<td>LIST</td>
</tr>
</tbody>
</table>

186 Debug Tool V10 Reference and Messages
CALL %DUMP command  “DO command (PL/I)” on page 109
CALL entry_name command ENABLE command SELECT command (PL/I)
(COBOL) SET INTERCEPT
CLEAR AT Expression command (C and C++) switch command (C and C++)
CLEAR DECLARE for command (C and C++) TRIGGER command
CLEAR ON GO command while command (C and C++)
CLEAR VARIABLES GOTO command

PLAYBACK FORWARD command
The PLAYBACK FORWARD command informs Debug Tool to perform STEP and RUNTO commands forward, starting from the current statement and going to the next statement.

PLAYBACK BACKWARD command
The PLAYBACK BACKWARD command informs Debug Tool to perform STEP and RUNTO commands backward, starting from the current statement and going to previous statements. Backward is the initial direction when you enter the PLAYBACK START command.

PLAYBACK STOP command
The PLAYBACK STOP command resumes normal debugging at the statement where you entered the PLAYBACK START command. All suspended breakpoints are enabled and all commands are available. Debug Tool continues to record the statements you run and, if you specified the DATA option, information about your program.

PLAYBACK DISABLE command
The PLAYBACK DISABLE command informs Debug Tool to stop recording the statements that you run and, if you specified the DATA option, information about your program. The information about the program that Debug Tool collected while recording is discarded. You can instruct Debug Tool to stop recording for one or more compile units. If you stop recording for one compile unit and continue recording for other compile units, the information that you collected for the one compile unit is discarded.
cuname
Indicates to Debug Tool to stop recording for the compile unit or compile units specified. Only the names of currently known compile units can be specified.
*
Indicates to Debug Tool to stop recording for all compile units. This is the default.

**POPUP command**

Displays the Command pop-up window, where you can type in multiline commands.

```plaintext
POPUP
```

**integer**
The number of lines for the window.

If you do not specify an integer, Debug Tool opens the window with the number of lines specified by the SET POPUP command.

**Related references**
“SET POPUP command” on page 242

**POSITION command**

Positions the cursor to a specific line in the specified window. This command does not work in the disassembly view.

```plaintext
POSITION
```

**integer**
Specifies that Debug Tool scroll the specified window to line number `integer`. Debug Tool matches `integer` to the line number in the prefix area of the specified window. Debug Tool 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 6. Source window prefix commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;AT Prefix command (full-screen mode)&quot; on page 72</td>
<td>Defines a statement breakpoint through the Source window prefix area.</td>
</tr>
<tr>
<td>&quot;CLEAR prefix (full-screen mode)&quot; on page 92</td>
<td>Clears a breakpoint through the Source window prefix area.</td>
</tr>
<tr>
<td>&quot;DISABLE prefix (full-screen mode)&quot; on page 108</td>
<td>Disables a breakpoint through the Source window prefix area.</td>
</tr>
<tr>
<td>&quot;ENABLE prefix (full-screen mode)&quot; on page 113</td>
<td>Enables a disabled breakpoint through the Source window prefix area.</td>
</tr>
<tr>
<td>&quot;L prefix command (full-screen mode)&quot; on page 151</td>
<td>Displays the values of the variables on that line.</td>
</tr>
<tr>
<td>&quot;M prefix (full-screen mode)&quot; on page 169</td>
<td>Adds the variables on that line to the Monitor window.</td>
</tr>
<tr>
<td>&quot;QUERY prefix (full-screen mode)&quot; on page 195</td>
<td>Queries what statements have breakpoints through the Source window prefix area.</td>
</tr>
<tr>
<td>&quot;RUNTO prefix command (full-screen mode)&quot; on page 201</td>
<td>Runs the program to the location that the cursor or statement identifier indicate in the Source window prefix area.</td>
</tr>
<tr>
<td>&quot;SHOW prefix command (full-screen mode)&quot; on page 263</td>
<td>Specifies what relative statement or verb within the line is to have its frequency count shown in the suffix area.</td>
</tr>
</tbody>
</table>

## 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 PROCEDURE ; command END; name
```

- **name**
  A valid Debug Tool procedure name. It must be a valid identifier in the current programming language. The maximum length is 31 characters.

- **command**
  A valid Debug Tool command other than a declaration or PANEL command.
Usage notes

- Because the Debug Tool procedure names are always uppercase, the procedure names are converted to uppercase even for programming languages that have mixed-case symbols.
- If a GO or STEP command is issued within a procedure or a nested procedure, any statements following the GO or STEP in that procedure and the containing procedure are ignored. If control returns to Debug Tool, it returns to the statement following the CALL of the containing PROCEDURE.
- It is recommended that procedure names be chosen so that they are valid for all possible programming language settings throughout the entire Debug Tool debug session.

Examples

- When procedure proc1 is called, the values of variables x, y, and z are displayed.
  proc1: PROCEDURE; LIST (x, y, z); END;
- Define a procedure named setat34 that sets a breakpoint at statement 34. Procedure setat34 contains a nested procedure lister that lists current statement breakpoints. Procedure lister can be called only from within setat34.
  setat34: PROCEDURE;
    AT 34;
    lister: PROCEDURE;
      LIST AT STATEMENT;
      END;
    CALL lister;
  END;

QUALIFY RESET command

The QUALIFY RESET command is equivalent to the SET QUALIFY RESET command.

QUERY command

The QUERY command displays the current value of the specified Debug Tool setting, the current setting of all the Debug Tool settings, or the current location in the suspended program.

For an explanation of the Debug Tool settings, see the SET command.

Attributes A through I:

Attributes J through P:

Attributes Q through Z:
Attributes J through P:

- KEYS
- LDD
- LIST—TABULAR
- LOCATION
- LOG
- LOG—NUMBERS
- LONGCUNAME
- MDBG
- MONITOR
  - COLUMN
    - DATATYPE
    - LIMIT
    - NUMBERS
    - WRAP
- MSGID
  - NATIONAL
  - LANGUAGE
- PACE
- PFKEYS
- PLAYBACK
- PLAYBACK—LOCATION
- POPUP
- PROGRAMMING—LANGUAGE
- PROMPT
Attributes Q through Z:

- QUALIFY
- REFRESH
- RESTORE
- REWRITE
- SAVE
- SCREEN
- SCROLL—DISPLAY
- SEQUENCE
- SETS
- SOURCE
- SUFFIX
- TEST
- WARNING
- WINDOW—SIZES

Notes:
1. You can use this command in remote debug mode.
2. Available only if the Dynamic Debug facility is installed.
3. 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 DYNEDEBUG setting.

ECHO
Displays the current ECHO setting.

EQUATES
Displays the current EQUATE definitions.

EXECUTE
Displays the current EXECUTE setting.

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 TABULAR
Displays the current LIST TABULAR setting.

LOCATION
Displays the statement identifier where execution is suspended. The current statement identified by QUERY LOCATION has not yet executed. If suspended at a breakpoint, the description of the breakpoint is also displayed.

LOG
Displays the current LOG setting.

LOG NUMBERS (full-screen mode)
Displays the current LOG NUMBERS setting.

LONGCUNAME
Displays the current LONGCUNAME setting.

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. Debug Tool does not differentiate between C and C++, use this option for C++ as well a C programs.

PROMPT (full-screen mode)
Displays the current PROMPT setting.

QUALIFY
Displays the current QUALIFY BLOCK setting.

REFRESH (full-screen mode)
Displays the current REFRESH setting.

RESTORE
Displays the current RESTORE setting.

REWRITE
Displays the current REWRITE setting. This setting is not supported in batch mode.

SAVE
Displays the current SAVE setting.

SCREEN (full-screen mode)
Displays the current SCREEN setting.

SCROLL DISPLAY (full-screen mode)
Displays the current SCROLL DISPLAY setting.
SEQUENCE (PL/I)
Displays current SEQUENCE setting.

SETS
Displays all settings that are controlled by the SET command.

SOURCE
Displays the current SOURCE setting.

SUFFIX (full-screen mode)
Displays the current SUFFIX setting.

TEST
Displays the current TEST setting.

WARNING (C)
Displays the current WARNING setting.

WINDOW SIZES
Displays the current WINDOW SIZE values and WINDOW CLOSE information. The window sizes are the values that apply when all windows are open.

Usage note
You can use the QUERY ASSEMBLER, QUERY AUTOMONITOR, QUERY CURRENT VIEW, QUERY DEFAULT LISTINGS, QUERY DEFAULT VIEW, QUERY DISASSEMBLY, QUERY DYNDDEBUG, 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)”
Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

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 “+”.

Chapter 5. Debug Tool commands  195
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 141

QUIT command

The QUIT command ends a Debug Tool 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.

\[
\text{QUIT} \{(\text{expression})\} ;
\]

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

If expression is specified, this value is used as the application return code value.
The actual return code for the run is determined by the execution environment.

You cannot use expression in remote debug mode.

ABEND
If you specify ABEND, Debug Tool raises a CEE2F1 exception to terminate each
active enclave.

DEBUG
If you specify DEBUG, Debug Tool ends and your program keeps running. Any
calls to restart Debug Tool are ignored. By default, when running under CICS,
a pseudo-conversational application will run until the end of the conversation
(until EXEC CICS RETURN without TRANSID is issued to return to CICS).

TASK
TASK applies to CICS pseudo-conversational applications. If you specify TASK,
Debug Tool processing will be terminated until the end of the current CICS
pseudo-conversational task (EXEC CICS RETURN TRANSID). When a new
task is started in the pseudo-conversation, Debug Tool debugging will resume.

Usage notes
- Debug Tool will only resume in a new pseudo-conversational task if CADP or
  DTCN successfully match on a pattern.
- QUIT is always logged in a comment line except where it appears in a command
  list. This enables you to reuse the log file as a primary commands file.
- If QUIT is entered from a Debug Tool commands file, no prompt is displayed.
  This behavior applies to the Debug Tool preferences files, primary commands
  files, and USE files.
For PL/I, the expression will be converted to FIXED BINARY (31,0), if necessary. In addition, if an expression is specified, it is used as if your program called the PLIRETC built-in subroutine.

For PL/I, the value of the expression must be nonnegative and less than 1000.

If you enter the QUIT DEBUG command and then want to restart Debug Tool, you must first restart your program.

If you enter the QUIT or QQUIT command while you are debugging a non-Language Environment assembler or non-Language Environment COBOL program running under CICS, Debug Tool 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 Debug Tool session.
  
  QUIT;

- End a Debug Tool session and use the value in variable x as the application return code.
  
  QUIT (x);

- End a Debug Tool session without ending the program.
  
  QUIT DEBUG;

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

Related references

“expression” on page 14

Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

QQUIT command

The QQUIT command ends a Debug Tool session without further prompting.

```plaintext
QUIT;
```

Usage notes

- In full-screen mode, the QQUIT command does not display a prompt panel to verify that you want to quit the debug session.

- If you enter the QQUIT command while you are debugging a non-Language Environment assembler or non-Language Environment COBOL program running under CICS, Debug Tool 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 Debug Tool session.

QUIT;
RESTORE command

The RESTORE command enables you to explicitly restore the settings, breakpoints, and monitor specifications that were previously saved by the SET SAVE AUTO command when Debug Tool terminated.

```
/SM590000/SM590000

RESTORE SETTINGS

BPS

MONITORS

BPS—MONITORS

MONITORS—BPS
```

**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 Debug Tool 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**
- Debug Tool User’s Guide

**Related references**
- “SET RESTORE command” on page 247
- “SET SAVE command” on page 249

---

## RETRIEVE command (full-screen mode)

The RETRIEVE command displays the last command entered on the command line. For long commands this might be only the last line of the command.

```
RETRIEVE COMMAND;
```

**COMMAND**
Retrieves commands. Any command retrieved to the command line can be performed by pressing Enter. The retrieved command can also be modified before it is performed. Successive RETRIEVE commands continue to display up to 12 commands previously entered on the command line. This operand is most useful when assigned to a PF key.

**Usage notes**
- The RETRIEVE command is not logged.

**Example**
Retrieve the last line so that it can be reissued or modified.

```
RETRIEVE COMMAND;
```
RUN command

The RUN command is synonymous to the GO command.

Refer to the following 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.

```
RUNTO statement_id;
```

**statement_id**

A valid statement identifier. If you are debugging a disassembled program, specify the statement identifier as an offset in hexadecimal form (X'offset').

**Usage notes**

- If you indicate a statement by positioning the cursor on the statement, the cursor must be in the Source window and positioned on a line where an executable statement begins.
- If you indicate a statement by positioning the cursor on the statement and there are multiple statements on the same line, the target of the RUNTO command is the first relative statement on the line. For optimized COBOL programs, the target of the command is the first executable command which was not discarded by the optimizer.
- If you indicate a statement by providing a statement id, the statement id must be an executable statement.
- Execution continues until one of the following conditions occurs:
  - The location indicated by the cursor position or the statement id is reached.
  - A previously set breakpoint is encountered.
  - The end of the job is reached.
- For optimized COBOL programs, the RUNTO command remains in effect until the statement you indicated is reached. For example, if your program encounters a breakpoint and then you enter the GO or RUN command, the program runs until the next breakpoint is encountered or the statement you indicated is reached.
- You can use the RUNTO command in remote debug mode only by entering it in 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 IPL111A, where IPL111A is known in the current enclave.
    
    ```
    RUNTO IPL111A ::> 11
    ```

- Run to statement 36, where statement 36 is located in the Source window.
1. Type RUNTO in the command line.
2. Place the cursor on statement 36.
3. Press Enter.

- Run to the statement 74, using a PF key.
  1. Define a PF key to run to the cursor position.
     SET PF13 = RUNTO;
  2. Place the cursor at the statement 74 and hit shift+PF1 key.

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

**Related references**

“RUN command” on page 200

### RUNTO prefix command (full-screen mode)

Runs to the statement when you issue this command through the Source window prefix area.

**Usage notes**

- For RUNTO prefix, no space is needed as a delimiter between the keyword and the integer; RUNTO 67 is equivalent to RUNTO67.
- For optimized COBOL programs, if there are multiple statements on a line, the RUNTO prefix runs to the first executable statement which was not discarded by the optimizer.

**Example**

Run to the statement 67, where statement 67 is located in the Source window.

- Type RUNTO in the prefix area of statement 67, then press Enter.

### SCROLL command (full-screen mode)

The SCROLL command provides horizontal and vertical scrolling in full-screen mode. Scroll commands can be made immediately effective with the IMMEDIATE command. The SCROLL keyword is optional.

The Log, Monitor, Memory, or Source window will not wrap around when scrolled.

```
DOWN
-LEFT
-NEXT
-RIGHT
-UP
-BOTTOM
-TO-integer
-TOP
```

**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 Debug Tool session panel. Default scroll and default
window settings are controlled by SET DEFAULT SCROLL and SET DEFAULT WINDOW
commands.
- When the SCROLL field on the Debug Tool session panel is typed over with a
new value, the equivalent SET DEFAULT SCROLL command is issued just as if you
had typed the command into the command line (that is, it is logged and
retrievable).
- The SCROLL command is not logged.
- To scroll the monitor value area left or right, SET MONITOR WRAP OFF must be in
effect and the cursor must be in the monitor value area.

Examples
- Scroll one page down in the window containing the cursor.
  SCROLL DOWN PAGE CURSOR;
- Scroll the monitor window 12 columns to the left.
  SCROLL LEFT 12 MONITOR;
- Scroll the monitor value window 15 columns to the right.
  SET MONITOR WRAP OFF;SCROLL RIGHT 15;

(Do not press Enter.) Place cursor in the monitor value area. Press Enter.
- Scroll the Source window to a line breakpoint.
  LIST AT STATEMENT;
  The STATEMENT COB019 ::> COB01A9 :> 56.1 breakpoint action is:
  ;
  SCROLL TO 56;
SELECT command (PL/I)

The SELECT command chooses one of a set of alternate commands.

If the reference can be satisfied by more than one of the WHEN clauses, only the first one is performed. If there is no reference, the first WHEN clause containing an expression that is true is executed. If none of the WHEN clauses are satisfied, the command specified on the OTHERWISE clause, if present, is performed. If the OTHERWISE clause should be executed and it is not present, a Debug Tool message is issued.

```
SELECT (reference) ;
WHEN (expression) command
otherwise command END ;
```

**reference**
A valid Debug Tool 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 Debug Tool PL/I expression.

**command**
A valid Debug Tool 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 Debug Tool 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 Debug Tool. Except where otherwise specified, settings remain in effect for the entire debug session.

The following table summarizes the forms of the SET command.

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET ASSEMBLER ON/OFF command</td>
<td>Controls the enablement of assembler debugging.</td>
</tr>
<tr>
<td>SET ASSEMBLER STEPOVER command</td>
<td>Controls the behavior of the STEP OVER command while debugging assembler compile units.</td>
</tr>
<tr>
<td>SET AUTOMONITOR command</td>
<td>Controls the addition of data items to the Monitor window.</td>
</tr>
<tr>
<td>SET CHANGE command</td>
<td>Controls the frequency of checking the AT CHANGE breakpoints.</td>
</tr>
<tr>
<td>SET COLOR command (full-screen and line mode)</td>
<td>Provides control of the color, highlighting, and intensity attributes.</td>
</tr>
<tr>
<td>SET COUNTRY command</td>
<td>Changes the current national country setting.</td>
</tr>
<tr>
<td>SET DBCS command</td>
<td>Controls whether DBCS shift-in and shift-out codes are recognized.</td>
</tr>
<tr>
<td>SET DEFAULT DBG command</td>
<td>Defines a default partitioned data set DD name or DS name that Debug Tool searches through to locate the .dbg files.</td>
</tr>
<tr>
<td>SET DEFAULT LISTINGS command</td>
<td>Defines a default partitioned data set (PDS) ddname or dname searched for program source listings or source files.</td>
</tr>
<tr>
<td>SET DEFAULT MDBG command</td>
<td>Defines a default partitioned data set DD name or DS name that Debug Tool searches through to locate the .mdbg files.</td>
</tr>
<tr>
<td>SET DEFAULT SCROLL command (full-screen mode)</td>
<td>Sets the default scroll amount.</td>
</tr>
<tr>
<td>SET DEFAULT VIEW command</td>
<td>Controls the default view for assembler compile units.</td>
</tr>
<tr>
<td>SET DEFAULT WINDOW command (full-screen mode)</td>
<td>Sets the window that is affected by a window referencing command.</td>
</tr>
<tr>
<td>SET DISASSEMBLY command</td>
<td>Controls whether the disassembly view is displayed in the Source window.</td>
</tr>
<tr>
<td>SET DYNDEBUG command</td>
<td>Controls whether the Dynamic Debug facility is activated.</td>
</tr>
<tr>
<td>SET ECHO command</td>
<td>Controls whether 60 and STEP commands are recorded in the log window.</td>
</tr>
<tr>
<td>SET EQUATE command</td>
<td>Equates a symbol to a string of characters.</td>
</tr>
<tr>
<td>SET EXECUTE command</td>
<td>Controls whether commands are performed or just syntax checked.</td>
</tr>
<tr>
<td>SET FIND BOUNDS command</td>
<td>Controls the columns searched in the Source window and in line mode.</td>
</tr>
<tr>
<td>SET FREQUENCY command</td>
<td>Controls whether statement executions are counted.</td>
</tr>
<tr>
<td>SET HISTORY command</td>
<td>Specifies whether entries to Debug Tool are recorded in the history table.</td>
</tr>
<tr>
<td>Command Description</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td></td>
</tr>
<tr>
<td>SET IGNORELINK command</td>
<td>Specifies whether to ignore any new LINK level (nested enclave).</td>
</tr>
<tr>
<td>SET INTERCEPT command (C and C++)</td>
<td>Intercepts input to and output from specified files. Output and prompts for input are displayed in the log.</td>
</tr>
<tr>
<td>SET INTERCEPT command (COBOL, full-screen mode, line mode, batch mode)</td>
<td>Intercepts input to and output from the CONSOLE. Output and prompts for input are displayed in the log.</td>
</tr>
<tr>
<td>SET INTERCEPT command (COBOL, remote debug mode)</td>
<td>Intercepts output from COBOL DISPLAY statements. Output is displayed in the Debug Console.</td>
</tr>
<tr>
<td>SET KEYS command (full-screen mode)</td>
<td>Controls whether PF key definitions are displayed.</td>
</tr>
<tr>
<td>SET LDD command</td>
<td>Controls how debug data is loaded for assemblies containing multiple CSECTs.</td>
</tr>
<tr>
<td>SET LIST TABULAR command</td>
<td>Controls the formatting of the output of the LIST command.</td>
</tr>
<tr>
<td>SET LOG command</td>
<td>Controls the logging of output and assignment to the log file.</td>
</tr>
<tr>
<td>SET LOG NUMBERS command (full-screen mode)</td>
<td>Controls whether line numbers are shown in the log window.</td>
</tr>
<tr>
<td>SET LONGCUNAME command</td>
<td>Controls whether a long or a short CU name is shown.</td>
</tr>
<tr>
<td>SET MDBG command</td>
<td>Associates a .mdbg files to one load module or DLL.</td>
</tr>
<tr>
<td>SET MONITOR command</td>
<td>Controls the format and layout of variable names and values displayed in the Monitor window.</td>
</tr>
<tr>
<td>SET MSGID command</td>
<td>Controls whether message identifiers are shown.</td>
</tr>
<tr>
<td>SET NATIONAL LANGUAGE command</td>
<td>Switches your application to a different run-time national language.</td>
</tr>
<tr>
<td>SET PACE command</td>
<td>Specifies the maximum pace of animated execution.</td>
</tr>
<tr>
<td>SET PFKEY command</td>
<td>Associates a Debug Tool command with a PF key.</td>
</tr>
<tr>
<td>SET POPUP command</td>
<td>Controls the number of lines displayed in the Command pop-up window.</td>
</tr>
<tr>
<td>SET PROGRAMMING LANGUAGE command</td>
<td>Sets the current programming language.</td>
</tr>
<tr>
<td>SET PROMPT command (full-screen mode)</td>
<td>Controls the display of the current program location.</td>
</tr>
<tr>
<td>SET QUALIFY command</td>
<td>Simplifies the identification of references and statement numbers by resetting the point of view.</td>
</tr>
<tr>
<td>SET REFRESH command (full-screen mode)</td>
<td>Controls screen refreshing when the SCREEN setting is ON.</td>
</tr>
<tr>
<td>SET RESTORE command</td>
<td>Controls the automatic and manual restoring of settings, breakpoints, and monitor specifications.</td>
</tr>
<tr>
<td>SET REWRITE command (full-screen mode)</td>
<td>Forces a periodic screen rewrite.</td>
</tr>
<tr>
<td>SET SAVE command</td>
<td>Controls the automatic saving of settings, breakpoints, and monitor specifications.</td>
</tr>
<tr>
<td>SET SCREEN command (full-screen mode)</td>
<td>Controls how information is displayed on the screen.</td>
</tr>
</tbody>
</table>
"SET SCROLL DISPLAY command (full-screen mode)" on page 253
Controls whether the scroll field is displayed.

"SET SEQUENCE command (PL/I)" on page 253
Controls whether Debug Tool interprets data after column 72 as a sequence number.

"SET SOURCE command" on page 254
Associates a source listing or source file with one or more compile units.

"SET SUFFIX command (full-screen mode)" on page 256
Controls the display of the Source window suffix area.

"SET TEST command" on page 257
Overrides the initial TEST run-time options specified at invocation.

"SET WARNING command (C, C++, COBOL, and PL/I)" on page 258
Controls display of the Debug Tool 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 260

---

**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

```
SET ASSEMBLER ON
```

ON

Disables the display of data that is useful while you debug an assembler program.

```
SET ASSEMBLER ;
```

OFF

Enables the display of data that is useful while you debug an assembler program.

```
SET ASSEMBLER OFF
```

**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 221
Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

SET ASSEMBLER STEPOVER command

Specifies how Debug Tool processes STEP OVER commands in assembler compile units. When EXTONLY is in effect, Debug Tool only steps over calls to external subroutines. When EXTINT is in effect, Debug Tool 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.

Debug Tool 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

Debug Tool 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)

Debug Tool assumes that you use one of the following instructions to call internal subroutines:
• BAL
• BAS
• BRAS
• BALR
• BASR
• BASSM
• BRASL
**EXTONLY**

Specifies that Debug Tool steps over external subroutines and steps through internal subroutines.

**EXTINT**

Specifies that Debug Tool 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:
  - Debug Tool might not regain control
  - Debug Tool might regain control only when another breakpoint is run
  - Debug Tool might regain control only when an external event occurs
  - Debug Tool 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 207
- “STEP command” on page 263
- Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

**SET AUTOMONITOR command**

Controls the monitoring of data items for the statement that Debug Tool will run next, the most recent statement that Debug Tool 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, do not need the SYM suboption of the TEST compiler option.
- assembler, disassembly, or non-Language Environment COBOL compile units

The SET AUTOMONITOR command does not work for compile units written in any other language. In addition, the compile unit must be compiled or assembled with one of the following compilers or assemblers:

- Enterprise COBOL for z/OS, 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, 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
**ON** Enables monitoring of data items for the statement that Debug Tool will run next, the most recent statement that Debug Tool 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 Debug Tool will run next. This is the default.

**PREVIOUS** Monitor data items on the most recent statement that Debug Tool ran.

**BOTH** Monitor data items for the statement that Debug Tool will run next and the most recent statement that Debug Tool ran.

**Usage notes**

- 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 Debug Tool writes the breakpoint location into the log. This provides a record of the breakpoints encountered (breakpoint trace). No variable information is displayed.
- To record the breakpoints encountered (breakpoint trace) in the log file, enter the following commands: **SET AUTOMONITOR ON LOG; AT * GO;**. For compile units compiled with a compiler that supports 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, Debug Tool displays only 32-bit general registers, floating-point registers, and storage operands. Debug Tool 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 non-Language Environment COBOL, array references are not included in the AUTOMONITOR output.

- 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.

- 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, Debug Tool displays the value of a variable on the line that Debug Tool 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, Debug Tool cannot determine the data from the previous enclave:
  - SET AUTOMONITOR ON LOG with PREVIOUS or BOTH
  - SET AUTOMONITOR ON NOLOG with PREVIOUS or BOTH

Debug Tool displays a message.

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

**Related references**

Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

### SET CHANGE command

Controls the frequency of checking the AT CHANGE breakpoints. The initial setting is STATEMENT/LINE.

![SET CHANGE diagram]

- **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.

**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.
  ```plaintext
  SET CHANGE;
  ```
- Specify that AT CHANGE breakpoints are checked at all path points.
  ```plaintext
  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.

```plaintext
SET COLOR color_attributes UI_elements ;
```

**color_attributes:**

- CYCLE
- BLUE
- GREEN
- PINK
- RED
- TURQUOISE
- WHITE
- YELLOW

**UI_elements:**

- BLINK
- NONE
- REVERSE
- UNDERLINE
- HIGH
- LOW

---

212  Debug Tool V10 Reference and Messages
**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 Debug Tool input displayed in the log window.

TEST OUTPUT
Selects the Debug Tool output displayed in the log window.

TITLE FIELDS
Selects the information fields in the top line of the screen, such as current programming language setting or the current location within the program.

TITLE HEADERS
Selects the descriptive headers in the top line of the screen, such as location.

TOFEOF MARKER
Selects the top-of-file and end-of-file lines in the session panel windows.

WINDOW HEADERS
Selects the header lines for the windows in the main session panel.

Examples
- Set the Source window display area to yellow reverse video.
  SET COLOR YELLOW REVERSE SOURCE AREA;
- Set the monitor window display area to high intensity green.
SET 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.

```
SET COUNTRY country_code;
```

`country_code`

A valid two-letter set that identifies the country code used. The country code can have one of the following values:

- **United States:** US
- **Japanese:** JP

Country codes cannot be truncated.

Usage notes

- This setting affects both your application and Debug Tool.
- At the beginning of an enclave, the settings are those provided by Language Environment, your operating system, or the Debug Tool run-time options. For nested enclaves, the parent’s settings are restored upon return from a child enclave.

Example

Change the current country code to correspond to Japan.

```
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.

```
SET DBCS ON OFF;
```

**ON**

Interprets shift-in and shift-out codes. If you debugging in full-screen mode and your terminal is not capable of displaying DBCS characters, this option is not available.

**OFF**

Ignores shift-in and shift-out codes.

Usage notes

- If you enter the commands SET NATIONAL LANGUAGE ENU and then SET DBCS ON, Debug Tool resets the national language to UEN to remain compatible with DBCS characters.
- If NATIONAL LANGUAGE is set to JPN or KOR and you are using full-screen mode, enter the SET DBCS ON command so that Debug Tool displays messages correctly.

Example

Chapter 5. Debug Tool commands 215
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 239

SET DEFAULT DBG command

Defines a default partitioned data set DD name or DS name that Debug Tool 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.

```
SET DEFAULT DBG
```

**ddname**

Specifies a valid z/OS DD name. If the operand is less than nine characters long and does not contain a period, Debug Tool 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**, Debug Tool 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 can not 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**

- **Debug Tool User’s Guide**
  - “SET SOURCE command” on page 254
  - “SET DEFAULT MDBG command” on page 218
  - “SET MDBG command” on page 236
  - Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393
  - “Specifying the location of source, listing, or separate debug file in remote debug mode by using environment variables” on page 395
  - “How does Debug Tool locate source, listing, or separate debug files?” in Debug Tool 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.

```
SET DEFAULT LISTINGS ddname dsn ...
```

- **ddname**
  - Specifies a valid z/OS DD name. If the operand is less than nine characters long and does not contain a period, it is interpreted as a DD name.
  - The *ddname* form can not be used if the data set allocated to it is C, C++ or Enterprise PL/I source and the EQAOPTS SUBSYS=ssss option is being used to access a source file in a library system.

- **dsn**
  - Specifies a valid, 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`, Debug Tool looks for a member named `AVER` in the PDS `myid.source.listing`. Because the member is called `AVERLIST`, the listing is not found. To specify the location, enter the following command:
  
  ```
  SET SOURCE ON (AVER) myid.source.listing(AVERLIST);
  ```

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

**Related tasks**

- [Debug Tool User's Guide](#)
  - "SET SOURCE command" on page 254
- [Appendix B, "Debug Tool commands supported in remote debug mode," on page 393](#)

**SET DEFAULT MDBG command**

Defines a default partitioned data set DD name or DS name that Debug Tool 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, Debug Tool 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 must set the EQAXOPT MDBG option to YES in the EQAOPTS options file, as described in Debug Tool Customization Guide. 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, Debug Tool 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 can not 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 254
- “SET DEFAULT DBG command” on page 216
- “SET MDBG command” on page 236
- Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393
- “Specifying the location of source, listing, or separate debug file in remote debug mode by using environment variables” on page 395
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.

```
SET DEFAULT SCROLL CSR ;
```

**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.

```
SET DEFAULT VIEW STANDARD ;
```

**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 B, “Debug Tool commands supported in remote debug mode,” on page 393

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.

```
SET DEFAULT WINDOW LOG;
```

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:

Chapter 5. Debug Tool commands
- DESCRIBE CUS
- LIST
- LIST NAMES CUS
- QUERY SOURCE

**SET DISASSEMBLY**

**ON**
Specifies that the disassembly view is displayed in the Source window.

**OFF**
Turns off the disassembly view. This is the default setting.

**Usage notes**

- The disassembly view is provided only for disassembled programs or programs written in supported languages that do not have debug information.
- You can use this command in remote debug mode.

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

**Related references**

- Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

**SET DYNDEBUG command**

Controls the activation or deactivation of the Dynamic Debug facility.

The Dynamic Debug facility must be installed and activated in order to debug the following types of programs:

- COBOL programs compiled with the `NONE` or `NOHOOK` suboptions of 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)
- non-Language Environment 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.

If the Dynamic Debug facility has been installed, the initial setting is **ON**. If it was not installed, the initial setting is **OFF** and you cannot activate the Dynamic Debug facility.

---

1. In non-CICS environments, SVC screening must be enabled to debug non-Language Environment COBOL programs, programs that run without the Language Environment runtime, or programs that are loaded by using the MVS LOAD and LINK macros. See [Debug Tool Customization Guide](#) for instructions on how to manage SVC screening.
ON Activates the Dynamic Debug facility.
OFF Deactivates the Dynamic Debug facility.

Usage notes
• After a dynamic debug hook has been inserted, either explicitly or implicitly, into any program during a debugging session, you cannot use the SET DYNDEBUG OFF command.
• You can use this command in remote debug mode.
• You can debug COBOL programs compiled with the NOHOOK suboption of the TEST compiler option of Enterprise COBOL for z/OS, Version 4.1, with the Dynamic Debug facility.
• 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
• For COBOL programs, you can use the GOTO or JUMPTO commands in the following situations:
  – A COBOL program compiled with hooks inserted by the compiler. If you are using Enterprise COBOL for z/OS, Version 4.1, 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 the 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 4.1, 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 4.1, and specify the EJP0D and NOHOOK suboption of the TEST compiler option. Specifying the EJP0D suboption might cause some loss of optimization.
• The Dynamic Debug facility does not support attention interrupts with 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 4.1
- 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 Debug Tool uses instead of the compiled-in hooks. This can improve the performance of the program while running under the control of Debug Tool.
  - Any COBOL compiler supported by Debug Tool
  - Any C/C++ compiler supported by Debug Tool
  - Any PL/I compiler supported by Debug Tool

- Refer to your system administrator to determine if the Dynamic Debug facility is installed on your system.

- The same program compiled with different TEST options may halt execution at different locations or the same scenarios. For instance, if you compile a program with TEST(ALL,...) and step through the first three lines, execution is halted on line four. However, if you compile the same program with TEST(NONE,SYM,...) and step through the first three lines, execution is halted on line five. The difference is due to optimization techniques used by the compiler.

A small arrowhead indicates where a Debug Tool would stop if the same program were compiled in two different ways.

<table>
<thead>
<tr>
<th>Program compiled with TEST(ALL)</th>
<th>Program compiled with TEST(NONE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>000001 MOVE...</td>
<td>000001 MOVE...</td>
</tr>
<tr>
<td>000002 ADD...</td>
<td>000002 ADD...</td>
</tr>
<tr>
<td>► 000003 LABEL: ...</td>
<td>000003 LABEL: ...</td>
</tr>
<tr>
<td>000004 MOVE...</td>
<td>► 000004 MOVE...</td>
</tr>
</tbody>
</table>

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

**Related references**

Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

---

**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 Debug Tool command. When an equated symbol is found in a Debug Tool command (other than the identifier operand in SET EQUATE and CLEAR EQUATE), the equated symbol is replaced by the specified string before parsing continues.

```
SET EQUATE identifier = string;
```

identifier

An identifier that is valid in the current programming language. The maximum length of the identifier is:

- For C, 32 SBCS characters
- For COBOL and non-Language Environment COBOL, 30 SBCS characters
- For PL/I, 31 SBCS characters

The identifier can contain DBCS characters.

string

A string constant in the current programming language. The maximum length of the replacement string is 255 SBCS characters.

Usage notes

- Operands of the following commands are for environments other than the standard Debug Tool environment (that is, TSO DS name, and so forth) and are not scanned for EQUATE 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.

```
SET EXECUTE [ON|OFF];
```

**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 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.

```
SET FIND BOUNDS leftcolumn rightcolumn *;
```

**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 116
- “QUERY command” on page 190

SET FREQUENCY command

Controls whether statement executions are counted. The initial setting is OFF.

```
SET FREQUENCY [ON|OFF] cu_spec ;
```

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 14
- “LIST FREQUENCY command” on page 153
- “SET SUFFIX command (full-screen mode)” on page 256

SET HISTORY command

Specifies whether entries to Debug Tool are recorded in the history table and optionally adjusts the size of the table. The history table contains information about the most recently processed breakpoints and conditions. The initial setting is ON; the initial size is 100.

```
SET HISTORY ON [OFF [integer]]
```

**ON** Maintains the history of invocations.

**OFF** Suppresses the history of invocations.

**integer**

The number of entries kept in the history table.

Usage notes

• History is not collected for disassembly compile units.

Examples

• Adjust the history table size to 50 lines.

```
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 154

SET IGNORELINK command

Specifies that any new LINK level (nested enclave) is ignored while the setting is ON. Debug Tool 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. Debug Tool does not stop at programs in new enclaves.

OFF  Programs in new enclaves (links) are not ignored. Debug Tool 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 Debug Tool because at the time it was created, the setting of SET IGNORELINK was OFF. Otherwise, Debug Tool runs to the next breakpoint in a previous enclave that was not ignored by Debug Tool 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.
- Debug Tool does not stop for any deferred entry breakpoints for a compile unit in a new enclave when the SET IGNORELINK setting is ON.
- Debug Tool 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.

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

Related references
- “DISABLE command” on page 106
- “ENABLE command” on page 112
- “QUERY command” on page 190
- “STEP command” on page 263

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 \texttt{SET INTERCEPT OFF} command. The initial setting is \texttt{OFF}.

\texttt{SET INTERCEPT \texttt{FILE} file_spec \texttt{ON};}

\texttt{OFF}

\texttt{FILE file_spec}

A valid \texttt{fopen()} file specifier including \texttt{stdin}, \texttt{stdout}, or \texttt{stderr}. The \texttt{FILE} keyword cannot be abbreviated.

\textbf{Usage notes}

- Intercepted streams or files cannot be part of any C I/O redirection during the execution of a nested enclave.
- You cannot use the \texttt{SET INTERCEPT} command while you replay recorded statements by using the \texttt{PLAYBACK} commands.

\textbf{Examples}

Turn on the I/O interception for the \texttt{fopen()} file specifier \texttt{dd:mydd}. The current programming language setting is C.

\texttt{SET INTERCEPT \texttt{ON \texttt{FILE} dd:mydd};}

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

\textbf{Related references}

- "\texttt{INPUT command (C, C++, and COBOL)}" on page 137
- "\texttt{SET REFRESH command (full-screen mode)}" on page 247

\textbf{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 \texttt{SET INTERCEPT OFF} command. The initial setting is \texttt{OFF}.

\texttt{ON}

Turns on console I/O interception. Debug Tool displays output in the log, preceded by the \texttt{CONSOLE} keyword to identify the output. Input causes a prompt entry in the log, with the \texttt{CONSOLE} identified. You can then enter input for the console on the command line by using the \texttt{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 137
“SET REFRESH command (full-screen mode)” on page 247

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 249
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 241

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 non-Language Environment 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
Debug Tool User’s Guide

Related references
"LOADDEBUGDATA command” on page 164
Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

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 each performed command and the resulting output is written to the log file and defines (or redefines) the file that is used. The initial setting is ON FILE INSPLOG. This is a valid DD name in z/OS.

ON Specifies that commands and output are written to the log file.

FILE fileid
Identifies the log file used. The FILE keyword cannot be abbreviated.

In non-CICS, fileid is a DD name or a fully-qualified data set name. Partitioned data sets cannot be used.
In CICS, `fileid` is a fully-qualified data set name.

If `fileid` has the form of a DD name, Debug Tool checks to see if the file is allocated.

In full-screen mode, the log file should not be allocated to the 3270 terminal device.

**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 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.
- In CICS, the log file (INSPL0G) is not automatically started. You need to use the SET LOG ON command.
- Ensure that you allocate a log file big enough to hold all the log output from a debug session, because the log file is truncated after it becomes full. (A warning message is not issued before the log is truncated.)
- 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.
SET LOG NUMBERS command (full-screen mode)

Controls whether line numbers are shown in the log window. The initial setting is ON.

```
SET LOG NUMBERS [ON|OFF] ;
```

ON  Shows line numbers in the log window.
OFF Suppresses line numbers in the log window.

Example

Specify that log line numbers are not shown.

```
SET LOG NUMBERS OFF;
```

SET LONGCUNAME command

Controls whether a short or long CU name is displayed.

```
SET LONGCUNAME [ON|OFF] ;
```

ON  Specifies that a long CU name is displayed.
OFF Specifies that a short CU name is displayed. The short CU name is displayed in the session panel header, source window header area, and the Source Identification Panel.

Usage notes

- You can enter the SET LONGCUNAME at any time, but it applies only to C, C++, and Enterprise PL/I programs. If you compiled your program with 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
- A HFS path and file name

These forms can result in long CU names that are truncated in the session panel header, which makes it difficult for you to identify the CU.

For these forms of compile unit names, Debug Tool displays short names in one of the following manners:
- For PDS file names, the short name is only the member name
- For sequential file names, the short name is the lowest level qualifier (name segment)
- For HFS file names, the short name is the file name, without path name

- Debug Tool commands affected by the LONGCUNAME setting: QUERY LOCATION, SET SOURCE, and AT ENTRY. All the other commands continue to require the long form of the CU name. For example, if you use the short name with the AT command (AT ARRAY3 ::> 'ARRAY3' ::> 10), Debug Tool displays an error message and does not set the breakpoint. However, if you enter the command AT ENTRY ARRAY3 ::> 'ARRAY3' ::>ARRAY3, Debug Tool sets the breakpoint or defers setting the breakpoint until the entry point is known to Debug Tool.
- You cannot use the SET LONGCUNAME command in remote debug mode.

**Examples**

- If the CU name is SMITH.TEST.SRC(ARRAY3), the short name is ARRAY3.
- If the CU name is SMITH.TEST.SOURCE.ABCD, the short name is ABCD.
- If the CU name is /testenvir/applications/cicsprograms/project1/prog2.cpp, the short name is prog2.cpp.

**SET MDBG command**

Associates a .mdbg file to one load module or DLL.

```
SET MDBG (lm_spec) fileid;
```

- `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 path and file name.

  In CICS, `fileid` is a fully-qualified data set name or an HFS path and file name.

  If `fileid` is less than nine characters in length and does not contain a period, Debug Tool assumes it is a DD name. Debug Tool checks to see if it is allocated. If it is not allocated, then Debug Tool assumes `fileid` is a data set name.

**Usage notes**

- Before you can use this command, you must set the EQAXOPT MDBG option to YES in the EQAOPTS options file, as described in [Debug Tool Customization Guide](#). 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 Release 11, or later.

Debug Tool 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 Debug Tool 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, Debug Tool immediately searches for the .mdbg file specified in fileid. If Debug Tool can not 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 Debug Tool searches for the .mdbg file. If Debug Tool can not 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 Debug Tool encounters that breakpoint, then it searches for the .mdbg file. If Debug Tool can not 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. Debug Tool 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 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 tasks

“SET SOURCE command” on page 254

“SET DEFAULT DBG command” on page 216

“SET DEFAULT MDBG command” on page 218

Appendix B, “Debug Tool commands supported in remote debug mode,” on page 393

“Specifying the location of source, listing, or separate debug file in remote debug mode by using environment variables” on page 395

“How does Debug Tool locate source, listing, or separate debug files?” in Debug Tool User’s Guide

“Specifying whether Debug Tool searches for .mdbg files” in Debug Tool Customization Guide

**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 Debug Tool displays in the Monitor window. The default value for integer is 1000. If you specify a new value, it must be greater than 1000.

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 Debug Tool can only display values in one scrollable line when the setting of MONITOR COLUMN is ON. You must first enter the SET MONITOR COLUMN ON command.

If you enter the SET MONITOR COLUMN OFF command while the SET MONITOR WRAP switch is set to OFF, the command is rejected. The Monitor window must be in columnar format to be able to display values in one scrollable line. You must first enter the SET MONITOR WRAP ON command.

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 Debug Tool messages are displayed with the message prefix identifiers. The initial setting is OFF.

```
SET MSGID [ON | 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 Debug Tool activity only. The initial setting is supplied by Language Environment or the NATLANG Debug Tool run-time option, according to the setting in the current enclave.

```
SET NATIONAL LANGUAGE language_code;
```

**language_code**
A valid three-letter set that identifies the language used or (for compatibility)
one of the two-letter language codes that was accepted in the previous release of INSPECT for C/370™ and PL/I. The language code can have one of the following values:

- United States English: ENU
- United States English (Uppercase): UEN
- Japanese: JPN
- Korean: KOR

If you enter the SET DBCS ON command and then you set the national language to ENU, Debug Tool resets the national language to UEN to remain compatible with DBCS characters.

For compatibility with the previous release of INSPECT for C/370 and PL/I:

- EN or ENGLISH is mapped to ENU
- UE or UENGLISH is mapped to UEN
- JA, JAPANESE, NI, or NIHONGO is mapped to JPN

Usage notes

- In order to display DBCS characters correctly in full-screen mode, the high order bit of the Language field in the VTAM Attribute Byte that must be set ON. To verify that this bit is set ON:
  1. In ISPF, select option 0 (Settings).
  2. On the command line, enter: `environ`.
  3. Tab to the section Terminal Status (TERMSTAT). In the Enable field, enter 2 (Query terminal information).
  4. Several pages of statistics appear. In the section GTTERM Information, note the value of the highest bit in the second byte of the field Attribute Byte. The value of this bit must be 1 (ON). For example, if the value of the Attribute Byte field is `x'008000C9'`, then DBCS characters display correctly because the second byte is `X'80'`. However, if the value of the Attribute Byte field is `x'000000C9'`, DBCS characters are not displayed properly. Contact the VTAM System Administrator to set the high order bit of the Language Field of the VTAM Attribute Byte to 1 (ON).

- The language you select by using the SET NATIONAL LANGUAGE command affects both your application and Debug Tool.

- At the beginning of an enclave, the settings are those provided by Language Environment, your operating system, or the NATLANG Debug Tool run-time option. For nested enclaves, the parent’s settings are restored upon return from a child enclave.

- If NATIONAL LANGUAGE is set to JPN or KOR and you are using full-screen mode, enter the SET DBCS ON command so that Debug Tool 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 215
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.

```
SET PACE number;
```

**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 Debug Tool User’s Guide for information about requesting an attention interrupt during interactive sessions.
- Associated with the SET PACE command is the STEP command. Animated execution is achieved by defining a PACE and then issuing a STEP n command where n is the number of steps to be seen in animated mode. STEP * can be used to see all steps to the next breakpoint in animated mode.
- When PACE is set to 0, no animation occurs.

**Example**

Set the animated execution pace to 1.5 steps per second.

```
SET PACE 1.5;
```

SET PFKEY command

Associates a Debug Tool command with a Program Function key (PF key). This setting is not supported in batch mode.

```
SET PFn string = command;
```

**PFN**

A valid program function key specification (PF1 - PF24).

**string**

The label shown in the PF key display (if the KEYS setting is ON) that is entered as a string constant. The string is truncated if longer than eight characters. If the string is omitted, the first eight characters of the command are displayed. For C and C++, the string must be surrounded by quotation marks ("). For COBOL, PL/I, assembler, and disassembly, the string can be surrounded by either quotation marks ("), or apostrophes (').

**command**

A valid Debug Tool command or partial command.

**Usage notes**

- If you specify the ? as the command, the ? is understood to be the command, not a request for syntax help.
- In Debug Tool, if there is any text on the command line at the time the PF key is pressed, that text is appended to the PF key string, with an intervening blank, for execution.
- The initial settings for PF keys 13-24 are equivalent to PF keys 1-12, respectively.
If you change the setting for a PF key in the 1–12 range, the equivalent key in the 13–24 range remains the same.

Example

Define the PF5 key to scroll the cursor-selected window forward.

- If the programming language setting is COBOL:
  ```
  SET PF5 "Down" = IMMEDIATE SCROLL DOWN;
  ```

- If the programming language setting is PL/I:
  ```
  SET PF5 'Down' = IMMEDIATE SCROLL DOWN;
  ```

- If the programming language setting is C++:
  ```
  SET PF5 "Down" = IMMEDIATE SCROLL DOWN;
  ```

In all cases, the setting for PF17 remains the same.

**SET 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.
- Debug Tool opens the Command pop-up window when you enter a continuation character or an incomplete command in the command line.

```
SET POPUP integer;
```

- integer
  The number of lines in the Command pop-up window when Debug Tool opens it. The initial default number of lines is 15.

Related references

“POPUP command” on page 188

**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 Debug Tool commands. The execution of a command is always consistent with the current programming language setting that was in effect when the command was parsed. The programming language setting at execution time is ignored.
CYCLE
Specifies that the programming language is set to the next language in the alphabetic sequence of supported languages.

AUTOMATIC
Cancels a HOLD by specifying that the programming language is set according to the current qualification and thereafter changed automatically each time the qualification changes or STEP or GO is issued.

HOLD
Specifies that the given language (or the current language, if no language is specified) remains in effect regardless of qualification changes. The language remains in effect until SET PROGRAMMING LANGUAGE changes the language or releases the hold.

ASSEMBLER
Sets the current programming language to ASSEMBLER.

C
Sets the current programming language to C. Debug Tool does not differentiate between C and C++, use this option for C++ as well as C programs.

COBOL
Sets the current programming language to COBOL.

DISASSEMBLY
Sets the current programming language to disassembly.

NONLECOBOL
Sets the current programming language to non-Language Environment COBOL.

PLI
Sets the current programming language to PL/I.

Usage notes
• If CYCLE or one of the explicit programming language names is specified, the current programming language setting is changed regardless of the currently suspended program or the current qualification.
• The current programming language setting affects how commands are parsed, not how they are performed. Commands are always performed according to the programming language setting where they were parsed. For example, it is not possible for a Debug Tool procedure to contain a mixture of C and COBOL commands; there is no way for the programming language setting to be changed while the procedure is being parsed. Also, it is not possible for a command parsed with one programming language setting to reference variables, types, or labels in another programming language.
If SET PROGRAMMING LANGUAGE AUTOMATIC is in effect (that is, HOLD is not in effect), changing the qualification automatically sets the current programming language to the specified block or compile unit.

SET PROGRAMMING LANGUAGE can be used to set the programming language to any supported language in the current or parent enclaves.

Example

Specify that C or C++ is the current programming language.
SET PROGRAMMING LANGUAGE C;

SET PROMPT 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.

```
SET PROMPT 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.

```
SET QUALIFY BLOCK block_spec CU cu_spec PROGRAM address LOAD load_spec RESET RETURN UP
```

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
• If SET PROGRAMMING LANGUAGE AUTOMATIC is in effect (that is, HOLD is not in effect), changing the qualification automatically sets the current programming language to the specified block or compile unit.
• If you are debugging a program that has multiple enclaves, you can issue the SET QUALIFY command only for the following items:
  – Load modules, compile units, and blocks that are known to Debug Tool and are in the current enclave
  – Load modules, compile units, and blocks that are not known to Debug Tool
  – Non-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 Debug Tool or library block.
• SET QUALIFY LOAD will not change the results of the QUERY QUALIFY command.
• If you specify cu_spec as a CU name without a load module name, Debug Tool searches for the CU in the following order:
  1. CU 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 Debug Tool, Debug Tool 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, Debug Tool 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.

Examples

- Indicate to Debug Tool that the load module statmod should be used when no load module is specified.
  SET QUALIFY LOAD statmod;

- Set the qualification back to the point of the suspended program.
  SET QUALIFY RESET;

- Set the block qualification to blockx. As a result, the load module qualification and compile unit qualification will be updated to the load module and compile unit that contain the block blockx.
  SET QUALIFY BLOCK blockx;

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

Related references
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.

```
SET REFRESH ON
OFF
```

- **ON**: Clears the screen before each rewrite. This is a required setting if your application handles line mode I/O.
- **OFF**: Rewrites without clear.

**Usage note**

SET REFRESH ON is needed for applications that also make use of the screen; for example, applications using ISPF services to display panels.

**Example**

Specify that rewrites only occur on those portions of the screen that have changed. The screen is not cleared before being rewritten.

```
SET REFRESH OFF;
```

SET RESTORE command

Controls the restoring of settings, breakpoints, and monitor specifications.

```
SET RESTORE SETTINGS NOAUTO
BPS AUTO MONITORS
```

**SETTINGS**

Indicates that SET values and WINDOW SIZE and WINDOW CLOSE settings are to be restored. The following SET values are not restored:

- SET DBCS
- SET FREQUENCY
- SET NATIONAL LANGUAGE
- SET PROGRAMMING LANGUAGE
- FILE operand of SET RESTORE SETTINGS
- SET QUALIFY
- SET SOURCE
- SET TEST

**BPS**

Indicates that breakpoints and LOADDEBUGDATA (LDD) specifications are to be restored. The following breakpoints are restored:
• APPEARANCE breakpoints
• CALL breakpoints
• DELETE breakpoints
• ENTRY breakpoints
• EXIT breakpoints
• GLOBAL APPEARANCE breakpoints
• GLOBALCALL breakpoints
• GLOBAL DELETE breakpoints
• GLOBAL ENTRY breakpoints
• GLOBAL EXIT breakpoints
• GLOBAL LABEL breakpoints
• GLOBAL LOAD breakpoints
• GLOBAL STATEMENT and GLOBAL LINE breakpoints
• LABEL breakpoints
• LOAD breakpoints
• OCCURRENCE breakpoints
• STATEMENT and LINE breakpoints
• TERMINATION breakpoint

MONITORS
Indicates that monitor and LOADDEBUGDATA (LDD) specifications are to be restored.

NOAUTO
Indicates that the specified data is not to be restored automatically at Debug Tool startup. It will be restored only when you explicitly request it by entering the RESTORE command. NOAUTO is the default until AUTO is specified.

AUTO
Indicates that, if possible, the specified data set is to be automatically restored by Debug Tool at startup.

Usage notes
• When SETTINGS are restored, they are restored before any preference or commands files are processed.
• When you use SET RESTORE BPS AUTO or SET RESTORE MONITORS AUTO, you must also use SET RESTORE SETTINGS AUTO. This ensures that Debug Tool knows the values of SET RESTORE BPS and SET RESTORE MONITORS the next time it starts up and then can implement them.
• Monitors are not necessarily restored to the same slot number from which they were saved.
• If you are debugging a CICS program and you want to use SET RESTORE parameter_name AUTO, you must log on with a user ID that is different from the default user ID.
• If you are debugging DB2 stored procedures, 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 use EQAopts to specify NULLFILE as the name of the default data set.

Related references
“RESTORE command” on page 198
“SET SAVE command” on page 231

SET REWRITE command (full-screen mode)
Forces a periodic screen rewrite during long sequences of output.

\[\text{SET REWRITE EVERY number ;} \]

number
Specifies how many lines of intercepted output are written by the application program before Debug Tool refreshes the screen. The initial setting is 50.

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.

\[\text{SET REWRITE EVERY number ;} \]

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 231

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 Debug Tool termination, the specified settings, breakpoint, or specifications are not to be saved. NOAUTO is the default until AUTO is specified.
AUTO
Indicates that, if possible, the specified data is to be saved at Debug Tool termination.

ONCE
Indicates that the settings information is to be saved once. The settings information is saved at the termination of the current debugging session but the saved value for SET SAVE SETTINGS is NOAUTO. This enables you save the settings of the current debugging session and not have the settings updated at the termination of subsequent debug sessions.

*  Indicates that the default file name is to be used to save settings, breakpoints, and monitor specifications at termination. The default name is userid(DBGTOOL).SAVESETS for settings and userid(DBGTOOL).SAVEBPS for breakpoints and monitor specifications. You can modify the default names by using EQAOPTS.

FILE setfileid
Indicates the data set name to be used to save and restore settings. The data set must exist before running this command.

In z/OS, setfileid is a DD name, a fully-qualified data set name (without apostrophes (')), or an HFS path and file name. In CICS, setfileid is a fully-qualified data set name or an HFS path and file name.

If setfileid is less than nine characters in length and does not contain a period, Debug Tool assumes it is a DD name. Otherwise, it is assumed to be a fully-qualified data set name.

In batch mode, the data set name is ignored. Use the INSPSAFE DD statement to indicate the name of the data set to use to restore and save settings.

This data set must be a sequential data set with a record format (RECFM) of VB and with a record length (LRECL) greater than or equal to 3204.

FILE bpfileid
Indicates the data set to be used to save breakpoints and monitor specifications. The data set must exist before running this command.

In z/OS, bpfileid is a DD name, a fully-qualified data set name (without apostrophes (')), or an HFS path and file name. In CICS, bpfileid is a fully-qualified data set name or an HFS path and file name.

If bpfileid is less than nine characters in length and does not contain a period, Debug Tool assumes it is a DD name. Otherwise, it is assumed to be a fully-qualified data set name.

In batch mode, the data set name is ignored. Use the INSPBPM DD statement to indicate the name of the data set to use to save breakpoints and monitor specifications.

This data set must be a PDS or PDSE (a PDSE is recommended) and you cannot specify a member name. This data set must have a record format (RECFM) of VB and with a record length (LRECL) greater than or equal to 3204. Debug Tool assigns a member name that is the load module name at enclave start. The breakpoints for each enclave are saved in a separate member of the PDS or PDSE. If you want to discard any saved breakpoints, LDD specifications, and monitor specifications, you can delete the member that has the name of the load module that started the enclave. Do not alter the contents of the member.

Usage notes
• You cannot use AUTO when you are debugging a CICS program and you are logged in with the same user ID as the default user ID.

• When you are debugging a CICS program, the CICS region must have update authorization to the SAVE SETTINGS and SAVE BPS data sets.

• When you enter the QUIT or QQ command from a nested enclave and the SET SAVE BPS AUTO, SET SAVE MONTORS AUTO, or both are in effect, only the data for the lowest level enclave is saved. No data for the higher level enclaves is saved.

• If you are debugging DB2 stored procedures, 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 use EQAOPTS to specify NULLFILE as the name of the default data set.

• Specifying setdsn for SAVE SETTINGS does not change the name of the data set from which AUTO RESTORE SETTINGS is done. It only changes the name of the data set used by AUTO SAVE SETTINGS and the RESTORE SETTINGS commands. AUTO RESTORE SETTINGS is always done from the default data set or DD name, depending on the environment.

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

Related tasks

Debug Tool User's Guide

Related references

“SET RESTORE command” on page 247
“RESTORE command” on page 198

SET SCREEN command (full-screen mode)
Controls how information is displayed on the screen. The initial setting is ON.

```plaintext
SET SCREEN
```

**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 no more than three objects specified and they must all be different. You cannot specify both MEMORY and LOG in the same sequence.
ON Activates the Debug Tool full-screen services.

OFF
Activates line mode. This mode is forced if the terminal is not a supported full-screen device.

Usage notes
- If neither CYCLE nor integer is specified, there is no change in the choice of configuration. If no windows are specified, there is no change in the assignment of windows to the configuration.
- If SET SCREEN OFF is entered while debugging in full-screen mode using a dedicated terminal under TSO, the session enters line mode using the TSO terminal. If SET SCREEN ON is later entered from the TSO terminal, control reverts to full-screen mode using a dedicated terminal.
- SET SCREEN OFF is ignored in CICS full-screen mode and in z/OS batch while debugging in full-screen mode using a dedicated terminal.

Examples
- Indicate that the Debug Tool full-screen services are used.
  SET SCREEN ON;
- Indicate that the log window is positioned above the Source window on the left hand side of the screen and the monitor window is to occupy the upper right side portion of the screen.
  SET SCREEN 2 LOG MONITOR SOURCE;

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

Related tasks
Debug Tool 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.

```
  SET SCROLL DISPLAY [ON] [OFF];
```

ON Displays scroll field.

OFF Suppresses scroll field.

Example

Specify that the scroll field is suppressed.
SET SCROLL DISPLAY OFF;

SET SEQUENCE command (PL/I)
Controls whether Debug Tool 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, Debug Tool 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 path and file name.

In CICS, *fileid* is a fully-qualified data set name or an HFS path and file name.

If *fileid* is less than nine characters in length and does not contain a period, Debug Tool assumes it is a DD name. Debug Tool checks to see if it is allocated. If it is not allocated, then *fileid* is assumed to be a data set name.

*Fileid* specifies a file identifier used in place of the default file identifier for the compile unit. A valid *fileid* is required unless it is already known to Debug Tool (by using a previous SET SOURCE command) or the default *fileid* is valid.

*Fileid* can not be a DD name if the data set allocated to it is C, C++ or Enterprise PL/I source and the EQAOPTS SUBSYS=ssss option is being used to access a source file in a library system.
Usage notes

- If you compiled your C or C++ program with the FORMAT(DWARF) suboption of the DEBUG compiler option, you cannot use the SET SOURCE command to specify the new location of the .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 Debug Tool, Debug Tool looks for a deferred LOADDEBUGDATA command with the specific cu_spec. If Debug Tool finds such a deferred LOADDEBUGDATA command, Debug Tool associates the fileid with the deferred LOADDEBUGDATA command. When the compile unit appears and is activated, Debug Tool 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 Debug Tool and issue:

  SET SOURCE ON (prog1) mainprog;

  When prog1 is made current during the debug session, Debug Tool searches for the listing in JSMITH.COBOL.LISTING(PROG1).

- Indicate that the COBOL listing associated with compile unit prog1 is found in DD name mainprog. In a TSO session:

  SET SOURCE ON (prog1) JSMITH.COBOL.LISTING(PROG1)

  This accomplishes the same result as the previous example without the execution of the ALLOCATE command.

- Indicate that the source associated with compile unit "/u/userid/code/oefun.c" is found in the HFS under the path and file name "/u/userid/code/oefun.c".

  SET SOURCE ON ("/u/userid/code/oefun.c") /u/userid/code/oefun.c;

- Indicate that the PL/I listing file associated with compile unit AVER is found in MYID.PLI.LISTING(AVER)

  SET SOURCE ON (AVER) myid.pli.listing(AVER) ;

- Indicate that the C source associated with compile unit JSMITH.C.SOURCE(myprog) is found in the PDS and member CODE.CLIB.SOURCE(myprog).

  SET SOURCE ON ("JSMITH.C.SOURCE(myprog)") CODE.CLIB.SOURCE(myprog)
Enter the SET LONGCUNAME OFF command to indicate that you want to use short 
CU names, then indicate that the C source associated with compile unit 
JSMITH.C.SOURCE(myprog) is found in the PDS and member 
CODE.CLIB.SOURCE(myprog):
SET LONGCUNAME OFF;
SET SOURCE ON (myprog) CODE.CLIB.SOURCE(myprog)

A PL/I program is compiled with 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 14
*"LIST command" on page 141
*"SET DEFAULT LISTINGS command" on page 217
*"SET DEFAULT DBG command" on page 216
*"SET DEFAULT MDBG command" on page 218
*"SET MDBG command" on page 236

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

Overides the initial TEST run-time options specified at invocation. The initial setting is ALL.

```
SET TEST test_level ;
```

`test_level`

Specifies what exception conditions cause Debug Tool to gain control, even though no breakpoint exists. The parentheses are optional.

`Test_level` can include the following:

**ALL**

Specifies that Debug Tool 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 THREADTERMCOND option in the EQAOPTS option file is specified, Debug Tool does not gain control. Contact your system administrator to determine if this option was specified.

If a condition occurs and a breakpoint exists for the condition, Debug Tool 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, Debug Tool does one of the following options:

- In interactive mode, Debug Tool reads commands from a commands file (if it exists) or prompts you for commands.
- In noninteractive mode, Debug Tool reads commands from the commands file.

**ERROR**

Specifies that only the following conditions cause Debug Tool to gain control without a user-defined breakpoint.

- An MVS or CICS ABEND that occurs when you are running without the Language Environment run time

For C:

- An attention interrupt
- A predefined Language Environment condition of Severity 2 or above
- Any C condition other than SIGUSR1, SIGUSR2, SIGINT or SIGTERM.

For COBOL:

- An attention interrupt
- A predefined Language Environment condition of Severity 2 or above.

For PL/I:

- An attention interrupt, directed at either PL/I or Debug Tool
- A predefined Language Environment condition of Severity 2 or above.
If a breakpoint exists for one of the above conditions, any commands specified in the breakpoint are executed. If no commands are specified, Debug Tool reads commands from a commands file or prompts you for commands in interactive mode.

**NONE**

Specifies that Debug Tool gains control only at an attention interrupt, or at a condition if a breakpoint is defined for that condition. If a breakpoint does exist for the condition, the commands specified in the breakpoint are executed.

**Usage note**

If the THREADTERMCOND option in the EQAOPTS file is set to prevent Debug Tool from stopping when a FINISH, CEE066, or CEE067 thread termination condition is raised by Language Environment, Debug Tool does not gain control when these conditions are raised. If you want Debug Tool to gain control when these conditions are raised, you can set an AT OCCURRENCE breakpoint or change the THREADTERMCOND option to allow Debug Tool to gain control.

**Examples**

- Indicate that only an attention interrupt or exception causes Debug Tool to gain control when no breakpoint exists.
  
  ```
  SET TEST ERROR;
  ```

- Indicate that no condition causes Debug Tool to gain control unless a breakpoint exists for that condition.
  
  ```
  SET TEST NONE;
  ```

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

**Related tasks**

[Debug Tool User’s Guide](#)

**Related references**

- “AT OCCURRENCE command” on page 67
- [z/OS Language Environment Debugging Guide](#)

---

**SET WARNING command (C, C++, COBOL, and PL/I)**

Controls display of the Debug Tool warning messages and whether exceptions are reflected to the C, C++, and PL/I programs. For COBOL programs, controls the ability to modify variables while you debug optimized code. The initial setting is ON.

```
SET WARNING [ON | OFF]
```

**ON**

Displays the Debug Tool warning messages, and conditions such as a divide check result in a diagnostic message. For COBOL programs, prohibits the modification of variables while you debug optimized programs.

**OFF**

Suppresses the Debug Tool warning messages, and conditions raise an exception in the program. For COBOL programs, allows the modification of variables while you debug optimized programs.

---

258 Debug Tool V10 Reference and Messages
Exceptions that occur due to interaction with you are likely to be due to typing errors and are probably not intended to be passed to the application program. However, you might want to raise a real exception in the program, for example, to test some error recovery code. (TRIGGER is not always appropriate for this because it does not set up the exception information.)

Usage notes

- You can use this command in remote debug mode.
- Debug Tool detects C conditions such as the following:
  - Division by zero
  - Array subscript out of bounds for defined arrays
  - Assignment of an integer value to a variable of enumeration data type where the integer value does not correspond to an integer value of one of the enumeration constants of the enumeration data type.
- Debug Tool detects the following PL/I computational conditions:
  - Invalid decimal data
  - CHARACTER to BIT conversion errors
  - Division by zero
  - Invalid length in varying strings
- You can modify variables in an optimized program that was compiled with one of the following compilers:
  - Enterprise COBOL for z/OS, 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 EJP0 suboption of the TEST compiler option. However, variables that are declared with the VALUE clause to initialize them cannot be modified.

- When Debug Tool evaluates a conditional expression (for example, the condition of the WHEN clause of the AT CHANGE command) and the conditional expression is invalid, then Debug Tool does one of the following actions:
  - If SET WARNING is set to ON, Debug Tool 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 Debug Tool to take.
  - If SET WARNING is set to OFF, Debug Tool does not stop nor display a message that it could not evaluate the conditional expression. Debug Tool 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.

- "Controlling Debug Tool when a comparison is invalid" in *Debug Tool User’s Guide*
SET command (COBOL)

The SET command assigns a value to a COBOL reference. The SET keyword cannot be abbreviated.

```
SET reference TO reference literal TRUE;
```

**reference**
A valid Debug Tool COBOL reference.

**literal**
A valid COBOL numeric literal constant.

**TRUE**
The value assigned to a level-88 *reference*.

In order to assign the value TRUE, the PTF for Language Environment APAR PK30521 must be installed on z/OS Version 1 Release 6, Version 1 Release 7, and Version 1 Release 8.

**Usage notes**

- You can assign the value TRUE only to a level-88 *reference*.
- If Debug Tool was started because of a computational condition or an attention interrupt, using an assignment to set a variable might not give expected results. This is due to the uncertainty of variable values within statements as opposed to their values at statement boundaries.
- SET assigns a value only to a single receiver; unlike COBOL, multiple receiver variables are not supported.
- Only formats 1, 4 and 5 of the COBOL SET command are supported.
- Index-names can only be program variables (because OCCURS is not supported for the Debug Tool session variables).
- COBOL ADDRESS OF identifier is supported only for identifiers that are LINKAGE SECTION variables. In addition, COBOL ADDRESS 0F as a receiver must be level 1 or 77, and COBOL ADDRESS 0F as a sender can be any level except 66 or 88.
- Debug Tool provides a hexadecimal constant that can be used with the SET command, where the hexadecimal value is 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'20000';

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

Related tasks
  Debug Tool User’s Guide

Related references
  “Allowable moves for the Debug Tool SET command”

Allowable moves for the Debug Tool SET command

The following table shows the allowable moves for the Debug Tool SET command.
<table>
<thead>
<tr>
<th>Source Field</th>
<th>AO</th>
<th>IN</th>
<th>IDI</th>
<th>PTR</th>
<th>ED</th>
<th>BI</th>
<th>ID</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address Of (AO)</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index Name (IN)</td>
<td></td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Index Data Item (IDI)</td>
<td></td>
<td></td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pointer Data Item (PTR)</td>
<td></td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address Hex Literal¹</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NULL (NUL)</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integer Literal</td>
<td></td>
<td></td>
<td>Y²</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>External Decimal (ED)</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary (BI)</td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal Decimal (ID)</td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object Reference (OR)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
</tr>
</tbody>
</table>
**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.

```plaintext
SHOW integer
```

`integer` Selects a relative statement (for C) or a relative verb (for COBOL) within the line. The default value is 1. For optimized COBOL programs, the default value is the first executable statement which was not discarded by the optimizer.

**Usage notes**

- If `SET SUFFIX` is currently OFF, `SHOW` prefix forces it ON.
- The suffix display returns to normal on the next interaction.
- The `SHOW` prefix command is not logged.

**Example**

Display the frequency count of the third statement or verb in the line (typed in the prefix area of the line where the statement is found).

```
SHOW 3
```

No space is needed as a delimiter between the keyword and the integer; hence, `SHOW 3` is equivalent to `SHOW3`.

**STEP command**

The `STEP` command causes Debug Tool to dynamically step through a program, executing one or more program statements. In full-screen mode, it provides animated execution.

`STEP` ends if one or more of the following conditions is reached:
- User attention interrupt
- A breakpoint is encountered
- Normal or unusual termination of the program

```plaintext
STEP integer INTO OVER RETURN
```

`integer` Indicates the number of statements performed. The default value is 1. If `integer` is greater than 1, the statement is performed as if it were that many repetitions.
of STEP with the same keyword and a count of one. The speed of execution, or
the pace of stepping, is set by either the SET PACE command, or with the Pace of
visual trace field on the Profile panels.

* Specifies that the program should run until interrupted. STEP * is equivalent to
GO.

INTO
Steps into any called procedures or functions. This means that stepping
continues within called procedures or functions.

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, Debug Tool can not
resume STEP mode on return and the application continues to run until it ends.

RETURN
Steps to the return point the specified number of levels back, halting at the
statement following the corresponding procedure call or function invocation.
This operand provides full-speed execution (with no animation) for the
remainder of the current procedure or function, and for any called procedures
or functions, resuming STEP mode on return.

If you are debugging a non-Language Environment COBOL or disassembled
program, do not use the STEP RETURN command because Debug Tool cannot
identify the return point. Instead, set a breakpoint in the calling program and
enter the GO command.

Usage notes
• In the figure below, PGM A calls PGM B.
Assume that the current execution point is on PGM B and, at the line `ADD 5 TO MYNUM`. At this point, you decide you don’t need to see any more of the code in PGM B. By issuing `STEP RETURN` on the command line, Debug Tool returns to the first line of code after the `CALL` command that called PGM B, as indicated by the arrow. You can then continue stepping through PGM A.

- If `STEP` is specified in a command list (for example, as the subject of an `IF` command or `WHEN` clause), all subsequent commands in the list are ignored.
- If `STEP` is specified within the body of a loop, it causes the execution of the loop to end.
- To suppress the logging of `STEP` commands, use the `SET ECHO` command.
- If two operands are given, they can be specified in either order.
- The animation execution timing is set by the `SET PACE` command.
- The source panel provides a means of suppressing the display of selected listings or files. This gives some control of "debugging scope," because animated execution does not occur within a load module where the source listing or source file is not displayed.
- If you are debugging a disassembled program and attempt to step out of the current CU, a message appears. The message informs you to set a breakpoint outside the current CU. Without that breakpoint, Debug Tool cannot stop the application. After you have set the breakpoint, you can resume running your application by entering a Debug Tool command like `STEP` or `GO`.
- If you are debugging a program that does not use the standard linkage conventions for R13, R14, and R15, and you enter the `STEP RETURN` or the `STEP` command on a statement that returns to a higher level CU, Debug Tool does not stop at the expected instruction in the higher-level CU.
- When `PLAYBACK ENABLE` is in effect, you can use the `STEP` command to move forward or backward one or more statements. You cannot use the INTO, OVER, and RETURN keywords. Each `STEP` command moves forward or backward the number of statements specified or implied by the integer parameter.
- If the `DATA` option of the `PLAYBACK ENABLE` command is in effect, you can access program variables after each `STEP` command.
- You can use the `STEP` command in remote debug mode only by entering it in the Action field, which is in the Optional Parameters section of the Add a Breakpoint task.

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.

```
STORAGE (address) = value;
```

- **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 non-Language Environment COBOL variable whose storage location is to be changed. In non-Language Environment 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.
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, 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, Debug Tool assumes the second parameter is the length.
- If you specify only one parameter, Debug Tool assumes the offset is 0 and that the length is equal to the length of value.
- The STORAGE command can not be used while you replay recorded statements by using the PLAYBACK commands.
- If you specify address with more than 8 significant digits or if reference references 64-bit addressable storage, Debug Tool assumes that the storage location is 64-bit addressable storage. Otherwise, Debug Tool assumes that the storage location is 31-bit addressable storage.
- If reference is a pointer, Debug Tool 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'000001234'
- 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(expression) { switch_body ;}
```

**switch_body:**

```
switch_body:
  case_clause
  default_clause
```

**case_clause:**

```
case case_expression: command
```
**default_clause:**

```
default_clause:
| default | : command |
```

**expression**

A valid Debug Tool C expression.

**case_expression**

A valid character or optionally signed integer constant.

**command**

A valid Debug Tool command.

The value of the `switch` expression is compared with the value of the expression in each `case` clause. If a matching value is found, control is passed to the command in the `case` clause that contains the matching value. If a matching value is not found and a `default` clause appears anywhere in the `switch` body, control is passed to the command in the `default` clause. Otherwise, control is passed to the command following the `switch` body.

If control passes to a command in the `switch` body, control does not pass from the `switch` body until a `break` command is encountered or the last command in the `switch` body is performed.

**Usage notes**

- Declarations are not allowed within a `switch` command.
- The `switch` command does not end with a semicolon. A semicolon after the closing brace is treated as a `Null` command.
- Although this command is similar to the `switch` statement in C, it is subject to Debug Tool restrictions on expressions.
- Duplicate `case_expression` values are not supported.
- You cannot use the `switch` command while you replay recorded statements by using the `PLAYBACK` commands.

**Examples**

- The following `switch` command contains several `case` clauses and one `default` clause. Each clause contains a function call and a `break` command. The `break` commands prevent control from passing down through subsequent commands in the `switch` body.
  
  If `key` has the value `'/`, the `switch` command calls the function `divide`. On return, control passes to the command following the `switch` body.

  ```c
  char key;
  printf("Enter an arithmetic operator\n");
  scanf("%c", &key);

  switch (key) {
  case '+':
    add();
    LIST (key);
    break;
  case '-':
    ```
subtract();
LIST (key);
break;
case '*':
multiply();
LIST (key);
break;
case '/':
divide();
LIST (key);
break;
default:
printf("Invalid key\n");
break;
}

- In the following example, break commands are not present. If the value of c is equal to 'A', all 3 counters are incremented. If the value of c is equal to 'a', lettera and total are increased. Only total is increased if c is not equal to 'A' or 'a'.

```c
char text[100];
int capa, i, lettera, total;

for (i=0; i < sizeof(text); i++) {
    switch (text[i]) {
      case 'A':
        capa++;
      case 'a':
        lettera++;
      default:
        total++;
    }
}
```

---

**SYSTEM command (z/OS)**

The SYSTEM command lets you issue TSO commands during a Debug Tool session. The SYSTEM keyword can only be abbreviated as SYS.

```bash
SYSTEM system_command;
```

*system_command*

A valid TSO system command or CLIST name that does not require a parameter.

**Usage notes**

- No parameters can be specified as part of the system command or CLIST invocation. To execute noninteractively when parameters are required, you must enter the complete invocation in a CLIST and then use a TSO or SYSTEM command to call that CLIST (without parameters).
- You cannot introduce a new Debug Tool session using the SYSTEM command.
- When operating interactively in TSO, there is no provision for entering a mode where commands are accepted repeatedly; however, it is possible to write your own such iterative sequence in a CLIST.
- You cannot issue CICS commands using SYSTEM.

**Examples**
• List all the data sets in the user catalog.
  
  `SYSTEM LISTCAT;`

• Temporarily places you in ISPF mode.
  
  `SYSTEM PDF;`

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

**Related references**

“TSO command (z/OS)” on page 274

---

**TRIGGER command**

The TRIGGER command raises the specified AT-condition in Debug Tool, or it raises the specified programming language condition in your program.
storage_clause:

\[
\text{%STORAGE}(-address-, -length-)\]

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
- SIGABRT
- SIGFPE
- SIGILL
- SIGINT
- SIGIOERR
- SIGSEGV
- SIGTERM
- SIGUSR1
- SIGUSR2

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

PL/I condition constants can be used; for syntax and acceptable abbreviations see the ON command.

When you are running without the Language Environment run time, use one of the following codes:

- Codes Sxxx and Uxxx to represent MVS System and User ABENDs. In this case the xxx is three hexadecimal digits representing the ABEND code.
- Any four-character string to represent a CICS ABEND code.

cu_spec

A valid compile unit specification.

entry_name

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

reference

A valid Debug Tool reference in the current programming language.

%STORAGE

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

address

The starting address of storage to be watched for changes.

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 THREADTERMCOND option in the EQAOPTS file is set to prevent Debug Tool from stopping when a FINISH, CEE066, or CEE067 thread termination condition is raised by Language Environment, Debug Tool does not gain control when these conditions are raised. If you want Debug Tool to gain control when these conditions are raised, you can set an AT OCCURRENCE breakpoint or change the THREADTERMCOND option to allow Debug Tool 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 Debug Tool commands associated with the breakpoint. The condition is not raised.
- Triggering a condition (TRIGGER CEE347), which raises the condition and causes a corresponding system action. A corresponding system action can be a condition handler.
- Perform the commands in the AT OCCURRENCE CEE347 breakpoint (the CEE347 condition is not raised). The current programming language setting is COBOL.

```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.

```c
TRIGGER SIGTERM;
```
- A previously defined STATEMENT breakpoint (for line 13) is triggered.

```c
AT 13 LIST "at 13";
TRIGGER AT 13;
/* "at 13" will be the echoed output here */
```
- Assume the following breakpoints exist in a program:

```c
AT CHANGE x LIST TITLED (x); AT STATEMENT 10;
```

If Debug Tool is started for the STATEMENT breakpoint and you want to trigger the commands associated with the AT CHANGE breakpoint, enter:

```c
TRIGGER AT CHANGE x;
```

Debug Tool displays the value of `x`.

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

**Related tasks**
TSO command (z/OS)

The TSO command lets you issue TSO commands during a Debug Tool session and is valid only in a TSO environment. The TSO keyword cannot be abbreviated.

```
TSO tso_command;
```

**tso_command**

A valid TSO system command or CLIST name that does not require a parameter.

**Usage notes**

- TSO is synonymous to SYSTEM.

**Example**

List all the data sets in the user catalog.

```
TSO LISTCAT;
```

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

**Related references**

“SYSTEM command (z/OS)” on page 270

USE command

The USE command causes the Debug Tool commands in the specified file or data set to be either performed or syntax checked. This file can be a log file from a previous session. The specified file or data set can itself contain another USE command. The maximum number of USE files open at any time is limited to eight. The USE keyword cannot be abbreviated.

```
USE ddname dsname;
```

**ddname**

A valid ddname in z/OS.

**dsname**

A z/OS data set containing the Debug Tool commands to be performed. If **dsname** is not enclosed in apostrophes (‘), Debug Tool assumes it is a partially-qualified data set name and the user ID is prefixed to form the fully-qualified data set name.
Usage notes

- To check the syntax of the commands in a USE file:
  1. Set the EXECUTE setting to 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 Debug Tool session.
- Recursive calls are not allowed; that is, a commands file cannot be used if it is already active. This includes the primary commands and preferences files. If another invocation of Debug Tool occurs during the execution of a USE file (for example, if a condition is raised while executing a command from a USE file), the USE file is not used for command input until control returns from the condition.
- The USE file is closed when the end of the file is reached.
- If a nonreturning command (such as GO) is performed from a USE file, the action taken (as far as closing the USE file) depends on certain things:
  - If the USE file was called directly or indirectly from the primary commands file or preferences file, it has the same characteristics as the primary commands file or preferences file. That is, it "keeps its place" and the next time Debug Tool requests a command, it reads from the USE file where it left off.
  - If the USE file was not called directly or indirectly from the primary commands file or preferences file, the rest of the USE file and the file that called the USE file is skipped.
- If the end of the USE file is reached without encountering a QUIT command, Debug Tool returns to the command source where the USE command was issued. This can be the terminal, a command string, or another commands file.
- A USE file takes on the aspects of whatever command source issued the USE command, relative to its behavior when a GO, GOTO, or STEP is executed. When called from the primary commands file, it continues with its next sequential command at the next breakpoint. If it is called from any other command sequence, the GO, GOTO, or STEP causes any remaining commands in the USE file to be discarded.

Examples

- Perform the Debug Tool commands in the z/OS data set USERID.COMMANDS.FILE. The data set must first be allocated with, for example, ALLOC FI(MYCMDS) DA('USERID.COMMANDS.FILE').
  USE MYCMDS;

  Alternatively, perform the commands in the z/OS data set
  USERID.COMMANDS.FILE.
  USE COMMANDS.FILE

- On z/OS, perform the Debug Tool commands in the partitioned data set member USERID.PDS(CMDS).
  USE PDS(CMDS)

- For CICS, perform Debug Tool commands in the fully-qualified data set
  TS64081.USE.FILE.
  USE 'TS64081.USE.FILE';

In addition to using sequential files, you can perform Debug Tool commands using partitioned data sets.
USE 'userid.thing.file(usefile)'
while command (C and C++)

The while command enables you to repeatedly perform the body of a loop until the specified condition is no longer met or evaluates to false. The while keyword must be lowercase and cannot be abbreviated.

```
while(expression) command;
```

**expression**

A valid Debug Tool C expression.

**command**

A valid Debug Tool command.

The expression is evaluated to determine whether the body of the loop should be performed. If the expression evaluates to false, the body of the loop never executes. Otherwise, the body does execute. After the body has been performed, control is given once again to the evaluation of the expression. Further execution of the action depends on the value of the condition.

A break command can cause the execution of a while command to end, even when the condition does not evaluate to false.

**Usage notes**

- If you are replaying recorded statements by using the PLAYBACK commands, then you cannot use the while command.

**Examples**

- List the values of x starting at 3 and ending at 9, in increments of 2.
  
  ```
  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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WINDOW CLOSE command on page 277</td>
<td>Closes the specified window in the Debug Tool full-screen session panel.</td>
</tr>
<tr>
<td>WINDOW OPEN command on page 277</td>
<td>Opens a previously-closed window in the Debug Tool full-screen session panel.</td>
</tr>
<tr>
<td>WINDOW SIZE command on page 278</td>
<td>Controls the relative size of currently visible windows in the Debug Tool full-screen session panel.</td>
</tr>
</tbody>
</table>
"WINDOW SWAP command" on page 279
Replaces the logical window being displayed in a physical window with another logical window.

"WINDOW ZOOM command" on page 280
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 Debug Tool 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, Debug Tool displays an error message.

If there is only one physical window visible, WINDOW CLOSE is invalid.

```
WINDOW CLOSE CURSOR;
```

**WINDOW OPEN command**

Opens a previously-closed physical window in the Debug Tool 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, Debug Tool displays an error message.

If the OPEN command is issued without an operand, Debug Tool opens the last closed physical window.

```
WINDOW OPEN LOG MEMORY MONITOR;
```

**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 Debug Tool full-screen session panel.

```
WINDOW SIZE integer CURSOR LOG MEMORY MONITOR SOURCE;
```

**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 \texttt{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.
  \texttt{WINDOW SIZE 15 SOURCE;}
- Adjust the size of the window where the cursor is currently positioned to 20 rows.
  \texttt{SIZE 20 CURSOR;}

\textbf{WINDOW SWAP command}

The \texttt{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 \texttt{SWAP} command, it remains closed until you enter the \texttt{WINDOW OPEN} command.

\begin{verbatim}
\textbf{WINDOW SWAP MEMORY LOG} \rightarrow \textbf{MEMORY SWAP LOG} \rightarrowbrities
\end{verbatim}

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:
  \texttt{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.

\textbf{Related tasks}
- [Debug Tool session panel] in \textit{Debug Tool 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. Debug Tool built-in functions

Debug Tool provides you with several built-in functions which allow you to manipulate variables. All Debug Tool built-in function names begin with a percent sign (%).

The table below summarizes the Debug Tool built-in functions. Unless otherwise indicated, the functions can be used with all supported languages.

<table>
<thead>
<tr>
<th>Debug Tool built-in function</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>%DEC (assembler, disassembly, and non-Language Environment COBOL)</td>
<td>Decimal value of an operand.</td>
</tr>
<tr>
<td>%GENERATION (PL/I)</td>
<td>A specific generation of a controlled variable</td>
</tr>
<tr>
<td>%HEX on page 282</td>
<td>Hexadecimal value of an operand</td>
</tr>
<tr>
<td>%INSTANCES (C, C++, and PL/I) on page 283</td>
<td>Maximum value of %RECURSION for a block</td>
</tr>
<tr>
<td>%RECURSION (C, C++, and PL/I) on page 284</td>
<td>An automatic variable or a parameter in a specific instance of a recursive procedure</td>
</tr>
<tr>
<td>%WHERE (assembler, disassembly, and non-Language Environment COBOL) on page 285</td>
<td>A string indicating the address of the operand.</td>
</tr>
</tbody>
</table>

### %DEC (assembler, disassembly, and non-Language Environment COBOL)

Returns the decimal value of an operand.

```plaintext
%DEC(expression);  
```

**expression**

A valid assembler, disassembly, or non-Language Environment COBOL expression.

**Examples**

Assuming register R1 contains the value 14, to display the value of the expression R1+2 in decimal, enter the following command:

```plaintext
LIST %DEC(R1+2);  
```

The Log window displays the value 16.

Refer to the following 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.

%GENERATION(—reference—,—expression—);

reference
A controlled variable.

expression
The generation number \( n \) of a controlled variable \( x \), where:
\[ 1 \leq n \leq \text{ALLOCATION}(x) \]

To return the oldest instance of \( x \), specify:
%GENERATION(\(x\),1)

To return the most recent instance of \( x \), specify:
%GENERATION(\(x\),\text{ALLOCATION}(x))

Usage notes
• You cannot use the %GENERATION built-in function while you replay recorded steps.
• If you want to use the %GENERATION built-in function while debugging Enterprise PL/I programs, you must 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
Debug Tool User’s Guide

%HEX

Returns the hexadecimal value of an operand.

%HEX(—reference—);

reference
One of the following:
• COBOL or PL/I reference
• C or C++ lvalue
• assembler, disassembly, or non-Language Environment COBOL expression

Examples
C and C++: To display the internal representation of the packed decimal variable \(z\text{var1}\) whose external representation is 235, enter the following command.
LIST %HEX(z\text{var1});

The Log window displays the hexadecimal string 235C.

COBOL: To display the external representation of the packed decimal \(p\text{var3}\), defined as \text{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.

```plaintext
%INSTANCES(—reference—);
```

*reference*

An automatic variable or a subroutine parameter. If necessary, you can use qualification to specify the variable.

%INSTANCES can be used like a Debug Tool variable.

**Usage notes**

Debug Tool does not support the %INSTANCES built-in function for Enterprise PL/I programs.

You cannot use the %INSTANCES built-in function while you replay recorded steps.

**Examples**

C and C++:

- %INSTANCES and %RECURSION can be used together to determine the number of times a function is recursively called. They can also give you access to an automatic variable or parameter in a specific instance of a recursive procedure.
  Assume, for example, your program contains the following statements.

```c
int RecFn(unsigned int i) {
  if (i == 0) {
    __ctest("");
  }
}
```

At this point, the __ctest() call gives control to Debug Tool, and you are prompted for commands. Enter the following command.

```plaintext
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.

```plaintext
%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.

```plaintext
%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");
• Debug Tool performs the necessary conversions when a call to a library function is made. The cast operator can be used. In the following example, the integer 2 is converted to a double, which is the required argument type for sqrt.
  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)"

%RECURSION (C, C++, and PL/I)

Returns a specific instance of an automatic variable or a parameter in a recursive procedure.

%RECURSION(—reference—,-—expression—);________________________

reference
  An automatic variable or a subroutine parameter. If necessary, you can use qualification to specify the variable.

expression
  The recursion number of the variable or parameter.
  To return the oldest recursion of x, specify:
  %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 Debug Tool references.
• %RECURSION can be used like a Debug Tool variable.
• You cannot use the %RECURSION built-in function while you replay recorded steps.

Refer to the following topics for more information related to the material discussed in this topic.
%WHERE (assembler, disassembly, and non-Language Environment 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.

%WHERE(expression);

expression
An expression that evaluates to a hexadecimal value that is one to four bytes in length.

The following rules are used to evaluate the value of the expression, in the order listed:
1. If the expression value is less than 4096, a decimal number is returned.
2. If the expression value is an address within any known compile unit, the name of the compile unit with a hexadecimal offset is returned.
3. If the expression value is within 4095 bytes of the value in a general register and a USING is in effect for that register, the name of the DSECT that corresponds to the USING instruction with a hexadecimal offset is returned.
4. A hexadecimal number representing the expression value is returned.

Although this function can be used only within an assembler, disassembly, or non-Language Environment 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
  \hl{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 7. Debug Tool variables

Debug Tool reserves several variables for its own information. These Debug Tool variable names begin with a percent sign (%), to distinguish them from program variables. You can access Debug Tool variables while testing programs in any supported language.

You can use all Debug Tool variables in expressions. Additionally, the variables %EPRn, %FPRn, %GPRn, and %LPRn. (representing the types of registers) can be modified, as shown in the COBOL example below.

```
MOVE name_table TO %GPR5;
```

**Note:** Use caution when assigning new values to registers. Important program information can be lost. Do not modify the base register.

To display the value of a Debug Tool variable, use the LIST command, as shown in the example below.

```
LIST %GPR15
```

The table below summarizes the Debug Tool variables.

<table>
<thead>
<tr>
<th>Debug Tool variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;%ADDRESS&quot; on page 289</td>
<td>Address of the location where your program was interrupted</td>
</tr>
<tr>
<td>&quot;%AMODE&quot; on page 289</td>
<td>Current® AMODE of the suspended program</td>
</tr>
<tr>
<td>&quot;%BLOCK&quot; on page 289</td>
<td>Name of the current block</td>
</tr>
<tr>
<td>&quot;%CAAADDRESS&quot; on page 289</td>
<td>Address of the CAA control block associated with the suspended program</td>
</tr>
<tr>
<td>&quot;%CC (assembler and disassembly only)* on page 289</td>
<td>(Assembler and disassembly only) Condition code from current PSW</td>
</tr>
<tr>
<td>&quot;%CONDITION&quot; on page 290</td>
<td>Name or number of the condition when Debug Tool is entered because of an AT OCCURRENCE</td>
</tr>
<tr>
<td>&quot;%COUNTRY&quot; on page 290</td>
<td>Current country code</td>
</tr>
<tr>
<td>&quot;%CU&quot; on page 290</td>
<td>Name of the primary entry point of the current compile unit</td>
</tr>
<tr>
<td>&quot;%EPA&quot; on page 290</td>
<td>Address of the primary entry point in the current compile unit</td>
</tr>
<tr>
<td>&quot;%EPRn or %EPRHn (%EPRHn assembler and disassembly only)* on page 290</td>
<td>(Assembler, disassembly, C and C++, and PL/I only) Extended-precision floating-point registers</td>
</tr>
<tr>
<td>&quot;%EPRBn (assembler and disassembly only)* on page 291</td>
<td>(Assembler and disassembly only) Extended-precision floating-point registers in binary format</td>
</tr>
<tr>
<td>&quot;%EPRDn (assembler and disassembly only)* on page 291</td>
<td>(Assembler and Disassembly only) Extended-precision floating-point registers in decimal format</td>
</tr>
<tr>
<td>&quot;%FPRn or %FPRHn (%FPRHn assembler and disassembly only)* on page 291</td>
<td>Single-precision floating-point registers in hexadecimal format</td>
</tr>
<tr>
<td>Debug Tool variable</td>
<td>Value</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------</td>
</tr>
<tr>
<td><code>%FPRBn</code> (assembler and disassembly only) on page 292</td>
<td>(Assembler and Disassembly only) Single-precision floating-point registers in binary format</td>
</tr>
<tr>
<td><code>%FPRDn</code> (assembler and disassembly only) on page 292</td>
<td>(Assembler and Disassembly only) Single-precision floating-point registers in decimal format</td>
</tr>
<tr>
<td><code>%GPRn</code> on page 293</td>
<td>32-bit General Purpose Registers at the point of interruption in a program</td>
</tr>
<tr>
<td><code>%GPRGn</code> on page 293</td>
<td>64-bit General Purpose Registers at the point of interruption in a program</td>
</tr>
<tr>
<td><code>%HARDWARE</code> on page 294</td>
<td>Type of hardware where the application is running</td>
</tr>
<tr>
<td><code>%LINE or %STATEMENT</code> on page 294</td>
<td>Current source line number</td>
</tr>
<tr>
<td><code>%LOAD</code> on page 294</td>
<td>Name of the load module of the current program, or an asterisk (*)</td>
</tr>
<tr>
<td>`%LPRBn (or %LPRHn (LPRHn as assembler and disassembly only)) on page 295</td>
<td>Double-precision floating-point registers in hexadecimal format</td>
</tr>
<tr>
<td><code>%LPRDn (assembler and disassembly)</code> on page 295</td>
<td>(Assembler and Disassembly only) Double-precision floating-point registers in binary format</td>
</tr>
<tr>
<td><code>%LPRDn (assembler and disassembly)</code> on page 296</td>
<td>(Assembler and Disassembly only) Double-precision floating-point registers in decimal format</td>
</tr>
<tr>
<td><code>%NLANGUAGE</code> on page 296</td>
<td>Integer identifying the type of change occurring when the program flow reaches a point of discontinuity, and the path condition is raised</td>
</tr>
<tr>
<td><code>%NLANGUAGE</code> on page 296</td>
<td>National language currently in use</td>
</tr>
<tr>
<td><code>%PATHCODE</code> on page 296</td>
<td>Current programming language</td>
</tr>
<tr>
<td><code>%PLANGUAGE</code> on page 296</td>
<td>Equivalent to <code>%CU</code></td>
</tr>
<tr>
<td><code>%PROGMASK (assembler and disassembly only)</code> on page 296</td>
<td>Program mask from current PSW</td>
</tr>
<tr>
<td><code>%PROGMASK (assembler and disassembly only)</code> on page 296</td>
<td>Equivalent to <code>%CU</code></td>
</tr>
<tr>
<td><code>%PSW</code> (assembler and disassembly only) on page 297</td>
<td>Current Program Status Word</td>
</tr>
<tr>
<td><code>%RC</code> on page 297</td>
<td>Return code from the most recent Debug Tool command</td>
</tr>
<tr>
<td><code>%RSTDSETS</code> on page 297</td>
<td>A value of 1 if user settings have been restored and 0 otherwise</td>
</tr>
<tr>
<td><code>%RUNMODE</code> on page 297</td>
<td>String identifying the presentation mode of Debug Tool</td>
</tr>
<tr>
<td><code>%STATEMENT</code></td>
<td>Equivalent to <code>%LINE</code></td>
</tr>
<tr>
<td><code>%SUBSYSTEM</code> on page 298</td>
<td>Name of the underlying subsystem, if any, where the program is running</td>
</tr>
<tr>
<td><code>%SYSTEM</code> on page 298</td>
<td>Name of the operating system supporting the program</td>
</tr>
</tbody>
</table>

You can access Debug Tool variables even when they have no intrinsic meaning in your operating system or language.
Refer to the following topics for more information related to the material discussed in this topic.

Related references

“Attributes of Debug Tool variables in different languages” on page 298

%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 Debug Tool is unable to locate the last statement that was executed before control left the program.

%AMODE

Contains the current AMODE of the suspended program: 24, 31, or 64. For COBOL programs, the value is always 31.

%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 102

“LIST expression command” on page 149

“SET QUALIFY command” on page 244

%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 Debug Tool 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 67

---

%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 244

---

%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 %EPRHn (%EPRHn 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 %EPRn or %EPRHn variable.

%EPRn and %EPRHn cannot be used as the target of an assignment while debugging Enterprise PL/I programs.

You cannot use the %EPRn or %EPRHn variable while you are replaying recorded statements.

**Usage note**

---
For assembler and disassembly, the LIST %EPRn command displays values in hexadecimal but the LIST %EPRHn 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 38
“Assignment command (assembler and disassembly)” on page 35

%EPRBn (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 %EPRBn 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 %EPRBn 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 35

%EPRDn (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 %EPRDn 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 %EPRDn 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 35

%FPrn or %FPRHn (%FPRHn 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 %FPRn or %FPRHn variable.

%FPRn and %FPRHn cannot be used as the target of an assignment while debugging Enterprise PL/I programs.

Usage note

For assembler and disassembly, the LIST %FPRn command displays values in hexadecimal, but for the LIST %FPRHn 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 171
“Assignment command (PL/I)” on page 38
“Assignment command (assembler and disassembly)” on page 35

%FPRBn (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 %FPRBn 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 35

%FPRDn (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 %FPRDn 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.
%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 171
"Assignment command (assembler and disassembly)" on page 35
"Assignment command (PL/I)" on page 38

%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 variable.

**Usage notes**
- If you modify a %GPRG register, the change is reflected when you resume program execution.
- In disassembly and assembler, you can replace GRn with %GPRn.
- 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 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 (*).

Debug Tool uses the value of %LOAD when you make an unqualified reference to a program or variable.

To change the current load module, use the SET QUALIFY command.

When the Dynamic Debug facility is deactivated (by entering the SET DYNDEBUG OFF command) or SVC screening is disabled 2, Debug Tool 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 244

%LPR\textit{n} or %LPRH\textit{n} (%LPRH\textit{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\textit{n} or %LPRH\textit{n} variable.

%LPR\textit{n} cannot be used as the target of an assignment while debugging Enterprise PL/I programs.

Usage note

For assembler and disassembly, the LIST %LPR\textit{n} command displays values in hexadecimal, but for the LIST %LPRH\textit{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 171
“Assignment command (PL/I)” on page 38
“Assignment command (assembler and disassembly)” on page 35

%LPRB\textit{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.

---

2. See the [Debug Tool Customization Guide](#) for instructions on how to control SVC screening.
To modify one of these registers, assign a value to the associated %LPRB<sub>n</sub> 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 35

---

**%LPRD<sub>n</sub> (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<sub>n</sub> 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 35

---

**%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 290 for more information.

%PSW (assembler and disassembly only)
Contains the current Program Status Word.

%RC
Contains a return code whenever a Debug Tool command ends.

%RC initially has a value of zero unless the log file cannot be opened, in which case it has a value of −1.

Note: The %RC return code is a Debug Tool variable. It is not related to the return code that can be found in Register 15.

%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 Debug Tool. 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 Debug Tool variables in different languages

The table below shows the attributes for Debug Tool variables when used with different programming languages.

<table>
<thead>
<tr>
<th>Debug Tool variable</th>
<th>C and C++ attributes</th>
<th>COBOL attributes</th>
<th>PL/I attributes</th>
<th>Assembler/disassembly attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>%ADDRESS</td>
<td>void *</td>
<td>USAGE POINTER</td>
<td>POINTER</td>
<td>a</td>
</tr>
<tr>
<td>%AMODE</td>
<td>signed short int</td>
<td>PICTURE S9(4)</td>
<td>FIXED</td>
<td>H</td>
</tr>
<tr>
<td>%BLOCK</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
<td>CHARACTER(j)</td>
<td>CLj</td>
</tr>
<tr>
<td>%CAAADDRESS</td>
<td>void *</td>
<td>USAGE POINTER</td>
<td>POINTER</td>
<td>a</td>
</tr>
<tr>
<td>%CC</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>h</td>
</tr>
<tr>
<td>%CONDITION</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
<td>CHARACTER(j)</td>
<td>CLj</td>
</tr>
<tr>
<td>%COUNTRY</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
<td>CHARACTER(j)</td>
<td>CLj</td>
</tr>
<tr>
<td>%CU</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
<td>CHARACTER(j)</td>
<td>CLj</td>
</tr>
<tr>
<td>Debug Tool variable</td>
<td>C and C++ attributes</td>
<td>COBOL attributes</td>
<td>PL/I attributes</td>
<td>Assembler/ disassembly attributes</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------</td>
<td>------------------</td>
<td>-----------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>%EPA</td>
<td>void *</td>
<td>USAGE POINTER</td>
<td>POINTER</td>
<td>A</td>
</tr>
<tr>
<td>%EPRn</td>
<td>long double</td>
<td>n/a</td>
<td>FLOAT</td>
<td>L</td>
</tr>
<tr>
<td>%EPRBn</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>LB</td>
</tr>
<tr>
<td>%EPRDn</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>LD</td>
</tr>
<tr>
<td>%EPRHn</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>LH</td>
</tr>
<tr>
<td>%FPRn</td>
<td>float</td>
<td>USAGE COMP-1</td>
<td>FLOAT</td>
<td>E</td>
</tr>
<tr>
<td>%FPRBn</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>EB</td>
</tr>
<tr>
<td>%FPRDn</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>ED</td>
</tr>
<tr>
<td>%FPRHn</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>EH</td>
</tr>
<tr>
<td>%GPRn</td>
<td>signed int</td>
<td>PICTURE S9(9)</td>
<td>FIXED</td>
<td>F</td>
</tr>
<tr>
<td>%GPRGn</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>FD</td>
</tr>
<tr>
<td>%HARDWARE</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
<td>CHARACTER(j)</td>
<td>CLj</td>
</tr>
<tr>
<td>%LINE or STATEMENT</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
<td>CHARACTER(j)</td>
<td>CLj</td>
</tr>
<tr>
<td>%LOAD</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
<td>CHARACTER(j)</td>
<td>CLj</td>
</tr>
<tr>
<td>%LPRn</td>
<td>double</td>
<td>USAGE COMP-2</td>
<td>FLOAT</td>
<td>D</td>
</tr>
<tr>
<td>%LPRBn</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>DB</td>
</tr>
<tr>
<td>%LPRDn</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>DD</td>
</tr>
<tr>
<td>%LPRHn</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>DH</td>
</tr>
<tr>
<td>%NLANGUAGE</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
<td>CHARACTER(j)</td>
<td>CLj</td>
</tr>
<tr>
<td>%PATHCODE</td>
<td>signed short int</td>
<td>PICTURE S9(4)</td>
<td>FIXED</td>
<td>H</td>
</tr>
<tr>
<td>%PLANGUAGE</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
<td>CHARACTER(j)</td>
<td>CLj</td>
</tr>
<tr>
<td>%PROGMASK</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>H</td>
</tr>
<tr>
<td>%PROGRAM</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
<td>CHARACTER(j)</td>
<td>CLj</td>
</tr>
<tr>
<td>%PSW</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>CL8</td>
</tr>
<tr>
<td>%RC</td>
<td>signed short int</td>
<td>PICTURE S9(4)</td>
<td>FIXED</td>
<td>H</td>
</tr>
<tr>
<td>%RSTDSETS</td>
<td>signed int</td>
<td>PICTURE S9(9)</td>
<td>FIXED</td>
<td>F</td>
</tr>
<tr>
<td>%RUNMODE</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
<td>CHARACTER(j)</td>
<td>CLj</td>
</tr>
<tr>
<td>%Rn</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>F</td>
</tr>
<tr>
<td>%SUBSYSTEM</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
<td>CHARACTER(j)</td>
<td>CLj</td>
</tr>
<tr>
<td>%SYSTEM</td>
<td>unsigned char[]</td>
<td>PICTURE X(j)</td>
<td>CHARACTER(j)</td>
<td>CLj</td>
</tr>
</tbody>
</table>
Chapter 8. Debug Tool messages

All messages are shown in this section are in mixed case English. The uppercase English message text is the same, but is in uppercase letters.

Each message has a number of the form EQAnnnnx, where EQA indicates that the message is an Debug Tool 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 Debug Tool attempted to fix.
E  An error message describes an error that Debug Tool detected or cannot fix.
S  A severe error message describes an error that indicates a command referring to bad data, control blocks, program structure, or something similar.
U  An unrecoverable error message describes an error that prevents Debug Tool from continuing.

Symbols in messages

Many of the Debug Tool messages contain information that is inserted by the system when the message is issued. In this publication, such inserted information is indicated by italicized symbols, as in the following:

EQA1046I The breakpoint-id breakpoint is replaced.

The portion of Debug Tool located on the host notifies you of errors associated with debugging functions carried out by the host.

Refer to the following 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 172

© Copyright IBM Corp. 1992, 2009
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: This 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 partially parsed command is:
Explanation: The explanation of a command was requested or a command was determined to be in error.

EQA1017I The next word can be one of:
Explanation: This title line will be followed by message 1015.

EQA1018I Enter Y for YES and N for NO

EQA1019I The breakpoint-id breakpoint is replaced.
Explanation: This alerts the user to the fact that a previous breakpoint action existed and was replaced.

EQA1020I 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.
### Explanation:

#### EQA1051I

The (deferred) breakpoint-id breakpoint action is:

**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:

**EQA1053I**

The (disabled) breakpoint-id breakpoint action is:

**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:

**EQA1055I**

The (disabled and deferred) breakpoint-id breakpoint action is:

**EQA1056I**

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:

**EQA1057I**

AT stmt-number can be risky because the code for that statement might have been merged with that of another statement.

**EQA1058I**

RUNTO is active at statement_id.

**EQA1059I**

The Entry breakpoint command for Load_Module_Name => CU_Name has been deferred until the CU appears.

---

### Chapter 8. Debug Tool messages

303
**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.

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 dsname 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 such 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 Debug Tool.

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.

EQA1135I  
***** PREVIOUS STATEMENT *****  
**Explanation:** Automonitor Previous Statement area in the Monitor window.

EQA1136I  
**character**  
**Explanation:** This message is used to produce output for **LIST** ( ... ).

EQA1137I  
**expression name = expression value**  
**Explanation:** This message is used to produce output for **LIST TITLED** ( ... ) when an expression is a scalar.

EQA1138I  
**expression element**  
**Explanation:** This insert is used for naming the expression for expression element.

EQA1139I  
>>> 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 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 Procedures satisfied the pattern in a LIST NAMES CUS command.

EQA1160I  The following Procedures have pattern pattern:
Explanation: This title line precedes a list of Procedure names for a LIST NAMES PROCEDURES command.

EQA1161I  There are no Procedures with pattern pattern.
Explanation: This line appears when no Procedures satisfied the pattern in 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 block name.
Explanation: The LIST NAMES command found named variables in the named block but none of the names satisfied the pattern.

EQA1165I  There are no session names with pattern pattern.
Explanation: The LIST NAMES command found named variables that had been declared in the session but none of the names satisfied the pattern.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA1166I</td>
<td>There are no known session procedures.</td>
<td></td>
</tr>
<tr>
<td>EQA1167I</td>
<td>register name = register value</td>
<td></td>
</tr>
<tr>
<td>EQA1168I</td>
<td>No LIST STORAGE data is available for the requested reference or address.</td>
<td></td>
</tr>
<tr>
<td>EQA1169I</td>
<td>No frequency data is available</td>
<td></td>
</tr>
<tr>
<td>EQA1170I</td>
<td>Frequency of verb executions in cu_name</td>
<td></td>
</tr>
<tr>
<td>EQA1171I</td>
<td>character string = count</td>
<td></td>
</tr>
<tr>
<td>EQA1172I</td>
<td>TOTAL VERBS= total statements, TOTAL VERBS EXECUTED= total statements executed, PERCENT EXECUTED= percent executed</td>
<td></td>
</tr>
<tr>
<td>EQA1173I</td>
<td>(history number) ENTRY hook for cu_name</td>
<td></td>
</tr>
<tr>
<td>EQA1174I</td>
<td>(history number) ENTRY hook for block block name in cu_name</td>
<td></td>
</tr>
<tr>
<td>EQA1175I</td>
<td>(history number) EXIT hook for cu_name</td>
<td></td>
</tr>
<tr>
<td>EQA1176I</td>
<td>(history number) EXIT hook for block block name in cu_name</td>
<td></td>
</tr>
<tr>
<td>EQA1177I</td>
<td>(history number) STATEMENT hook at statement cu_name =&gt; statement_id</td>
<td></td>
</tr>
<tr>
<td>EQA1178I</td>
<td>(history number) PATH hook at statement cu_name =&gt; statement_id</td>
<td></td>
</tr>
<tr>
<td>EQA1179I</td>
<td>(history number) Before CALL hook at statement cu_name =&gt; statement_id</td>
<td></td>
</tr>
<tr>
<td>EQA1180I</td>
<td>(history number) CALL CEETEST at statement cu_name =&gt; statement_id</td>
<td></td>
</tr>
<tr>
<td>EQA1181I</td>
<td>(history number) WAITING for program input from ddname</td>
<td></td>
</tr>
<tr>
<td>EQA1182I</td>
<td>(history number) LOAD occurred at statement cu_name =&gt; statement_id</td>
<td></td>
</tr>
<tr>
<td>EQA1183I</td>
<td>(history number) DELETE occurred at statement cu_name =&gt; statement_id</td>
<td></td>
</tr>
<tr>
<td>EQA1184I</td>
<td>(history number) condition name raised at statement cu_name =&gt; statement_id</td>
<td></td>
</tr>
<tr>
<td>EQA1185I</td>
<td>(history number) LABEL hook at statement cu_name =&gt; statement_id</td>
<td></td>
</tr>
<tr>
<td>EQA1186I</td>
<td>Unable to display value of variable name. Use LIST (variable name) for further details</td>
<td></td>
</tr>
<tr>
<td>EQA1187I</td>
<td>There are no CUs compiled with debug data. To see the CU names, issue SET DISASSEMBLY ON before LIST command.</td>
<td></td>
</tr>
<tr>
<td>EQA1188I</td>
<td>(history number) DATE hook at statement cu_name =&gt; statement_id</td>
<td></td>
</tr>
<tr>
<td>EQA1189I</td>
<td>(history number) STATEMENT hook at statement cu_name =&gt; statement_id</td>
<td></td>
</tr>
<tr>
<td>EQA1190I</td>
<td>Unable to update the requested location.</td>
<td></td>
</tr>
</tbody>
</table>
EQA1191I The length of target and source must be equal.
Explanation: The number of bytes to be altered must be equal the length of source.

EQA1192I The number of bytes to be altered is too large.
Explanation: A maximum of 8 bytes of storage can be change when source is a hexadecimal number, and 4 bytes when source is integer number.

EQA1193I There are no variables in section_name.
Explanation: The LIST TITLED command found no variables that had been declared in the section.

EQA1194I The following variables are known in section section_name.
Explanation: This is a title line that is the result of a LIST TITLED command. It precedes the list of all variables contained within a section.

EQA1226I The EQUATE named EQUATE name is replaced.
Explanation: This alerts the user to the fact that a previous EQUATE existed and was replaced.
Programmer response: Verify that this was intended.

EQA1227I The following EQUATE definitions are in effect:
Explanation: This is the header for the QUERY EQUATES command.

EQA1228I EQUATE identifier = "EQUATE string"
Explanation: Used to display EQUATE identifiers and their associated strings. The string is enclosed in quotation marks so that any leading or trailing blanks are noticeable.

EQA1229I The program is currently exiting block block name.
Explanation: Shows the bearings in an interrupted program.

EQA1230I The program is currently 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.
Chapter 8. Debug Tool messages

**EQA1247I** COLOR color highlight 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 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.

**EQA1255I** The CU specified for the LOADDEBUGDATA command is not a disassembly CU.  
Explanation: Only a disassembly CU can be identified as assembler CU.

**EQA1256I** 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.

**EQA1257I** 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 valid LIST command.

**EQA1258I** 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 Debug Tool. 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 non-Language Environment 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: Debug Tool 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 Debug Tool 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 Debug Tool 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: Debug Tool 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: Debug Tool 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 Debug Tool.

- Truncated keywords are no longer truncated.
- Lowercase to uppercase conversion was done where appropriate.
- Only a single command is contained in a record. If multiple commands are involved, additional records are prepared using this format.

EQA1279I  
TEST (cu_name => statement_id):

Explanation: Debug Tool is ready to accept a command from the terminal. This message is used in line mode when an initial prompt occurs at a statement and a statement table is available.

Programmer response: Enter a command. If you are not sure what you can enter, enter HELP or ?. Information is displayed identifying the commands you are allowed to enter.

EQA1280I  
TEST (cu_name ENTRY):

Explanation: Debug Tool is ready to accept a command from the terminal. This message is used in line mode when an initial prompt occurs at a compile unit entry.

Programmer response: Enter a command. If you are not sure what you can enter, enter HELP or ?. Information will be displayed identifying the commands you are allowed to enter.

EQA1281I  
TEST (cu_name block name ENTRY):

Explanation: Debug Tool is ready to accept a command from the terminal. This message is used in line mode when an initial prompt occurs at a block entry.

Programmer response: Enter a command. If you are not sure what you can enter, enter HELP or ?. Information will be displayed identifying the commands you are allowed to enter.

EQA1282I  
TEST (cu_name EXIT):

Explanation: Debug Tool is ready to accept a command from the terminal. This message is used in line mode when an initial prompt occurs at a compile unit exit.

Programmer response: Enter a command. If you are not sure what you can enter, enter HELP or ?. Information will be displayed identifying the commands you are allowed to enter.

EQA1283I  
TEST (cu_name block name EXIT):

Explanation: Debug Tool is ready to accept a command from the terminal. This message is used in line mode when an initial prompt occurs at a block exit.

Programmer response: Enter a command. If you are not sure what you can enter, enter HELP or ?. Information will be displayed identifying the commands you are allowed to enter.

EQA1284I  
TEST (Application program has terminated):

Explanation: Debug Tool is ready to accept a command from the terminal. This message is used in line mode when an initial prompt occurs at 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:** Debug Tool is ready to accept a command from the terminal. This message is used in line mode when an initial prompt occurs at a label.

**Programmer response:** Enter a command. If you are not sure what you can enter, enter HELP or ?. Information is displayed identifying the commands you are allowed to enter.

EQA1286I (Application program has terminated)

**Explanation:** Debug Tool is ready to accept a command from the terminal. This message is used in full-screen mode when an initial prompt occurs at the termination of the application program.

EQA1287I Unknown

**Explanation:** Debug Tool is ready to accept a command from the terminal. This message is used in full-screen mode when an initial prompt occurs and the location is unknown.

EQA1288I initialization

**Explanation:** Debug Tool is ready to accept a command from the terminal. This message is used in full-screen mode when an initial prompt occurs after Debug Tool initialization and before any program hooks are reached.

EQA1289I ddname: program output

**Explanation:** Displays program output with the ddname preceding the output.

EQA1290I The program is waiting for input from ddname

**Explanation:** Debug Tool 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:** Prompts user for intercepted input of variable-formatted file.

EQA1293I TEST (cu-name);

**Explanation:** Debug Tool is ready to accept a command from the terminal. This message is used in linemode when an initial prompt occurs at a statement and a statement table is not available.

**Programmer response:** Enter a command. If you are not sure of what you can enter, enter HELP or ?. Information is displayed identifying the available commands you are allowed to enter.

EQA1294I The suspended LOCAL MONITOR commands are:

**Explanation:** This is the header produced by LIST MONITOR when suspended local monitors are present.

EQA1301I A ABEND code ABEND occurred.

**Explanation:** A system or user ABEND has been detected by Debug Tool.

EQA1306I You were prompted because the CONDITION name condition was raised in your program.

**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 Debug Tool was given control.

EQA1308I You were prompted because a condition was raised in your program.

**Explanation:** The program stopped running due to the occurrence of a condition whose name is unknown.

EQA1309I CONDITION name is a severity or class SEVERITY code condition.

**Explanation:** The condition named is described by its severity level or class code. See the z/OS Language Environment Programming Guide.

EQA1316I Block block name contains the following statements:

**Explanation:** This message precedes the message that identifies all statement numbers in the block.

EQA1317I block level space_characters block name

**Explanation:** This message is used instead of EQA1097I when the number of block levels is greater than the indentation allowed.

EQA1318I space_characters INCLUDE_file_name index_number

**Explanation:** The first insert controls the indentation, the second is the INCLUDE file name, and the third insert is the source index block level.

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.

EQA1328I  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.

EQA1329I  Its address is address and its length is length
Explanation: Text of a DESCRIBE ATTRIBUTES message for PL/I.

EQA1330I  Its offset is offset and its length is length
Explanation: Text of a DESCRIBE ATTRIBUTES message for PL/I.

EQA1331I  Its length is length
Explanation: Text of a DESCRIBE ATTRIBUTES message for PL/I.

EQA1332I  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.

EQA1333I  The string was found.
Explanation: FIND was successful in locating the target string. The line on which the string was found is displayed just above this message when operating in line mode.

EQA1334I  The operating system has generated the following message:
Explanation: The Operating System can issue its own messages. These are relayed to the user.

EQA1335I  OS message
Explanation: The operating system can issue its own messages. These are relayed to the user.

EQA1336I  IBM Debug Tool Version 10 Release 1 Mod 0
            time stamp 5655-V50: Copyright IBM Corp. 1992, 2009
Explanation: Text of a DESCRIBE ATTRIBUTES message for PL/I.

EQA1337I  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.

EQA1338I  The RETRIEVE queue is empty.
Explanation: There are no entries in the retrieve queue.

EQA1339I  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.

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.
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.

Playback statement sequence error. PLAYBACK found stmt was found before PLAYBACK req stmt. Statement ignored.

Explanation: Playback was not the proper state to process the specified statement. The statement is ignored.

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.

Return code RC from from PLAYBACK run-time API name. PLAYBACK disabled.

Explanation: The specified return code indicated an error condition detected by the Playback run-time API. Playback is disabled.

Programmer response: Return code 63 indicates not enough memory was available for Playback. Restart your Debug Tool session and enter the PLAYBACK ENABLE command using the integer option. For example: PLAYBACK ENABLE * 10000

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.

PLAYBACK START is not active. You are not in Playback mode.

Explanation: This message is issued in response to the QUERY PLAYBACK LOCATION command when Playback replay is not active.

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.

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.

Command is not allowed in Playback replay mode.

Explanation: The previous command was not supported after playback start.

Command is not allowed in Playback NODATA replay mode.

Explanation: The previous command was ignored because of errors that were indicated by previous messages.

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.

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.

Playback replay has reached the last record in the PLAYBACK buffer. You must enter PLAYBACK BACKWARD or PLAYBACK STOP.

Explanation: Playback replay has reached the last record in the buffer. You cannot move farther forward.

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.

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.

EQA1367I  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.

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:  Debug Tool 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:  Debug Tool detected a storage violation.

EQA1379I  No MEMORY data is available for the requested reference or address.
Explanation:  The given reference or address is invalid.

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.

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.
<table>
<thead>
<tr>
<th>Code</th>
<th>Message Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA1402E</td>
<td>Each window must have unique letters of L, M, S, and E.</td>
<td>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.</td>
</tr>
<tr>
<td>EQA1403E</td>
<td>Invalid prefix command was entered.</td>
<td>The user entered an invalid prefix command.</td>
</tr>
<tr>
<td>EQA1404E</td>
<td>Search target not found.</td>
<td>The target for the search command was not found.</td>
</tr>
<tr>
<td>EQA1405E</td>
<td>No previous search arguments exist; find not performed.</td>
<td>A FIND command was issued without an argument. Since the FIND command had not been issued previously, Debug Tool had nothing to search for.</td>
</tr>
<tr>
<td>EQA1406E</td>
<td>Invalid window ID.</td>
<td>The window header field contains an invalid window ID. Valid window IDs are SOURCE, MONITOR, and LOG.</td>
</tr>
<tr>
<td>EQA1407E</td>
<td>Invalid scroll amount entered.</td>
<td>Scroll field contains an invalid scroll amount.</td>
</tr>
<tr>
<td>EQA1408E</td>
<td>Duplicate window ID</td>
<td>More than one window header field contains the same window id.</td>
</tr>
<tr>
<td>EQA1409E</td>
<td>No line, statement or offset breakpoints were found.</td>
<td>No line, statement or offset breakpoints were found of the specified (or defaulted) status.</td>
</tr>
<tr>
<td>EQA1410E</td>
<td>Variable name is a LABEL on a modifiable instruction. No AT commands can be issued against it.</td>
<td>The specified label is on an instruction that is modified at some point in the program. Breakpoints cannot be set on such an instruction.</td>
</tr>
<tr>
<td>EQA1411E</td>
<td>Invalid operand number was entered.</td>
<td>The user entered a number corresponding to an invalid operand.</td>
</tr>
<tr>
<td>EQA1412E</td>
<td>Prefix command not supported for current programming language.</td>
<td>Prefix command not supported for current programming language.</td>
</tr>
<tr>
<td>EQA1413E</td>
<td>There are no operands in the statement to display.</td>
<td>The specified prefix command is on a statement with no operands.</td>
</tr>
<tr>
<td>EQA1414E</td>
<td>Prefix command was entered on invalid statement.</td>
<td>The user entered prefix command on an invalid line.</td>
</tr>
<tr>
<td>EQA1415E</td>
<td>Specified operand number is too big.</td>
<td>The user entered an operand number that exceeds the number of operands in that statement.</td>
</tr>
<tr>
<td>EQA1416E</td>
<td>Multiple invalid prefix commands. For details, reenter commands individually.</td>
<td>More than one invalid prefix commands. To see details you must reenter the invalid command one at a time.</td>
</tr>
<tr>
<td>EQA1417E</td>
<td>Invalid line for prefix command. Line must be in the active block.</td>
<td>The user entered a prefix command on a line that is contained in a block that is not currently active.</td>
</tr>
<tr>
<td>EQA1418E</td>
<td>One or more of multiple updates were not processed.</td>
<td>One or more invalid updates. The commands were not processed. For details, please see messages in the LOG.</td>
</tr>
<tr>
<td>EQA1430W</td>
<td>The EQUATE named EQUATE name was has not been established.</td>
<td>CLEAR EQUATE &lt;name&gt; was attempted for an EQUATE name that has not been established.</td>
</tr>
<tr>
<td>EQA1431W</td>
<td>There are no EQUATE definitions in effect.</td>
<td>CLEAR EQUATE or QUERY EQUATES was issued but there are no EQUATE definitions.</td>
</tr>
<tr>
<td>EQA1432E</td>
<td>function is not supported.</td>
<td>Language/Country is not supported.</td>
</tr>
<tr>
<td>EQA1433E</td>
<td>Switching to the programming language language-name is invalid because there are no language-name compilation units in the initial load module.</td>
<td>A SET PROGRAMMING LANGUAGE command was issued, but the initial load module contains no compilation units compiled in the language specified (or implied).</td>
</tr>
</tbody>
</table>
### Debug Tool V10 Reference and Messages

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA1434E</td>
<td>Error in setting debug name to ??????????.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>Refer to the maximum number of CUs allowed for debugging.</td>
</tr>
<tr>
<td>EQA1435E</td>
<td>Error in setting name.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>This is a generic message for SET command errors.</td>
</tr>
<tr>
<td>EQA1436W</td>
<td>SET EXECUTE is OFF -- command will not be executed.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The command was parsed but not executed.</td>
</tr>
<tr>
<td>EQA1437W</td>
<td>SET DYNDEBUG can not be executed at this time. SET DYNDEBUG can only be executed at the beginning</td>
</tr>
<tr>
<td>Explanation:</td>
<td>of a debugging session, before any STEP or GO commands. The DYNDEBUG status has not been changed.</td>
</tr>
<tr>
<td>EQA1438W</td>
<td>SET DYNDEBUG can not be executed at this time. SET DYNDEBUG can only be executed at the beginning</td>
</tr>
<tr>
<td>Explanation:</td>
<td>of a debugging session, before any STEP or GO commands. The DYNDEBUG status has not been changed.</td>
</tr>
<tr>
<td>EQA1439E</td>
<td>This CU is not AUTOMONITOR capable for expressions.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The CU is not AUTOMONITOR capable.</td>
</tr>
<tr>
<td>Programmer response:</td>
<td>Refer to the description of the SET AUTOMONITOR command in the Debug Tool Reference and Messages document to determine the requirements this CU must fulfill in order to use the SET AUTOMONITOR command.</td>
</tr>
<tr>
<td>EQA1440E</td>
<td>SET AUTOMONITOR ON is not valid for this CU. Use the SET AUTOMONITOR ON LOG command to activate the statement trace function.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The current CU is not AUTOMONITOR capable. SET AUTOMONITOR ON LOG will activate the statement trace.</td>
</tr>
<tr>
<td>Programmer response:</td>
<td>Refer to the description of the SET AUTOMONITOR command in the Debug Tool Reference and Messages document to determine the requirements this CU must fulfill in order to use the SET AUTOMONITOR command.</td>
</tr>
<tr>
<td>EQA1441I</td>
<td>The statement trace is now active. Use the SET AUTOMONITOR OFF command to deactivate the statement trace.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The statement trace is active for a CU that is not AUTOMONITOR capable.</td>
</tr>
<tr>
<td>EQA1442E</td>
<td>DYNDEBUG cannot be SET OFF when running without the Language Environment run-time.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The Dynamic Debug facility cannot be deactivated while running without the Language Environment run-time.</td>
</tr>
<tr>
<td>EQA1443I</td>
<td>There are no INTERCEPT specifications in effect.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>QUERY INTERCEPT was issued but there are no INTERCEPT specifications active.</td>
</tr>
<tr>
<td>EQA1444I</td>
<td>For this command to be effective, SETTINGS must be saved with SET RESTORE SETTINGS AUTO in effect.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>In order for SET RESTORE BPS AUTO or SET RESTORE MONITORS AUTO to be effective, the settings must be automatically restored when Debug Tool is started. You entered one of these commands but the current setting is SET RESTORE SETTINGS NOAUTO.</td>
</tr>
<tr>
<td>EQA1445E</td>
<td>The command is not supported with PL/I. Use PL/I built-in function HEX to obtain hexadecimal values.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>%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).</td>
</tr>
<tr>
<td>EQA1450E</td>
<td>Unable to display the result from expression evaluation</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The entire result from the expression evaluation cannot be displayed; for example, the array is too large.</td>
</tr>
<tr>
<td>EQA1451E</td>
<td>operand contains incompatible data type.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>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 Debug Tool IF command, and “Allowable moves for the MOVE command (COBOL)” on page 172 for allowable moves for the Debug Tool MOVE command.</td>
</tr>
<tr>
<td>EQA1452E</td>
<td>argument name is not a valid argument.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The specified argument is not valid.</td>
</tr>
<tr>
<td>EQA1453E</td>
<td>The number of arguments is not correct.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>There are either too many or too few arguments specified.</td>
</tr>
<tr>
<td>EQA1454E</td>
<td>operand name is not a valid operand.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The specified operand is undefined or is an invalid literal.</td>
</tr>
<tr>
<td>EQA1455E</td>
<td>An unsupported operator/operand is specified.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>An operator or an operand was not understood, and therefore was not processed. Examples of when this message is issued when using COBOL include:</td>
</tr>
<tr>
<td></td>
<td>• An attempt to perform arithmetic with a nonnumeric data item</td>
</tr>
</tbody>
</table>
• An attempt to perform arithmetic with a windowed date field or a year-last date field.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA1456S</td>
<td>The variable variable name is undefined or is incorrectly qualified.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The named variable could not be located or undefined.</td>
</tr>
<tr>
<td>Programmer response:</td>
<td>You need to qualify to a different block in order to locate the variable.</td>
</tr>
<tr>
<td>EQA1457E</td>
<td>The exponent exponent contains a decimal point.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>This feature is not supported. No decimal point is allowed in exponent specification.</td>
</tr>
<tr>
<td>EQA1458E</td>
<td>The address of data item has been determined to be invalid.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>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.</td>
</tr>
<tr>
<td>EQA1459E</td>
<td>Literal string is not a valid literal.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The combination of characters specified for the literal is not a valid literal.</td>
</tr>
<tr>
<td>EQA1460E</td>
<td>Operand operand name should be numeric.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>A nonnumeric operand was found where a numeric operand was expected.</td>
</tr>
<tr>
<td>EQA1461E</td>
<td>Invalid data for data item is found.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>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.</td>
</tr>
<tr>
<td>EQA1462E</td>
<td>Invalid sign for data item is found.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The sign position of a signed data item contains an invalid sign. The item might not have been initialized.</td>
</tr>
<tr>
<td>EQA1463E</td>
<td>A divisor of 0 is detected in a divide operation.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The expression contains a divide operation where the divisor was determined to be zero.</td>
</tr>
<tr>
<td>EQA1464E</td>
<td>Data item is used as a receiver but it is not a data name.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The target of an assignment is not valid.</td>
</tr>
<tr>
<td>EQA1465E</td>
<td>The TGT for a program is not available.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The program might have been deleted or canceled.</td>
</tr>
</tbody>
</table>
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:  The command could not be completed due to inadequate resources.
Programmer response:  Increase the region size and restart Debug Tool.

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 Debug Tool, 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 Debug Tool 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 name 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 +.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA1536E</td>
<td>The operand of the address operator must be a function designator or an lvalue that is not a bitfield.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>This message applies to the C unary operator &amp; (address).</td>
</tr>
<tr>
<td>EQA1537E</td>
<td>Invalid constant for the C language.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>This message applies to C constants.</td>
</tr>
<tr>
<td>EQA1538E</td>
<td>Argument argument in function call function is incompatible with the function definition. Since Warning is on, the function call is not made.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>This message applies to C function calls. The argument must have a type that would be valid in an assignment to the parameter.</td>
</tr>
<tr>
<td>EQA1539E</td>
<td>The binary operator operator requires integer operands.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>This message applies to the C binary operator % (remainder), &lt;&lt; (bitwise left shift), &gt;&gt; (bitwise right shift), &amp; (bitwise and), ?? (bitwise exclusive or),</td>
</tr>
<tr>
<td></td>
<td>(bitwise inclusive or), and the corresponding assignment operators (for example, %&lt;, and &lt;&lt;=).</td>
</tr>
<tr>
<td>EQA1540E</td>
<td>The binary operator operator requires a modifiable lvalue for its first operand.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>This message applies to the C binary assignment operators.</td>
</tr>
<tr>
<td>EQA1541E</td>
<td>The binary operator operator requires arithmetic operands.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>This message applies to the C binary operators * and /.</td>
</tr>
<tr>
<td>EQA1542E</td>
<td>Source in assignment to an enum is not a member of the enum. Since Warning is on, the operation is not performed.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>This message applies to C. You attempted to assign a value to enum, but the value is not legitimate for that enum.</td>
</tr>
<tr>
<td>EQA1543E</td>
<td>Invalid value for the shift operator operator. Since Warning is on, the operation will not be performed.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>This message applies to the C binary operators &lt;&lt; (bitwise left shift) and &gt;&gt; (bitwise right shift). Shift values must be nonnegative and less than 33. These tests are made only when WARNING is on.</td>
</tr>
<tr>
<td>EQA1544E</td>
<td>Array subscript is negative. Since Warning is on, the operation is not performed.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>This message applies to the C subscripts.</td>
</tr>
<tr>
<td>EQA1555E</td>
<td>name is a tag name. This cannot be listed since it has no storage associated with it.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>This message applies to C tag names.</td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EQA1557E</td>
<td>name is not an lvalue. This cannot be listed since it has no storage associated with it.</td>
</tr>
<tr>
<td>EQA1558E</td>
<td>name has storage class void, not permitted on the LIST command.</td>
</tr>
<tr>
<td>EQA1559E</td>
<td>The second operand of the %RECURSION operator must be arithmetic.</td>
</tr>
<tr>
<td>EQA1560E</td>
<td>The second operand of the %RECURSION operator must be positive.</td>
</tr>
<tr>
<td>EQA1561E</td>
<td>The first operand of the %RECURSION operator must be a parameter or an automatic variable.</td>
</tr>
<tr>
<td>EQA1562E</td>
<td>The first operand of the %INSTANCE operator must be a parameter or an automatic variable.</td>
</tr>
<tr>
<td>EQA1563E</td>
<td>Generation specified for %RECURSION is too large.</td>
</tr>
<tr>
<td>EQA1564E</td>
<td>The identifier identifier has been replaced.</td>
</tr>
<tr>
<td>EQA1565E</td>
<td>The declaration is too large</td>
</tr>
<tr>
<td>EQA1566E</td>
<td>An attempt to modify a constant was made. Since Warning is on, the operation is not performed.</td>
</tr>
<tr>
<td>EQA1567E</td>
<td>An attempt to take the address of a variable with register storage was made. Since Warning is on, the operation is not performed.</td>
</tr>
<tr>
<td>EQA1568E</td>
<td>Type of expression to %DUMP must be a literal string.</td>
</tr>
<tr>
<td>EQA1569E</td>
<td>Octal constant is too long.</td>
</tr>
<tr>
<td>EQA1570E</td>
<td>Octal constant is too big.</td>
</tr>
<tr>
<td>EQA1571E</td>
<td>Hex constant is too long.</td>
</tr>
<tr>
<td>EQA1572E</td>
<td>Decimal constant is too long.</td>
</tr>
<tr>
<td>EQA1573E</td>
<td>Decimal constant is too big.</td>
</tr>
<tr>
<td>EQA1574E</td>
<td>Float constant is too long.</td>
</tr>
<tr>
<td>EQA1575E</td>
<td>Float constant is too big.</td>
</tr>
<tr>
<td>EQA1576E</td>
<td>The environment is not yet fully initialized.</td>
</tr>
<tr>
<td>EQA1577E</td>
<td>Size of the aggregate is too large</td>
</tr>
<tr>
<td>EQA1578E</td>
<td>Only &quot;=&quot; and &quot;¬=&quot; are allowed as operators in comparisons involving program control data.</td>
</tr>
<tr>
<td>EQA1579E</td>
<td>Program control data may be compared only with program control data of the same type.</td>
</tr>
<tr>
<td>EQA1580E</td>
<td>Area variables cannot be compared.</td>
</tr>
<tr>
<td>EQA1581E</td>
<td>Aggregates are not allowed in conditional expressions such as the expressions in IF ... THEN, WHILE (...), UNTIL (...), and WHEN (... ) clauses.</td>
</tr>
</tbody>
</table>
**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 uninitializor 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 uninitializor 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.
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.

Aggregate used in wrong context.

Explanation: This message applies to PL/I constants.

The second expression in the built-in function 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.

The second expression in the built-in function built-in function must not be an aggregate.

Explanation: Debug Tool does not support aggregates in this context.

The first argument in the built-in function built-in function must be an array expression.

Explanation: The named built-in function expects an array to be the first argument.

Argument number number in the built-in function 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.

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.

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.

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.

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.

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.

built-in function name built-in function 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.

COUNT( variable name ) must refer to an open STREAM file.

Explanation: You must name an open STREAM file in the COUNT built-in function.

LINENO( variable name ) must refer to an open PRINT file.

Explanation: You must name an open PRINT file in the LINENO built-in function.

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.

The argument in the built-in function 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.

Argument to POINTER is an aggregate when pointer is being used as a locator.

Explanation: This message applies to PL/I constants.

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)) > 1k383.

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.
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.

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.

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.

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.

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.

built-in function (x) is undefined if ABS(x) > 1.

Explanation: This applies to the PL/I ASIN and ACOS built-in functions.

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.

ATAN(z) is undefined if z is COMPLEX and z = +i or z = -i.

Explanation: This applies to the PL/I ATAN built-in function.

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.

SQRT(x) is undefined if x is REAL and x < 0.

Explanation: This applies to the PL/I SQRT built-in function.

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.

Built-in function(X,Y) is undefined if X=0 and Y=0

Explanation: This applies to PL/I constants.

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
  \[
  \text{ABS}(X) \leq (2^{18})\pi
  \]
  \[
  \text{ABS}(X) \leq (2^{17})\pi \text{ if } X \text{ is complex}
  \]

- TAN
  \[
  \text{ABS}(X) \leq (2^{18})\pi \text{ if } X \text{ is real}
  \]
  \[
  \text{ABS}(X) \leq (2^{17})\pi \text{ if } X \text{ is complex}
  \]

- COSH, EXP and SINH
  \[
  \text{ABS}(X) \leq (2^{17})\pi \text{ if } X \text{ is complex}
  \]

- COSD, SIND and TAND
  \[
  \text{ABS}(X) \leq (2^{18})\pi
  \]

For long floating-point arguments, the limits are:

- COS and SIN
  \[
  \text{ABS}(X) \leq (2^{50})\pi
  \]
  \[
  \text{ABS}(X) \leq (2^{49})\pi \text{ if } X \text{ is complex}
  \]

- TAN
  \[
  \text{ABS}(X) \leq (2^{50})\pi \text{ if } X \text{ is real}
  \]
  \[
  \text{ABS}(X) \leq (2^{49})\pi \text{ if } X \text{ is complex}
  \]

- COSH, EXP and SINH
  \[
  \text{ABS}(X) \leq (2^{50})\pi \text{ if } X \text{ is complex}
  \]

- COSD, SIND and TAND
  \[
  \text{ABS}(X) \leq (2^{50})\pi
  \]

For extended floating-point arguments, the limits are:

- COS and SIN
  \[
  \text{ABS}(X) \leq (2^{106})\pi
  \]
  \[
  \text{ABS}(X) \leq (2^{105})\pi \text{ if } X \text{ is complex}
  \]

- TAN
  \[
  \text{ABS}(X) \leq (2^{106})\pi \text{ if } X \text{ is real}
  \]
  \[
  \text{ABS}(X) \leq (2^{105})\pi \text{ if } X \text{ is complex}
  \]

- COSH, EXP and SINH
  \[
  \text{ABS}(X) \leq (2^{106})\pi \text{ if } X \text{ is complex}
  \]

- COSD, SIND and TAND
  \[
  \text{ABS}(X) \leq (2^{106})\pi
  \]

The subject of the SUBSTR pseudovariable (character string) is not a string.

Explanation: You are trying to get a substring from something other than a string.
EQA1632E  Argument to pseudovariable must be complex numeric
Explanation:  This message applies to PL/I constants.

EQA1633E  The first argument to a pseudovariable must refer to a variable, not an expression or a pseudovariable.
Explanation:  The arguments that accompany a pseudovariable are incorrect.

EQA1634E  The length of the bit string that would be returned by UNSPEC is greater than the maximum for a bit variable. Processing of the expression will stop.
Explanation:  This will occur in UNSPEC(A) where A is CHARACTER(n) and n > 4095, where A is CHARACTER(n) VARYING and n > 4093, where A is AREA(n) and n > 4080, etc.

EQA1635E  Maximum number of arguments to PLIDUMP subroutine is two
Explanation:  This message applies to PL/I constants.

EQA1636E  Invalid argument in CALL %DUMP
Explanation:  This message applies to PL/I constants.

EQA1637E  PL/I cannot process the expression expression name.
Explanation:  This applies to PL/I constants.

EQA1638E  Argument argument number to the MPSTR built-in function must not have the GRAPHIC attribute.
Explanation:  GRAPHIC (DBCS) strings are prohibited as arguments to the MPSTR built-in function.

EQA1639E  ALLOCATION( variable name ) is invalid because the ALLOCATION built-in function can be used only with controlled variables.
Explanation:  You must name a variable that is ALLOCATEable.
Programmer response:  The variable by that name cannot be a controlled variable within the current context. If the variable exists somewhere else (and is a controlled variable), you should use qualification with the variable name.

EQA1640E  variable name is an aggregate and hence is invalid as an argument to the POINTER built-in function when that built-in function is used as a locator.
Explanation:  The argument to the POINTER built-in function is invalid. The argument to the POINTER built-in function should be an OFFSET data type for the first argument, or an AREA data type for the second argument.

EQA1641E  Structures are not supported within this context.
Explanation:  Given dDCL 1 A, 2 B FIXED, 2 C FLOAT, the name A refers to a structure.
Programmer response:  Break the command into commands for each of the basic elements of the structure, or use the DECLARE command with a BASED variable to define a variable overlaying the structure.

EQA1642E  The first argument to the built-in function name built-in function must have POINTER type.
Explanation:  This applies to the POINTERADD built-in function. The first argument must have pointer type, and it must be possible to convert the other to FIXED BIN(31,0).

EQA1643E  The argument in the built-in function name built-in function must have data type: data type.
Explanation:  This message applies to various built-in functions. By built-in function, the datatypes required are:
ENTRYADDR  ENTRY
BINARYVALUE  POINTER
BINVALUE  POINTER

EQA1644W  STRINGRANGE is disabled and the SUBSTR arguments are such that STRINGRANGE ought to be raised. Debug Tool will revise the SUBSTR reference as if STRINGRANGE were enabled.
Explanation:  See the Language Reference built-in function chapter for the description of when STRINGRANGE is raised. See the Language Reference condition chapter for the values of the revised SUBSTR reference.

EQA1645E  The subject of the pseudovariable name pseudovariable must have data type: data type.
Explanation:  This message applies to various pseudovariables. By pseudovariable, the datatypes required are:
ENTRYADDR  ENTRY VARIABLE

EQA1646E  built-in function (z) is undefined if z is COMPLEX.
Explanation:  This applies to the PL/I ACOS, ASIN, ATAND, COSD, 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 Debug Tool Command Log for more detailed information.

EQA1650E The range of statements statement_id - statement_id is invalid because the two statements belong to different blocks.
Explanation: AT stmt1-stmt2 is valid only if stmt1 and stmt2 are in the same block.

EQA1651W The breakpoint-id breakpoint has not been established.
Explanation: You just issued a CLEAR/LIST command against a breakpoint that does not exist.
Programmer response: Verify that you referred to the breakpoint using the same syntax that was used to establish it. Perhaps a CLEAR command occurred since the command that established the breakpoint.

EQA1652E Since the program for the statement statement-number does not have hooks at statements, AT commands are rejected for all statements in the program.
Explanation: The program has not been prepared properly so AT commands are rejected for all statements in the program.
Programmer response: Make sure the program has been prepared properly by checking Part 2. Preparing your program for debugging and Appendix F, Syntax of the TEST Compiler option in the Debug Tool User's Guide. Also, the LIST LINE NUMBERS command can be used to list all statement or line numbers that are valid locations for an AT LINE or AT STATEMENT breakpoint.

EQA1653E A file name is invalid in this context.
Explanation: A command (for example, AT ENTRY) specified a C file name where a function or compound statement was expected.

EQA1654E Since the cu cu_name does not have hooks at block entries and exits, all AT ENTRY and AT EXIT commands will be rejected for the cu.
Explanation: A compile unit must have been compiled with TEST(BLOCK), TEST(PATH) or TEST(ALL) for hooks to be present at block entries and block exits.

EQA1655E Since the program for the label label-name does not have hooks at labels, AT commands are rejected for all labels in the program.
Explanation: A compilation unit must have been compiled with TEST(PATH) or TEST(ALL) for hooks to be present at labels.

EQA1656E statement_id contains a value that is invalid in this context.
Explanation: %STATEMENT and %LINE are invalid in AT commands at block entry and block exit, and in AT and LIST STATEMENT commands at locations that are outside of the program.

EQA1657W There are no breakpoint-class breakpoints set.
Explanation: The command CLEAR/LIST AT was entered but there are no AT breakpoints presently set, or the command CLEAR/LIST AT class was entered but there are no AT breakpoints presently set in that class.

EQA1658W There are no enabled breakpoint-class breakpoints set.
Explanation: The command CLEAR/LIST AT was entered but there are no enabled AT breakpoints presently set in the requested class of breakpoints.

EQA1659W There are no disabled breakpoint-class breakpoints set.
Explanation: The command CLEAR/LIST AT was entered but there are no disabled AT breakpoints presently set in the requested class of breakpoints.

EQA1660W The breakpoint-id breakpoint is not enabled.
Explanation: You issued a specific LIST AT ENABLED command against a breakpoint that is not enabled.

EQA1661W The breakpoint-id breakpoint is not disabled.
Explanation: You issued a specific LIST AT DISABLED command against a breakpoint that is not disabled.

EQA1662W The breakpoint-id breakpoint cannot be triggered because it is disabled.
Explanation: You cannot TRIGGER a disabled breakpoint.

EQA1663W There are no breakpoints set. No breakpoints are currently set.

EQA1664W There are no disabled breakpoints set.
Explanation: No disabled breakpoints are currently set.

EQA1665W There are no enabled breakpoints set.
Explanation: No enabled breakpoints are currently set.

EQA1666W The breakpoint-id breakpoint is already enabled.
Explanation: You cannot ENABLE an enabled breakpoint.

EQA1667W The breakpoint-id breakpoint is already disabled.
Explanation: You cannot DISABLE a disabled breakpoint.

EQA1668W The attempt to set this breakpoint has failed.
Explanation: For some reason, when Debug Tool tried to set this breakpoint, an error occurred. This breakpoint cannot be set.
EQA1669W  The FROM or EVERY value in a breakpoint command must not be greater than the specified TO value.
Explanation: In an every_clause specified with a breakpoint command, if the TO value was specified, the FROM or EVERY value must be less than or equal to the TO value.

EQA1670W  GO/RUN BYPASS is ignored. It is valid only when entered for an AT CALL, AT GLOBAL CALL, or AT OCCURRENCE.
Explanation: GO/RUN BYPASS is valid only when Debug Tool is entered for an AT CALL, AT GLOBAL CALL, or AT OCCURRENCE breakpoint.

EQA1671W  AT OCCURRENCE breakpoint or TRIGGER of condition condition-name cannot have a reference specified. This command not processed.
Explanation: The following AT OCCURRENCE conditions must have a qualifying reference: CONDITION, ENDFILE, KEY, NAME, PENDING, RECORD, TRANSMIT and UNDEFINEDFILE. This would also apply to the corresponding TRIGGER commands.

EQA1672W  AT OCCURRENCE breakpoint or TRIGGER of condition condition-name must have a valid reference specified. This command not processed.
Explanation: The following AT OCCURRENCE conditions must have a valid qualifying reference: CONDITION, ENDFILE, KEY, NAME, PENDING, RECORD, TRANSMIT and UNDEFINEDFILE. This would also apply to the corresponding TRIGGER commands.

EQA1673W  An attempt to automatically restore an AT breakpoint type breakpoint failed.
Explanation: Debug Tool was attempting to restore a breakpoint that had been set in the previous process and has failed in that attempt. There are two reasons this could have happened. If the Compile Unit (CU) has been changed (that is, modified and recompiled/linked) between one process and the next or a breakpoint had been established for a statement or variable that no longer exists due to the change, when Debug Tool attempts to reestablish that breakpoint, it will fail with this message.

EQA1674W  An attempt to automatically disable an AT breakpoint type breakpoint failed.
Explanation: Debug Tool was attempting to disable a breakpoint for a CU that has been deleted from storage (or deactivated), and failed in that attempt.

EQA1675E  variable name is not a LABEL variable or constant. No GOTO or JUMPTO commands can be issued against it.
Explanation: You are trying to use a GOTO or JUMPTO command with a variable name that cannot be associated with a label in the program.

EQA1676S  label name is a label variable that is uninitialized or that has been zeroed out. It cannot be displayed and should not be used except as the target of an assignment.
Explanation: You are trying to make use of a LABEL variable, but the control block representing that variable contains improper information (for example, an address that is zero).

EQA1677S  file name is a file variable that is uninitialized or that has been zeroed out. It cannot be displayed and should not be used except as the target of an assignment.
Explanation: You are trying to make use of a FILE variable, but the control block representing that variable contains improper information (for example, an address that is zero).

EQA1678E  The program CU-name has a short statement number table, and therefore no statement numbers in the program can be located.
Explanation: A command requires determining which statement was associated with a particular storage address. A statement table could not be located to relate storage to statement identifications.

Programmer response: Check to see if the program had been compiled using release name. If so, was the statement table suppressed?

EQA1679E  variable name is not a controlled variable. An ALLOCATE breakpoint cannot be established for it.
Explanation: You cannot establish an AT ALLOCATE breakpoint for a variable that cannot be allocated.

EQA1680E  variable name is a controlled parameter. An ALLOCATE breakpoint can be established for it only when the block in which it is declared is active.
Explanation: Debug Tool cannot, at this time, correlate a block to the named variable. As a result, a breakpoint cannot be established.

Programmer response: Establish the breakpoint via an AT ENTRY ... AT ALLOCATE ...

EQA1681E  variable name is not a FILE variable or constant.
Explanation: ON/SIGNAL file-condition (variable) is invalid because the variable is not a PL/I FILE variable.

EQA1682E  variable name is not a CONDITION variable.
Explanation: ON/SIGNAL CONDITION (variable) is invalid because the variable is not a PL/I CONDITION variable.

EQA1683E  Since the cu cu_name does not have hooks at statements with modified behavior due to the Millennium Language Extensions, all AT DATE commands will be rejected for the cu.
Explanation: A compile unit must have been compiled with the DATEPROC option and either TEST(STMT) or TEST(ALL) for hooks to be present at statements affected by the Millennium Language Extensions.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA1684E</td>
<td>Since the program for the statement has DYNAMIC DEBUG turned off, AT commands are rejected for all statements in the program.</td>
<td>A compile unit must have been compiled with TEST(STMT) or TEST(ALL) or set DYNDEBUG ON for AT statements.</td>
</tr>
<tr>
<td>EQA1685E</td>
<td>The command AT Keyword is not supported in the Compile Unit Cu_name.</td>
<td>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.</td>
</tr>
<tr>
<td>EQA1686E</td>
<td>The command opcion Keyword is not supported for Disassembly View.</td>
<td>The command or option is not supported for a DISASSEMBLY compile unit. See the Debug Tool Reference and Messages document for information about the restrictions on the use of this command.</td>
</tr>
<tr>
<td>EQA1687E</td>
<td>The command LIST Keyword is not supported in the Compile Unit Cu_name.</td>
<td>The command is not supported for a DISASSEMBLY compile unit. See the Debug Tool Reference and Messages document for information about the restrictions on the use of this command.</td>
</tr>
<tr>
<td>EQA1688E</td>
<td>Variable Variable is not available during Playback replay.</td>
<td>The expression cannot be evaluated during Playback replay, because the indicated variable is not available during playback.</td>
</tr>
<tr>
<td>EQA1689E</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>EQA1690I</td>
<td>The current programming language does not return information for DESCRIBE ENVIRONMENT.</td>
<td>The current programming language does not support the DESCRIBE ENVIRONMENT command.</td>
</tr>
<tr>
<td>EQA1691I</td>
<td>The command_name command is not supported in the current program.</td>
<td>The command, command_name, is not supported in programs that are compiled with Enterprise PL/I.</td>
</tr>
<tr>
<td>EQA1692W</td>
<td>Restoring of assembler breakpoints is not currently supported.</td>
<td>Breakpoints in assembler compile units are not restored.</td>
</tr>
<tr>
<td>EQA1693E</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>EQA1694I</td>
<td>The command_name command is not supported in the current program.</td>
<td>The command, command_name, is not supported in programs that are compiled with Enterprise PL/I.</td>
</tr>
<tr>
<td>EQA1695E</td>
<td>Variable Variable has a hex value that is too long to display.</td>
<td>The expression has a hex value that exceeds the maximum length limit required to be displayable.</td>
</tr>
<tr>
<td>EQA1696E</td>
<td>Conditional Expression Conditional_Logic_Expression in WHEN clause can not be evaluated.</td>
<td>The conditional expression is not valid. Make sure the variable is known in current compile unit or that the attributes are compatible.</td>
</tr>
<tr>
<td>EQA1697E</td>
<td>Conditional Expression Conditional_Logic_Expression in WHEN clause can not be evaluated at current location.</td>
<td>The conditional expression is not valid. Make sure the variable is known in current compile unit or that the attributes are compatible.</td>
</tr>
<tr>
<td>EQA1698E</td>
<td>The CU containing a referenced variable has not yet been entered. Storage does not exist for the referenced variable.</td>
<td>You have attempted to evaluate a variable in an implicitly created CU. Storage has not yet been allocated for this variable.</td>
</tr>
<tr>
<td>EQA1699E</td>
<td>Address Length Info Flags Name</td>
<td>This message contains the output from the DESCRIBE LOADMODS command.</td>
</tr>
<tr>
<td>EQA1700E</td>
<td>The session procedure, procedure name, is either undefined or is hidden within a larger, containing procedure.</td>
<td>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.</td>
</tr>
<tr>
<td>Message Code</td>
<td>Description</td>
<td>Explanation</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>EQA1701E</td>
<td>The maximum number of arguments to the %DUMP built-in subroutine is 2, but number were specified.</td>
<td>%DUMP does not accept more than two parameters.</td>
</tr>
<tr>
<td>EQA1702E</td>
<td>Invalid argument in CALL %DUMP.</td>
<td>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.</td>
</tr>
<tr>
<td>EQA1703E</td>
<td>No arguments can be passed to a session procedure.</td>
<td>Parameters are not supported with the CALL procedure command.</td>
</tr>
<tr>
<td>EQA1704E</td>
<td>Invalid or incompatible dump options or suboptions</td>
<td>This message is from the feedback code of Language Environment CEE3DMP call.</td>
</tr>
<tr>
<td>EQA1705E</td>
<td>Dump argument exceeds the maximum length allowed.</td>
<td>The dump option allows a maximum of 255 characters. The dump title allows a maximum of 80 characters.</td>
</tr>
<tr>
<td>EQA1706E</td>
<td>pgmname must be loaded before calling the program.</td>
<td>The CALL command was terminated unsuccessfully.</td>
</tr>
<tr>
<td>EQA1707E</td>
<td>The following data was produced by Fault Analyzer.</td>
<td>This message is used as a header for the call %FA.</td>
</tr>
<tr>
<td>EQA1708I</td>
<td>The HOGAN environment is not available.</td>
<td>The Computer Sciences Corporation's KORE-HOGAN product is not installed.</td>
</tr>
<tr>
<td>EQA1709E</td>
<td>Command CALL %HOGAN is only available in a CICS environment.</td>
<td>The CALL %HOGAN command is only valid in a CICS environment with Computer Sciences Corporation’s KORE-HOGAN installed.</td>
</tr>
<tr>
<td>EQA1710E</td>
<td>You are not authorized to execute that function.</td>
<td>The function that you requested has been rejected by a security manager.</td>
</tr>
<tr>
<td>EQA1711E</td>
<td>Program can not be found.</td>
<td>An error occurred in locating the program needed to perform the function you requested.</td>
</tr>
<tr>
<td>EQA1712E</td>
<td>Function not available in Dual Terminal Mode.</td>
<td>The function that you requested is not supported when Debug Tool is running in Dual Terminal mode.</td>
</tr>
<tr>
<td>EQA1713E</td>
<td>Load module load_module could not be found.</td>
<td>The indicated load module was specified as an operand of the DESCRIBE LOADMODS command but is not an active load module.</td>
</tr>
<tr>
<td>EQA1714I</td>
<td>BP_Operation successful for suspended breakpoint.</td>
<td>The requested breakpoint was successfully performed on a suspended breakpoint.</td>
</tr>
<tr>
<td>EQA1720E</td>
<td>There is no declaration for variable name.</td>
<td>A command (for example, CLEAR VARIABLES) requires the use of a variable, but the specified variable was not declared (or was previously cleared).</td>
</tr>
<tr>
<td>EQA1721E</td>
<td>The size of the variable is too large.</td>
<td>A variable can require no more than 2<strong>24 - 1 bytes in a non-XA machine and no more than 2</strong>31 - 1 bytes in an XA machine.</td>
</tr>
<tr>
<td>EQA1722E</td>
<td>Error in declaration; invalid attribute variable name.</td>
<td>A session variable is declared with invalid or unsupported attribute.</td>
</tr>
<tr>
<td>EQA1723E</td>
<td>There is no session variables defined.</td>
<td>The CLEAR VARIABLES command is entered but there is no declaration for session variables.</td>
</tr>
<tr>
<td>EQA1724E</td>
<td>There is no tag type tag named tag name.</td>
<td>This message applies to C. It is issued, for example, after DESCRIBE ATTRIBUTES enum x if x is not an enum tag.</td>
</tr>
<tr>
<td>EQA1725E</td>
<td>tag type tag name is already defined.</td>
<td>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.</td>
</tr>
</tbody>
</table>
| EQA1726E     | tag type tag name cannot be cleared while one or more declarations refer to that type. | 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 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.

EQA1740E EQALANGX debug file cannot be found for Compile_Unit_name. Use the SET SOURCE command to indicate the new location of the EQALANGX file.
Explanation: The EQALANGX file containing the listing and the debugging tables cannot be found. Some of the possible conditions that could cause this are: The debug file does not exist under the default DSName, or the user does not have authorization to access the debug file.

EQA1741E Error in setting DBCS ON when the terminal is not DBCS capable.
Explanation: Error in setting DBCS ON when the debug session terminal is not DBCS capable.

EQA1742I Debug Trace: Trace Data
Explanation: This is output generated by internal Debug Tool trace for problem determination purposes only.

EQA1743I save_restore_cmd not restored from dsname
Explanation: An attempt was made to 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.

EQA1744I save_restore_cmd is in effect for dsname
Explanation: The specified data was successfully restored from the specified data set.

EQA1745I save_restore_cmd restored from 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.

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.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA1752S</td>
<td>Comparison in command-name command was invalid. The command was ignored.</td>
<td>This message applies to COBOL. The operands to be compared are of incompatible types.</td>
</tr>
<tr>
<td>EQA1753S</td>
<td>The nesting of &quot;switch&quot; command exceeded the maximum.</td>
<td>This message applies to C. There are too many nested levels of switch commands.</td>
</tr>
<tr>
<td>EQA1754S</td>
<td>An error occurred in &quot;switch&quot; command processing. The command is terminated.</td>
<td>This message applies to C. The switch command is terminated because an error occurred during processing.</td>
</tr>
<tr>
<td>EQA1755S</td>
<td>Comparison with the keyword-name keyword in command-name command was invalid. The command was ignored.</td>
<td>This message applies to COBOL. The operands to be compared are incompatible. For example, the following comparison is invalid:</td>
</tr>
<tr>
<td></td>
<td>EVALUATE TRUE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>When 6 List ('invalid'); when other List ('other'); END-EVALUATE</td>
<td></td>
</tr>
<tr>
<td>EQA1756W</td>
<td>EQALANGX file not available for CSECT CMD_NAME.</td>
<td>A debug data file is not found for the supplied CSECT name.</td>
</tr>
<tr>
<td>EQA1757W</td>
<td>Cannot save_restore breakpoints and/or monitors because an OPEN exit is active.</td>
<td>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.</td>
</tr>
<tr>
<td>EQA1758E</td>
<td>Operation is not permitted in Browse Mode.</td>
<td>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.</td>
</tr>
<tr>
<td>EQA1759E</td>
<td>Operations that alter the control flow are not permitted in Browse Mode.</td>
<td>When Browse Mode is active, operations that modify the flow of control are not allowed.</td>
</tr>
<tr>
<td>EQA1760I</td>
<td>Use QUIT DEBUG or QUIT ABEND to exit Debug Tool in Browse Mode.</td>
<td>When Browse Mode is active, you cannot use QUIT, QUIT; or QUIT (expression) because this alters the control flow. You must use either QUIT DEBUG or QUIT ABEND.</td>
</tr>
<tr>
<td>EQA1763E</td>
<td>save_restore_cmd failed because dsname is not a partitioned data set.</td>
<td>The specified data set must be a partitioned data set.</td>
</tr>
<tr>
<td>EQA1764E</td>
<td>save_restore_cmd could not locate data set dsname.</td>
<td>The specified data set could not be located.</td>
</tr>
<tr>
<td>EQA1765E</td>
<td>save_restore_cmd error rc-reason allocating dsname.</td>
<td>The specified data set could not be allocated. The return code and reason code are shown as ddd-xxxxxyyyy. In most cases, xxx 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 MVS Programming: Authorized Assembler Services Guide You should also inspect the MVS console log for other messages associated with this error.</td>
</tr>
<tr>
<td>EQA1766E</td>
<td>The target of the GOTO command is in an inactive block.</td>
<td>You are trying to GOTO a block that is not active. If it is inactive it doesn’t have a register save area, base registers, and so on (all of the mechanics established that would allow the procedure to run).</td>
</tr>
<tr>
<td>EQA1767S</td>
<td>No offset was found for label &quot;label&quot;.</td>
<td>No offset associated with the label was found; the code associated with the label might have been removed by optimization.</td>
</tr>
<tr>
<td>EQA1768S</td>
<td>The label &quot;label&quot; is not known.</td>
<td>The label is not known.</td>
</tr>
<tr>
<td>EQA1769S</td>
<td>The label &quot;label&quot; is ambiguous - multiple labels of this name exist.</td>
<td>The label is ambiguous; multiple labels of this name exist.</td>
</tr>
<tr>
<td>EQA1770S</td>
<td>The GOTO is not permitted, perhaps because of optimization.</td>
<td>The GOTO command is not recommended. For COBOL, this might be due to optimization, or because register contents other than the code base cannot be guaranteed for the target.</td>
</tr>
<tr>
<td>EQA1771S</td>
<td>The GOTO is not permitted due to language rules.</td>
<td>The GOTO command is not recommended. For COBOL, this might be due to optimization, or because register contents other than the code base cannot be guaranteed for the target.</td>
</tr>
</tbody>
</table>
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 not be associated with a label in the program.

EQA1778S The JUMPTO is not allowed, perhaps because of optimization.
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.

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: Debug Tool 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. Debug Tool 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. Debug Tool 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 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.

EQA1793S 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.

EQA1794S 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.

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 be 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.
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Explanation</th>
<th>Programmer response</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA1823E</td>
<td>An end of line was encountered before the end of a DBCS name or DBCS literal.</td>
<td>An end of line was encountered before finding a closing shift-in control code. This message is for the System/370 environment.</td>
<td></td>
</tr>
<tr>
<td>EQA1824E</td>
<td>A DBCS literal or DBCS name contains no DBCS characters.</td>
<td>A shift-out shift-in pair of control characters were found with no intervening DBCS characters. This message is for the System/370 environment.</td>
<td></td>
</tr>
<tr>
<td>EQA1825E</td>
<td>End-of-source was encountered while processing a DBCS name or DBCS literal.</td>
<td>No closing Shift-In control code was found before end of file. This message is for the System/370 environment.</td>
<td></td>
</tr>
<tr>
<td>EQA1826E</td>
<td>A DBCS literal was not delimited by a trailing quote or apostrophe.</td>
<td>No closing quotation mark.</td>
<td></td>
</tr>
<tr>
<td>EQA1827E</td>
<td>Invalid separator character found following a DBCS name.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQA1828E</td>
<td>Fixed binary constants are limited to 31 digits.</td>
<td>A fixed binary constant must be between (-2^{31}) and (2^{31}) exclusive.</td>
<td></td>
</tr>
<tr>
<td>EQA1829E</td>
<td>Fixed decimal constants are limited to 20 digits.</td>
<td>A fixed decimal constant must be between (-10^{20}) and (10^{20}) exclusive.</td>
<td></td>
</tr>
<tr>
<td>EQA1830E</td>
<td>Float binary constants are limited to 109 digits.</td>
<td>This limit applies to all PL/I FLOAT BINARY constants.</td>
<td></td>
</tr>
<tr>
<td>EQA1831E</td>
<td>Float decimal constants are limited to 33 digits.</td>
<td>This limit applies to all PL/I FLOAT DECIMAL constants.</td>
<td></td>
</tr>
<tr>
<td>EQA1832E</td>
<td>Floating-point exponents are limited to 3 digits.</td>
<td>This limit applies to all C float constants and to all PL/I FLOAT BINARY constants.</td>
<td></td>
</tr>
<tr>
<td>EQA1833E</td>
<td>Float decimal exponents are limited to 2 digits.</td>
<td>This limit applies to all PL/I FLOAT DECIMAL constants.</td>
<td></td>
</tr>
<tr>
<td>EQA1834E</td>
<td>Float binary constants must be less than 1E+252B.</td>
<td>This limit applies to all PL/I FLOAT BINARY constants.</td>
<td></td>
</tr>
<tr>
<td>EQA1835E</td>
<td>Float decimal constants must be less than 7.2370055773226221397318656304298E+75.</td>
<td>This limit applies to all PL/I FLOAT DECIMAL constants.</td>
<td></td>
</tr>
<tr>
<td>EQA1836E</td>
<td>Float constants are limited to 35 digits.</td>
<td>This limit applies to all C float constants.</td>
<td></td>
</tr>
<tr>
<td>EQA1837E</td>
<td>Float constants must be bigger than 5.3976053469340278908664699142502496E-79 and less than 7.23700557732262213973186563042929E+75.</td>
<td>This is the range of values allowed by C.</td>
<td></td>
</tr>
<tr>
<td>EQA1856S</td>
<td>The module &quot;module&quot; cannot be loaded - it was NOT found.</td>
<td>The LOAD MODULE request failed, the module was not found.</td>
<td></td>
</tr>
<tr>
<td>EQA1857S</td>
<td>The module &quot;module&quot; cannot be loaded - there is NOT enough storage to do the load.</td>
<td>The LOAD MODULE request failed, there is not enough storage.</td>
<td></td>
</tr>
<tr>
<td>EQA1858S</td>
<td>The module &quot;module&quot; cannot be loaded.</td>
<td>The LOAD MODULE request was unsuccessful.</td>
<td></td>
</tr>
<tr>
<td>EQA1859S</td>
<td>The module &quot;module&quot; cannot be deleted.</td>
<td>The DELETE MODULE request was unsuccessful.</td>
<td></td>
</tr>
<tr>
<td>EQA1860S</td>
<td>The module &quot;module&quot; was NOT loaded by the Debug Tool and therefore CANNOT be deleted.</td>
<td>Only modules loaded by the Debug Tool may be deleted.</td>
<td></td>
</tr>
<tr>
<td>EQA1861S</td>
<td>The module &quot;module&quot; cannot be loaded because it was already loaded by Debug Tool.</td>
<td>The LOAD MODULE request was unsuccessful.</td>
<td></td>
</tr>
<tr>
<td>EQA1862E</td>
<td>IDISNAP could not be loaded. Verify Fault Analyzer is available or loaded.</td>
<td>IDISNAP is a part of the product Fault Analyzer. Verify Fault Analyzer is installed properly.</td>
<td></td>
</tr>
<tr>
<td>EQA1863I</td>
<td>%FA complete. See your Fault Analyzer history file.</td>
<td>%FA was complete.</td>
<td></td>
</tr>
</tbody>
</table>

Chapter 8. Debug Tool messages 335
EQA1864S  LOAD is not supported during initialization.
STEP or GO, then retry the LOAD command.
Explanation:  You cannot issue a LOAD request until Debug Tool
initialization is complete.

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 GOS/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 Debug Tool 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 Debug Tool 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.

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 Debug Tool 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 Debug Tool tried to trigger the specified condition, it failed and the condition was not signaled.

EQA1908S  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.

EQA1909E  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 run-time environment or you have to qualify to some block explicitly.

EQA1910E  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.

EQA1911S  There is no block named block_name.
Explanation: The block that you named could not be located by Debug Tool.
Programmer response: Provide further block name qualification: by load module name, by compile unit name, or by additional block name(s) if a nested block.

Chapter 8. Debug Tool messages 337
### EQA1922S
**Message:** 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 Debug Tool from locating the target block.

**Programmer response:** Verify that the named block should be within the current qualification.

### EQA1923S
**Message:** There is no compilation unit named `cu_name`.

**Explanation:** The compilation unit (program) that you named could not be located by Debug Tool.

### EQA1924S
**Message:** 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
**Message:** 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 Debug Tool was first used. On the System/370, Debug Tool is aware of additional load modules only if they were FETCHed after Debug Tool got control for the first time.

### EQA1926S
**Message:** There is no cu named `cu_name` within `load module load module name`.

**Explanation:** The compilation unit might be misspelled or missing.

**Programmer response:** Add further qualification so that the correct load module will be known.

### EQA1927S
**Message:** 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:** Provide further block name qualification—by load module name, by compile unit name, or by additional block names(s) if a nested block.

### EQA1928S
**Message:** 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
**Message:** 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
**Message:** There is no compilation unit named `CU-name` in the current enclave.

**Explanation:** The compilation unit (program) that you named could not be located in the current enclave by Debug Tool.

### EQA1931S
**Message:** 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
**Message:** Block or CU `block_name` is not currently available.

**Explanation:** The block or CU that you named could not be located by Debug Tool.

**Programmer response:** Provide further block name qualification—by load module name, by compile unit name, or by additional block names(s) if a nested block.

### EQA1933W
**Message:** The program was compiled with the OPTIMIZE compiler option either by a release of the COBOL compiler that does not support debugging of optimized code, or by a release of the compiler that is missing the required service for debugging of optimized code. Until it is recompiled with the proper release and service level of the COBOL compiler, results of Debug Tool commands are unpredictable.

**Explanation:** Debug Tool does not have accurate information about the program, and thus cannot provide reliable results.

**Programmer response:** Recompile the program with one of the following or later versions of the COBOL compiler:
- COBOL FOR OS/390 & VM Version 2 with APAR PQ63234 installed
- Enterprise COBOL for z/OS and OS/390 Version 3 Release 1 with APAR PQ63235 installed
- Enterprise COBOL for z/OS and OS/390 Version 3 Release 2

### EQA1934E
**Message:** 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
**Message:** 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
**Message:** The assignment was performed but the assigned value might not be used by the program, due to optimization.

**Explanation:** Results might be unreliable because the program might use the previous value that was saved in temporary storage or a register.
**Programmer response:** Recompile the program without the Optimize option.

**EQA1937W** This breakpoint is deferred.

**Explanation:** The compilation unit (program) that you specified could not be located by the Debug Tool. The breakpoint is deferred until this CU is entered.

**EQA1938W** Provide a CU (Program) Name to qualify the block name.

**Explanation:** The CU name (Program) must be added to the block name to allow Debug Tool to locate the block named.

**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 command was issued but the MONITOR table is full.

**EQA1951E** 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 Debug Tool. Uppercase United States English (UEN) will be used instead.

Explanation: The national-language-specified conflicts with the supported national languages for this installation of Debug Tool. Verify that the Language Environment run-time NATLANG option is correct.

EQA1971E  The return code in the QUIT command must be nonnegative and less than 1000.

Explanation: For PL/I, the value of the return code must be nonnegative and less than 1000.
EQA1972E  variable name is not a LABEL constant No AT commands can be issued against it
Explanation: LABEL variables cannot be the object of the AT command.

EQA1973E  The FIND argument cannot exceed a string length of 64
Explanation: Shorten the search argument to a string length 64 or less.

EQA1974E  The FIND argument is invalid, the string length is zero
Explanation: Supply a search argument inside the 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 can not be found. Some of the possible conditions that could cause this are: The Debug File was deleted from the system, or the user does not have authorization to access the debug file.

EQA1976E  The debug information for Compile_Unit_name has already been validated, changing the debug file is not allowed. The command will not be performed.
Explanation: A Debug File containing the listing and the debugging tables has already 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 can not be used.
Explanation: A Debug File containing the listing and the debugging tables does not match the creation date of the object, and the data it contains is not valid.

EQA1979E  The Debug File for Compile_Unit_name is not available or was not found.
Explanation: The Debug File was nowhere to be found.

EQA1980E  The Debug File for Compile_Unit_name could not be opened or read.
Explanation: I/O errors when trying to open/read Debug File.

EQA1981E  Invalid mode name, transaction program name, or partner LU name associated with symbolic_destination_name. Mode_name= mode_name and partner_LU_name= partner_LU_name
Explanation: A conversation allocation request failed due to invalid conversation characteristics obtained from the APPC/MVS side information file. There could be several reasons for this:
1. The mode_name characteristic specifies a mode name that is either not recognized by the LU as valid or is reserved for SNA service transaction programs.
2. The TP_name characteristic specifies an SNA service transaction program name.
3. The partner_LU_name characteristic specifies a partner LU name that is not recognized by the LU as being valid.

Programmer response: Contact your APPC/MVS system administrator to modify the characteristics associated with the given symbolic_destination_name in the side information file. For information about the recommended values for mode_name and TP_name, see the CODE/370 Installation manual. The OS/2 system error log can contain valuable diagnostic information. To access the system error log, select System Error Log from the FFST/2™ folder or type SYSLOG at the OS/2 command line.

EQA1982E  Permanent conversation allocation failure for symbolic_destination_name. Partner_LU_name= partner_LU_name and mode_name= mode_name
Explanation: The conversation cannot be allocated because of a condition that is not temporary. There could be several reasons for this:
1. The workstation where the partner_LU_name is defined is turned off or Communications Manager/2 is not started.
2. The partner_LU_name has not been defined.
3. The current session limit for the specified partner_LU_name and mode_name pair is zero.
4. A system definition error or a session-activation protocol error has occurred.

Programmer response: Ensure that you specified the correct symbolic_destination_name or contact your APPC/MVS system administrator to correct the condition. The OS/2 system error log can contain valuable diagnostic information. To access the system error log, select System Error Log from the FFST/2™ folder or type SYSLOG at the OS/2 command line.

EQA1983E  Temporary conversation allocation failure for symbolic_destination_name. Partner_LU_name= partner_LU_name and mode_name= mode_name
Explanation: The conversation cannot be allocated because of a condition that might be temporary. There could be several reasons for this:
1. Undefined mode_name (not temporary)
2. Temporary lack of resources at the host LU or workstation LU
Verify that mode_name is defined on the target workstation using the CM/2 Communication Manager Setup panels. If mode_name is defined on the workstation, contact your MVS/ESA system programmer to ensure that mode_name is also defined on the MVS system. The OS/2 system error log can contain valuable diagnostic information. To access the system error log, select System Error Log from the FFST/2™ folder or type SYSLOG at the OS/2 command line.
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 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 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.

Unrecognized transaction program 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.

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.

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.

The environment variable QPPLISTFILES is not defined.

Explanation: For Q++ programs, Debug Tool 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.

Invalid session ID - session_ID

Explanation: Conversation initialization failed due to an invalid session ID in the Session Parameter. There could be several reasons for this,

1. The session ID is longer than 8 characters or contains invalid characters. Valid session IDs consist of 1-8 alphanumeric characters.
2. There is already another PWS Debug Tool session with the given session ID.

Programmer response: 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.

Invalid session parameter - session_parameter

Explanation: Conversation initialization failed due to 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 Debug Tool manual.

CICS terminal TERM is not accessible

Explanation: The terminal id specified to receive Debug Tool screen was detected but not acquired.

Programmer response: Correct the Debug Tool Term Id using DTCN Replace function or logon to already defined one.

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.
The load module might be missing or it cannot be located. Load module qualification is referring to a load module that cannot be located.

Programmer response: Correct or enter the TCP/IP port ID.

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 Debug Tool was first used. Debug Tool is aware of additional load modules only if they were FETCHed after Debug Tool got control for the first time.

There is no CU named & & & & within load module & & & &.

Explanation: The compilation unit may be misspelled or missing.

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 SET QUALIFY or supply the desired qualification to the command in question.

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.

VTAM 3270 error_type error, RRC=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 Debug Tool.

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% operand of the TEST 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 7. Definitions for error_type, insert1, insert2, and insert3

<table>
<thead>
<tr>
<th>error_type</th>
<th>insert1</th>
<th>insert2</th>
<th>insert3</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPL or INQUIRE RPL</td>
<td>ggunnnn where: gg - CsFlag, nn - CsFunc, oo - CSFlagOf, and mm - CSReqMod</td>
<td>rncnnn where: r-R15, r-R0, or r-CRNL Sense if RtnCodeFdBk2 = 0404 or 0405, and oo0o=Error Offset in EQAYVTAM</td>
<td>pppdkk where: pp=RPLCode, dd=RtnCode, and kkkk-FdBk2</td>
</tr>
<tr>
<td>ACB</td>
<td>ggunnnn where: gg - CsFlag, nn - CsFunc, oo - CSFlagOf, and mm - CSReqMod</td>
<td>Error Offset in EQAYVTAM</td>
<td>ACBERR</td>
</tr>
<tr>
<td>MODCB</td>
<td>ggunnnn where: gg - CsFlag, nn - CsFunc, oo - CSFlagOf, and mm - CSReqMod</td>
<td>Error Offset in EQAYVTAM</td>
<td>xxxxx where: xxxx-0000, y9-MODCB R0, and z2-MODCB R15</td>
</tr>
<tr>
<td>Logic</td>
<td>ggunnnn where: gg - CsFlag, nn - CsFunc, oo - CSFlagOf, and mm - CSReqMod</td>
<td>Error Offset in EQAYVTAM</td>
<td>0</td>
</tr>
<tr>
<td>Function</td>
<td>ggunnnn where: gg - CsFlag, nn - CsFunc, oo - CSFlagOf, and mm - CSReqMod</td>
<td>Error Offset in EQAYVTAM</td>
<td>EQAYVTAM function code</td>
</tr>
<tr>
<td>Storage</td>
<td>ggunnnn where: gg - CsFlag, nn - CsFunc, oo - CSFlagOf, and mm - CSReqMod</td>
<td>Error Offset in EQAYVTAM</td>
<td>0</td>
</tr>
<tr>
<td>Undefined LU</td>
<td>ggunnnn where: gg - CsFlag, nn - CsFunc, oo - CSFlagOf, and mm - CSReqMod</td>
<td>Error Offset in EQAYVTAM</td>
<td>0</td>
</tr>
<tr>
<td>Unknown</td>
<td>ggunnnn where: gg - CsFlag, nn - CsFunc, oo - CSFlagOf, and mm - CSReqMod</td>
<td>Error Offset in EQAYVTAM</td>
<td>0</td>
</tr>
</tbody>
</table>

Programmer response: If an ACB error is reported, check with your installer to ensure that the VTAM modifications required by Debug Tool have been made. If undefined LU error is reported, check the MFI% operand of the TEST parameter to ensure that the correct dedicated terminal Logical Unit identifier was specified and that the terminal in known to VTAM.

Otherwise, contact IBM support.
**EQA1999I • EQA2257I**

**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’.

**EQA2002E** Syntax error: The expression is incomplete.

**EQA2003E** Cannot find the symbol ‘%1’.

**EQA2004E** Cannot find a type for symbol ‘%1’.

**EQA2005E** Syntax error: The expression is not a pointer.

**EQA2006E** Syntax error: The expression is not a class.

**EQA2007E** Syntax error: Invalid literal starting at %1

**EQA2008E** Invalid operand

**EQA2009E** Invalid type

**EQA2010E** Invalid operator

**EQA2011E** Invalid operation

**EQA2012E** Unsupported operator

**EQA2013E** Unknown expression type

**EQA2014E** An expression may not return a value of type ‘%1’.

**EQA2015E** An expression may not return ‘%1’ because it has no data members.

**EQA2016E** The expression cannot be parsed.

**EQA2017E** Functions with C++ linkage may not be used in expressions. Only functions with C linkage are permitted.

**EQA2018E** Invalid reference

**EQA2019E** Subscript out of bounds

**EQA2020E** Failed to evaluate

**EQA2021E** Call show_self

**EQA2022E** Evaluate now

**EQA2023E** Not within current scope

**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
Debug Tool 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. Debug Tool 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
*** Commands file commands end ***

**Explanation:** End of commands in the commands file.

### EQA2265I
Global Preferences file exists: &&&&

**Explanation:** The global preferences file is opened successfully.

### EQA2266I
*** User preferences file commands follow ***

**Explanation:** Start of commands in the user preferences file.

### EQA2267I
*** Commands file commands follow ***

**Explanation:** Start of commands in the commands file.

### EQA2268I
*** Global preferences file commands end ***

**Explanation:** End of commands in the global preferences file.

### EQA2269I
*** User preferences file commands end ***

**Explanation:** End of commands in the user preferences file.
<table>
<thead>
<tr>
<th>Message ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA2280E</td>
<td>LIST CONTAINER command is only available in a CICS Environment.</td>
</tr>
<tr>
<td>EQA2291S</td>
<td>The module &quot;module_name&quot; cannot be deleted.</td>
</tr>
<tr>
<td>EQA2281E</td>
<td>DESCRIBE CHANNEL command is only available in a CICS Environment.</td>
</tr>
<tr>
<td>EQA2292S</td>
<td>The module &quot;module_name&quot; was NOT loaded by the Debug Tool and therefore CANNOT be deleted.</td>
</tr>
<tr>
<td>EQA2282E</td>
<td>This command is not supported in this CICS Version/Release.</td>
</tr>
<tr>
<td>EQA2293S</td>
<td>The module &quot;module_name&quot; cannot be loaded because it was already loaded by Debug Tool.</td>
</tr>
<tr>
<td>EQA2283E</td>
<td>There is no container with that name in the specified channel.</td>
</tr>
<tr>
<td>EQA2294I</td>
<td>The LOADDEBUGDATA command for CU_name has been deferred until the CU appears.</td>
</tr>
<tr>
<td>EQA2284E</td>
<td>There are no containers to display.</td>
</tr>
<tr>
<td>EQA2295I</td>
<td>The CU specified for the LOADDEBUGDATA command is already an assembler or non-Language Environment COBOL CU.</td>
</tr>
<tr>
<td>EQA2285E</td>
<td>There are no channels to display.</td>
</tr>
<tr>
<td>EQA2296E</td>
<td>The CU specified for the LOADDEBUGDATA command is not a disassembly CU.</td>
</tr>
<tr>
<td>EQA2286E</td>
<td>There is no channel with that name in this program.</td>
</tr>
<tr>
<td>EQA2297E</td>
<td>An error occurred while attempting to load the debug (EQALANGX) file for a specified CU.</td>
</tr>
<tr>
<td>EQA2287E</td>
<td>There is no current channel in this program.</td>
</tr>
<tr>
<td>EQA2298I</td>
<td>The LOAD MODULE request was unsuccessful.</td>
</tr>
<tr>
<td>EQA2288S</td>
<td>The module &quot;module_name&quot; cannot be loaded - it was NOT found.</td>
</tr>
<tr>
<td>EQA2299E</td>
<td>The EQALANGX file does not match the object for object_name. The EQALANGX file cannot be used.</td>
</tr>
<tr>
<td>EQA2289S</td>
<td>The module &quot;module_name&quot; cannot be loaded because it was already loaded by Debug Tool.</td>
</tr>
<tr>
<td>EQA2290S</td>
<td>There is no container with that name in the specified channel.</td>
</tr>
<tr>
<td>EQA2299S</td>
<td>There is no compilation unit named compile_unit_name.</td>
</tr>
<tr>
<td>EQA2299S</td>
<td>The module &quot;module_name&quot; cannot be loaded.</td>
</tr>
<tr>
<td></td>
<td>The LOAD MODULE request was unsuccessful.</td>
</tr>
</tbody>
</table>

*Explanation:* The module "module_name" cannot be loaded - it was NOT found. The LOAD MODULE request failed, the module was not found. The LOAD MODULE request failed, there is not enough storage. Read CICS Documentation on Channels and Containers. The container name provided does not belong to the channel specified or does not exist. You must use CICS TS 3.1 or later to be able to use this feature. Only modules loaded by Debug Tool may be deleted. The LOAD MODULE request was unsuccessful. The indicated CU is not currently known to Debug Tool. The LOADDEBUGDATA will be run when the CU appears in a loaded module. An L00 has already been done for the CU specified in the L00 command. This L00 may have been done previously by the user or an implicit L00 may have been done for the CU. This happens when a user-entered L00 is successful and, subsequently, the CU goes away and later reappears. Only a disassembly CU can be identified as assembler CU. 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. The compilation unit (program) that you named could not be located by Debug Tool. 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.
A pattern of "*" is invalid.

Explanation: The NAMES EXCLUDE command does not allow a pattern of "*".

Value too long to display.

Not allocated

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.

Format of value failed

NULL

Register not used

Invalid string

Divide by zero

Invalid expression

Expression not supported

Incompatible types

Expression validation failed

Expression evaluation failed

Evaluation not supported

Expression not evaluated

Variable not found

Invalid value for update

Update of value not allowed at this time

Update of value not supported

Operation not supported

Please see log window for messages

Invalid address

Storage unit style is not supported

Storage address style is not supported

Register variable(s) out of range

Invalid program name

Failure loading view information

Failure evaluating expression context. Contact IBM support.

Frequency data is not allowed with this breakpoint type.

Invalid line to set a line breakpoint

Initialization failure - Debug Tool and front-end levels are incompatible

Invalid storage data

Incomplete update - a portion of storage is not updateable

Storage is not updateable

Failure updating storage contents

Procedure name is an internal procedure, not an entry point

Breakpoint type not supported

Program at end of job

A DLL load occurred. Current program location cannot be determined

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
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA2381E</td>
<td>The target of the Jump to Location is invalid.</td>
</tr>
<tr>
<td>EQA2382E</td>
<td>The target of the Run to Location is invalid.</td>
</tr>
<tr>
<td>EQA2383I</td>
<td>The environment is not yet fully initialized. Use Step or Run.</td>
</tr>
<tr>
<td>EQA2384E</td>
<td>The Defer option is not permitted for Line Breakpoints.</td>
</tr>
<tr>
<td>EQA2385E</td>
<td>The entry point is not active or debug data is not available.</td>
</tr>
<tr>
<td>EQA2386E</td>
<td>You are monitoring an inaccessible or uninitialized variable.</td>
</tr>
<tr>
<td>EQA2387E</td>
<td>Cannot modify BreakPoints. Delete existing BreakPoint then add new.</td>
</tr>
<tr>
<td>EQA2388E</td>
<td>Cannot monitor this type of expression or variable</td>
</tr>
<tr>
<td>EQA2389E</td>
<td>The C component of the LE runtime has not been initialized.</td>
</tr>
<tr>
<td>EQA2390E</td>
<td>Exec Imminent. Click any button to continue.</td>
</tr>
<tr>
<td>EQA2391E</td>
<td>Time stamp on listing does not match time stamp on object.</td>
</tr>
<tr>
<td>EQA2392E</td>
<td>Unable to find requested executable module.</td>
</tr>
<tr>
<td>EQA2393E</td>
<td>Debug Tool has frozen this thread. Registers are not available.</td>
</tr>
<tr>
<td>EQA2394E</td>
<td>Stop at date field references.</td>
</tr>
<tr>
<td>EQA2395E</td>
<td>The entered expression is invalid, please check the expression.</td>
</tr>
<tr>
<td>EQA2396E</td>
<td>Expression breakpoints are not supported for this language.</td>
</tr>
<tr>
<td>EQA2397E</td>
<td>The expression has inaccessible or uninitialized data.</td>
</tr>
<tr>
<td>EQA2398E</td>
<td>No hooks present at statements, breakpoint not set.</td>
</tr>
<tr>
<td>EQA2399E</td>
<td>Debug Tool has frozen this thread. Call stack is not available.</td>
</tr>
<tr>
<td>EQA2400E</td>
<td>Operations involving engine settings are currently not supported.</td>
</tr>
<tr>
<td>EQA2401E</td>
<td>This register cannot be edited. Changes have been ignored.</td>
</tr>
<tr>
<td>EQA2402I</td>
<td>Program was stopped due to load occurrence breakpoint.</td>
</tr>
<tr>
<td>EQA2403I</td>
<td>Program was stopped due to storage change breakpoint.</td>
</tr>
<tr>
<td>EQA2404E</td>
<td>The debug information is not accessible for the requested thread.</td>
</tr>
<tr>
<td>EQA2405E</td>
<td>The PL/I component of the LE runtime has not been initialized.</td>
</tr>
<tr>
<td>EQA2406E</td>
<td>The Cobol component of the LE runtime has not been initialized.</td>
</tr>
<tr>
<td>EQA2407E</td>
<td>Too many local variables for local monitor. Use program monitor instead.</td>
</tr>
<tr>
<td>EQA2408E</td>
<td>Variable cannot be displayed because this compile unit was compiled without symbolic information.</td>
</tr>
</tbody>
</table>

**Explanation:** The current compile unit was compiled without symbolic information. Variable information is not accessible to Debug Tool. The CU must be recompiled with TEST to provide this information.

**Programmer response:** If the LAST option needs to be exercised, consider debugging the application in dual terminal mode.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA2409E</td>
<td>The LAST option on an EXEC CICS SEND command has been suppressed.</td>
</tr>
</tbody>
</table>

**Explanation:** The application has issued an EXEC CICS SEND command with the LAST Option while being debugged in single terminal mode. This would end the terminal session being used by Debug Tool, so Debug Tool has suppressed the LAST option.

**Programmer response:** If the LAST option needs to be exercised, consider debugging the application in dual terminal mode.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA2410E</td>
<td>Search target not found.</td>
</tr>
</tbody>
</table>

**Explanation:** The search string was not found.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA2411E</td>
<td>Variable needs further qualification or qualification is invalid.</td>
</tr>
</tbody>
</table>

**Explanation:** A qualified reference is invalid. One or more qualifiers may be undefined or not in the same structure as the desired data item.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA2412E</td>
<td>You are currently on an instruction that will leave the current Compile Unit and may cause Debug Tool to lose control. You must ensure that a breakpoint is set on a subsequent instruction or statement. At location LOCN.</td>
</tr>
</tbody>
</table>

**Explanation:** While stepping through a program in the disassembly view, you are about to execute an instruction that will
cause a transfer (branch) out of the current Compile Unit. A breakpoint is required where you would like to obtain control.

Programmer response: Set a breakpoint and/or enter STEP or GO to continue.

---

**EQA2413I**  
You are currently on an instruction that must run without a breakpoint. Debug Tool may lose control. You must ensure that a breakpoint is set on a subsequent instruction or statement. At location LOCN.

Explanation: While stepping through a program in the disassembly view, you are about to execute an instruction that must run from the original location and therefore the breakpoint must be temporarily removed. A breakpoint is required where you would like to obtain control. The breakpoint is automatically removed.

Programmer response: Set a breakpoint and/or enter STEP or GO to continue.

---

**EQA2413I**  
You are currently on an instruction that must run without a breakpoint. Debug Tool may lose control. You must ensure that a breakpoint is set on a subsequent instruction or statement. At location LOCN.

Explanation: While stepping through a program in the disassembly view, you are about to execute an instruction that must run from the original location and therefore the breakpoint must be temporarily removed. A breakpoint is required where you would like to obtain control. The breakpoint is automatically removed.

Programmer response: Set a breakpoint and/or enter STEP or GO to continue.

---

**EQA2414I**  
Debug Tool could not stop at one or more instructions because a valid save area backchain was not found. At location LOCN.

Explanation: While debugging a program, Debug Tool could not stop the application because a valid save area 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.

Programmer response: Set a breakpoint and/or enter STEP or GO to continue.

---

**EQA2415I**  
Debug Tool detected a missing or invalid Debug Tool SVC EQA005VC(GC0014E). The installed version is 12.11. DYNDEBUG is disabled.

Explanation: During initialization Debug Tool did not find a usable Dynamic Debug facility SVC. Either the SVC was not found or a downlevel version was detected. The Dynamic Debug facility is disabled.

Programmer response: Have your installer install the correct Dynamic Debug facility SVC.

---

**EQA2416I**  
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.

---

**EQA2417I**  
Return code 63 from PLAYBACK run-time API. You must DISABLE PLAYBACK.

Explanation: The run-time API is no longer able to collect application DATA. The return code indicates a terminal error in the run-time.

Programmer response: Use the PLAYBACK DISABLE command to disable Playback. Return code 63 indicates not enough memory was available for Playback. Restart your Debug Tool session and enter the PLAYBACK ENABLE command using the integer option. For example: PLAYBACK ENABLE * 10000.

---

**EQA2419W**  
Playback data collection has wrapped. Earlier data has been overlayed.

Explanation: Playback data collection has used all available memory. The earliest collected data will be overlayed with newer data.

Programmer response: If it is necessary to retain more Playback data, specify a larger memory size on the PLAYBACK ENABLE command.

---

**EQA2420W**  
The assignment was performed but the assigned value might not be used by the program, due to optimization.

Explanation: Results might be unreliable because the program might use the previous value that was saved in temporary storage or a register.

Programmer response: Recompile the program without the Optimize option.

---

**EQA2421E**  
The assignment was not performed because the assigned value might not be used by the program, due to optimization.

Explanation: Results would be unreliable because the program might use the previous value that was saved in temporary storage or a register.

Programmer response: You can SET WARNING OFF to allow the update to take place or recompile the program without optimization.

---

**EQA2422E**  
A breakpoint cannot be set on this statement when the STORAGE runtime option is in effect. Remove STORAGE or set the breakpoint after the next LR R13,Rx instruction.

Explanation: When the STORAGE runtime option is in effect, breakpoints are not allowed on the prologue instructions between the first BALR R14,R15 and the next LR R13,Rx. You may set a breakpoint on an instruction following the next LR R13,Rx or you may rerun your program without the STORAGE runtime option and set a breakpoint on the specified statement.

---

**EQA2423S**  
A AbendCode ABEND occurred.

Explanation: The indicated System or User ABEND was detected.

Programmer response: Investigate the cause of the ABEND.
### EQA2424I
Debug Tool detected a missing or invalid Debug Tool SVC EQA01SVC(IGX00051). The installed version is \textit{vers}. DYNDEBUG is disabled for read only programs.

**Explanation:** During initialization Debug Tool did not find a usable Dynamic Debug facility SVCs EQA00SVC(IGC0014E) and EQA01SVC(IGX00051). Either the SVC was not found or a downlevel version was detected. The Dynamic Debug facility is disabled for read only programs. Other Dynamic Debug facilities may not operate correctly.

**Programmer response:** Have your installer install the correct Dynamic Debug facility SVC.

### EQA2425I
Debug Tool detected a down level Debug Tool SVC EQA01SVC(IGX00051). The installed version is \textit{vers}. Version 2 is required when using CICS with TRANISO=YES. DYNDEBUG is disabled.

**Explanation:** During initialization Debug Tool detected a downlevel Dynamic Debug facility SVC version. The Dynamic Debug facility is disabled for read only programs.

**Programmer response:** Have your installer install the correct Dynamic Debug facility SVC.

### EQA2426I
Debug Tool detected a down level Debug Tool SVC EQA01SVC(IGX00051). The installed version is \textit{vers}. Version 8 is required for this version of Debug Tool.

**Explanation:** During initialization Debug Tool detected a downlevel Dynamic Debug facility SVC version.

**Programmer response:** Have your installer install the correct Dynamic Debug facility SVC.

### EQA2427I
Debug Tool detected a down level Debug Tool SVC EQA01SVC(IGX00051) or EQA00SVC(IGC0014E). EQA01SVC is version \textit{VERS}. EQA00SVC is version \textit{VERS}. EQA01SVC version 6 and EQA00SVC version 5 are required.

**Explanation:** During initialization Debug Tool detected a downlevel Dynamic Debug facility SVC version.

**Programmer response:** Have your installer install the correct Dynamic Debug facility SVC.

---

### EQA2428E
This command is either invalid or unsupported.

### EQA2429E
This command is not supported.

### EQA2430E
This command is not supported with this UI.

### EQA2431I
Automonitor is on for this debug session.

### EQA2432I
Automonitor is off.

---

### EQA2433E
Load Debug Data failed 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 Debug Tool Version 10 Release 1 Mod 0

**Explanation:** This message is used to place the Debug Tool logo, a time stamp, and copyright at the beginning of the line.

---

### EQA2442I
5655-V50: Copyright IBM Corp. 1992, 2009

**Explanation:** This message is used to place the Debug Tool logo, a time stamp, and copyright at the beginning of the line.

### EQA2443I
Assembler debug mode is active.

### EQA2444I
Assembler debug mode is no longer active.

### EQA2445I
Disassembly debug mode is now active.

### EQA2446I
Disassembly debug mode is no longer active.

### EQA2447I
The setting of LOG is on.

### EQA2448I
The setting of LOG is off.

### EQA2449I
Dynamic debug mode is on.

### EQA2450I
Dynamic debug mode is off.

### EQA2451I
The setting of WARNING is on.

### EQA2452I
The setting of WARNING is off.

### EQA2453E
SET DYNDEBUG can not be executed at this time. SET DYNDEBUG can only be executed at the beginning of a debugging session, before any STEP or GO commands. The DYNDEBUG status has not been changed.
This CU is not AUTOMONITOR capable for expressions.

*Programmer response:* Refer to the description of the SET AUTOMONITOR command in the [Debug Tool Reference and Messages](#) document to determine the requirements this CU must fulfill in order to use the SET AUTOMONITOR command.

Program uses non-standard linkage. R13 contains an invalid address.

*Explanation:* Debug Tool has stopped in a program and the value in GPR 13 is not a valid address.

*Programmer response:* Debug Tool will attempt to continue. However, some information may be missing or incorrect.

Returning from enclave where Debug Tool was initialized. Handling of non-LE events has been suspended. Debugging of non-LE programs is suspended in this Debug Tool session.

*Explanation:* The Language Environment was invoked with TEST/NOPROMPT or NOTEST causing Debug Tool to be initialized during an enclave that was not the top enclave. The enclave in which Debug Tool was initialize is now terminating. Debug Tool will no longer intercept non-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, you must terminate the prior use of SVC SCREENING (TCBSVCS, TCBSVCSP, TCBSVCA2) before starting Debug Tool or have your installer provide an EQAOPTS that specifies CONFLICT=OVERRIDE. CONFLICT=OVERRIDE allows Debug Tool to save and restore the previous use of SVC SCREENING (TCBSVCS, TCBSVCSP, TCBSVCA2).

**SVC Screening is in use by another product and SVC Screening CONFLICT=OVERRIDE is specified by EQAOPTS. Previous use of SVC Screening will be restored at the end of this Debug Tool session.**

*Explanation:* The Debug Tool was invoked with an EQAOPTS options module that specified CONFLICT=OVERRIDE. Debug Tool will save and restore the SVC Screening values.

SVC SCREENING is indicated by TCBSVCS, TCBSVCSP, and TCBSVCA2. These values are saved during Debug Tool startup and restored at Debug Tool termination.

Code page in the VADSCP suboption is not between 00001 and 99999. Default code page 00037 is assumed.

*Explanation:* Debug Tool was invoked with an invalid VADSCP suboption in the TEST runtime option string. Internal conversion tables for the default code page 00037 are used for translation between Debug Tool and the distributed debugger.

*Programmer response:* Correct the VADSCP suboption and restart the debug session.

**Code page conversion of string failed. z/OS Unicode conversion services return code is VERsand 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.


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.
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.

In order to be Language Environment compliant your application needs to follow the Language Environment rules.

If you need this support, ensure that the $ssss subsystem is running on the system that you are running Debug Tool on.

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.

The savearea has become Language Environment compliant and all commands will work properly.

Follow the Language Environment rules.

The following & ampersands names are currently & ampersands by & ampersands.

This message precedes the output of the NAMES command and indicates the type of names that follow this message.

& ampersands

This message lists the names output of the NAMES command.

The EQALANGX debug data also contains data for the following CUs:

Used to display CSECTs also included in the EQALANGX data for the current CU.

The SVC Screening required for Debug Tool will be merged with the SVC Screening already active for COPE.

COPE's usage of SVC Screening will be restored at the end of this Debug Tool session. Debug Tool was invoked with an EQAOPTS options module that specified MERGE. Debug Tool will save the COPE screening values on entry, merge them with Debug Tool's during the execution of the program, and then restores the original COPE SVC Screening values when Debug Tool terminates. SVC Screening is indicated by TCBSVCS, TCBSVCS2, and TCBSVCA2.

If you did not intend to MERGE the Debug Tool SVC Screening tables with another program, modify your EQAOPTS to specify what you require.

Your Language Environment enabled application has stopped at a location where no Language Environment compliant R13 savearea backchain exists. At location LOCN. Commands such as LIST CALLS will not operate properly.

In order to be Language Environment compliant your application needs to follow the Language Environment rules.

Debug Tool could not stop at one or more instructions because the program is executing with a PSW that specifies AMODE 64. At location LOCN.

During execution of the program Debug Tool encountered a hook or other event when the PSW specified AMODE64. Debug Tool will ignore all events that occur in this state and the program will continue to execute.

An implicit CU was created for "$& ampersands& ampersands" in "$& ampersands& ampersands".

An implicit 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.

An implicit CLEAR LOAD was issued for module "$& ampersands& ampersands".

An implicit CLEAR 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.

An implicit CLEAR LOAD was issued for module "$& ampersands& ampersands".

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.
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 Debug Tool error has occurred processing an assembler expression.
Programmer response: Report this error to your IBM representative.

EQA2486S  The source and receiver are not compatible for assignment.
Explanation: An assembler assignment contain a source and receiver that are not compatible for assignment. For example, a string longer than four bytes cannot be assigned to an arithmetic receiver.
Programmer response: Correct the assignment operands and retry the command.

EQA2487S  A runtime error occurred processing an assignment statement. The source or receiver may not be addressable.
Explanation: An error such as an addressing error occurred while Debug Tool was attempting to execute an assignment. Either the source or receiver may be incorrectly specified.
Programmer response: Report this error to your IBM representative.

EQA2488W  An LDD command was executed for a Disassembler CU that contained user breakpoints. These breakpoints have been removed.
Explanation: The Disassembler breakpoints no longer apply to an assembler program. The user should reapply any required breakpoints.
Programmer response: Set appropriate breakpoints.

EQA2489E  Symbol is an ambiguous or incorrectly qualified reference.
Explanation: The specified symbol was used in a non-Language Environment COBOL expression. However, either there are more than one variable by this name and the reference was not sufficiently qualified to disambiguate the reference or the qualification specified was invalid.
Programmer response: Correctly qualify the referenced variable and retry the command.

EQA2490I  The DEFAULT VIEW is now &&&.
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 Debug Tool initialization is complete.

EQA2499I  A load module that was loaded as the result of a LOAD command contains LE CUs but LE is not active. The LE CUs will not be created.
Explanation: A LOAD command was used in a non-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.

EQA2501I  DTCN internal error
Explanation: DTCN discovered an internal error.
Programmer response: Contact IBM service.

EQA2502E  Internal CICS error
Explanation: During processing, DTCN discovered an internal CICS error.
Programmer response: Correct the error and issue the command again. If the error persists, contact your CICS system programmer and/or IBM service.

EQA2503E  Key Not Defined.
Explanation: There is no action defined with the PF key used by the user.
Programmer response: Use the keys displayed in the bottom line. For more information about the actions defined for this panel, use PF2 key for general help.

EQA2504E  Add failed - profile exists
Explanation: The add command failed because a profile for that terminal is already stored in the Debug Tool Profile Repository.
Programmer response: You can use Show(PF7) command to display the profile or modify the TermId+TranId and Add a new profile.

EQA2505E  Replace failed - profile does not exist
Explanation: The profile for that terminal does not exist in the Debug Tool Profile Repository and cannot be updated. Specify different terminal to update.
Programmer response: You can use Next(PF8) command to browse the Profile Repository starting from any point.

EQA2506E  Delete failed - profile does not exist
Explanation: The profile for the terminal does not exist in the Debug Tool Profile Repository.
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: There are no more profiles in the Debug Tool Profile Repository.

EQA2510I  DTCN closed
Explanation: DTCN deleted all profiles stored in the Debug Tool 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.
### Debug Tool V10 Reference and Messages

#### EQA2514I
**Profile saved**
Explanation: A profile was saved in the Debug Tool Profile Repository.

#### EQA2515I
**Profile replaced**
Explanation: Existing profile was updated in the Debug Tool Profile Repository.

#### EQA2516I
**Profile deleted**
Explanation: Existing profile was deleted from the Debug Tool Profile Repository.

#### EQA2517I
**Profile not saved. Press PF4, or PF3 again to exit without saving.**
Explanation: PF3 has been pressed, but the new profile has not been saved in the repository.

**Programmer response:** Press PF4 to save the profile in the repository, or press PF3 again to exit from DTCN without saving the new profile.

#### EQA2518I
**Duplicate profile exists. Specify additional debug resources.**
Explanation: An attempt has been made to save a profile in the DTCN repository, but its debug resources match an existing profile.

**Programmer response:** Provide additional resource IDs to qualify your debugging needs better.

#### EQA2519E
**Site rules require that this field be filled in.**
Explanation: Your site has specified in its EQAOPTS member that this field must be filled in. For more information, refer to DTCNFORCEOxxxxxx 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.

#### EQA2525E
**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: Columnar format in the Monitor window can be displayed only if the width of Monitor window is bigger than 36 characters.

**Programmer response:** To change the Monitor window size, you need first change the width of the Monitor COLUMN OFF.

#### EQA2603E
**The width of the Monitor window is less than 36 characters. SET MONITOR COLUMN ON is not allowed.**
Explanation: Debug Tool 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.

#### EQA2604E
**The channel channel_name was not found.**
Explanation: Debug Tool 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.

#### EQA2605E
**There are no current 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:** Debug Tool 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:** Debug Tool 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.

**EQA2631E** Invalid character found in an address field.

**Explanation:** Debug Tool 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 $ or @ commands in those areas.

**Programmer response:** Type in hexadecimal characters, the $ or @ commands, or clear any characters you might have accidentally entered. Then press Enter.

**EQA2632E** Invalid character found in a data field.

**Explanation:** Debug Tool found an invalid character in the data column of the Memory window. You can put only hexadecimal characters or the $ or @ commands in those areas.

**Programmer response:** Type in hexadecimal characters, the $ or @ commands, or clear any characters you might have accidentally entered. Then press Enter.

**EQA2633E** Invalid character found in a history entry field.

**Explanation:** Debug Tool found an invalid character in the history entry field of the Memory window. You can put only hexadecimal characters or the $ or @ commands in those areas.

**Programmer response:** Type in hexadecimal characters, the $ or @ commands, or clear any characters you might have accidentally entered. Then press Enter.
<table>
<thead>
<tr>
<th>EQA2634E</th>
<th>Multiple changes found in a history entry field.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong></td>
<td>Debug Tool found multiple changes in a history entry field of the Memory window. You can only enter one command at a time.</td>
</tr>
<tr>
<td><strong>Programmer response:</strong></td>
<td>Clear the extra characters, then press Enter.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA2635E</th>
<th>The FIND command is not valid in the Memory window.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong></td>
<td>You cannot use the FIND command in the Memory window.</td>
</tr>
<tr>
<td><strong>Programmer response:</strong></td>
<td>Do not use the FIND command in the Memory window.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA2636E</th>
<th>Invalid scroll amount is specified for the Memory window.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong></td>
<td>You cannot use any one of the following scroll amounts for the Memory window: TOP, BOTTOM, MAX, and TO.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA2637E</th>
<th>Invalid Memory window width.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong></td>
<td>The width of the physical window assigned to the Memory window is less than the full screen width.</td>
</tr>
</tbody>
</table>
| **Programmer response:** | Do one of the following:  
  • 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. |

<table>
<thead>
<tr>
<th>EQA2638E</th>
<th>Invalid Amode value is specified.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong></td>
<td>The valid Amode values are 24 and 31.</td>
</tr>
<tr>
<td><strong>Programmer response:</strong></td>
<td>Enter a valid Amode value.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA2639E</th>
<th>SCROLL LEFT is not valid in the Memory window.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong></td>
<td>You cannot enter the SCROLL LEFT command in the Memory window. The Memory window displays memory content in the entire width of the window.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EQA2640E</th>
<th>The .dbg file for compile_unit_name, created by the FORMAT(DWARF) suboption of the DEBUG compiler option can not be found. Use of the SET SOURCE command to indicate the location of the source file is not allowed at this time.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Explanation:</strong></td>
<td>A .dbg file containing the debugging tables is not available.</td>
</tr>
<tr>
<td><strong>Programmer response:</strong></td>
<td>Make sure the .dbg file for the program is available.</td>
</tr>
</tbody>
</table>
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`, can not 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 can not 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`, can not 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.

**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 can not 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 can not 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 can not 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`.

---

**EQA3001I** .

**EQA3002I** ( )

**EQA3003I** Licensed Materials - Property of IBM

**EQA3004I** Copyright IBM Corp.

**EQA3005I** US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract

**EQA3006I** with IBM Corporation.

---

Chapter 8. Debug Tool messages 359
<table>
<thead>
<tr>
<th>Message Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQA3009I</td>
<td>** Opened File List</td>
</tr>
<tr>
<td>EQA3010I</td>
<td></td>
</tr>
<tr>
<td>EQA3011I</td>
<td>** End Opened File List</td>
</tr>
<tr>
<td>EQA3012I</td>
<td>** Job Search List</td>
</tr>
<tr>
<td>EQA3013I</td>
<td>** End Job Search List</td>
</tr>
<tr>
<td>EQA3014I</td>
<td>** Default Search List</td>
</tr>
<tr>
<td>EQA3015I</td>
<td>** End Default Search List</td>
</tr>
<tr>
<td>EQA3016I</td>
<td>** Referenced List</td>
</tr>
<tr>
<td>EQA3017I</td>
<td></td>
</tr>
<tr>
<td>EQA3018I</td>
<td>** End Referenced List</td>
</tr>
<tr>
<td>EQA3019E</td>
<td>Container name exceeds maximum</td>
</tr>
<tr>
<td>EQA3020E</td>
<td>File name exceeds maximum</td>
</tr>
<tr>
<td>EQA3021E</td>
<td>Open failure on DD:JCLLIB, specifies default concatenation list</td>
</tr>
<tr>
<td>EQA3022E</td>
<td>Number of containers exceeds default concatenation limit</td>
</tr>
<tr>
<td>EQA3023E</td>
<td>Container name read from file too long</td>
</tr>
<tr>
<td>EQA3024E</td>
<td>Number of containers on JCLLIB exceeds limit</td>
</tr>
<tr>
<td>EQA3025E</td>
<td>Attempt to nest source too deeply</td>
</tr>
<tr>
<td>EQA3026E</td>
<td>Number of source files used exceeds limit</td>
</tr>
<tr>
<td>EQA3027E</td>
<td>File name for INCLUDE or external PROC name is too long</td>
</tr>
<tr>
<td>EQA3028E</td>
<td>INCLUDE file or cataloged PROC not found: name</td>
</tr>
<tr>
<td>EQA3030E</td>
<td>Container Name parameter is not valid</td>
</tr>
<tr>
<td>EQA3031E</td>
<td>File Name parameter is not valid</td>
</tr>
<tr>
<td>EQA3032E</td>
<td>Open failure on DD:INFILE, specifies JCL file</td>
</tr>
<tr>
<td>EQA3033E</td>
<td>Comment continuation flag was not followed by valid card</td>
</tr>
<tr>
<td>EQA3034E</td>
<td>End of file after comment continuation flag</td>
</tr>
<tr>
<td>EQA3038E</td>
<td>DLM= delimiter not two characters</td>
</tr>
<tr>
<td>EQA3039E</td>
<td>DLM= parameter on DD statement &lt; 2 chars.</td>
</tr>
<tr>
<td>EQA3040W</td>
<td>DLM= contains &amp; not followed by &amp;</td>
</tr>
<tr>
<td>EQA3043E</td>
<td>Illegal statement found in PROC definition</td>
</tr>
<tr>
<td>EQA3046E</td>
<td>Illegal statement within include group</td>
</tr>
<tr>
<td>EQA3048E</td>
<td>PEND without matching procedure invocation</td>
</tr>
<tr>
<td>EQA3050E</td>
<td>Memory allocation failure</td>
</tr>
<tr>
<td>EQA3054E</td>
<td>Includes nested too deeply</td>
</tr>
<tr>
<td>EQA3058E</td>
<td>Symbolic name too long</td>
</tr>
<tr>
<td>EQA3059E</td>
<td>Multiple JOB cards</td>
</tr>
<tr>
<td>EQA3062E</td>
<td>Too many pars</td>
</tr>
<tr>
<td>EQA3063E</td>
<td>Parenthesis and apostrophe nesting exceeded</td>
</tr>
<tr>
<td>EQA3065E</td>
<td>If nesting exceeds limit</td>
</tr>
<tr>
<td>EQA3066E</td>
<td>ENDIF without IF</td>
</tr>
<tr>
<td>EQA3067E</td>
<td>Keyword parameter or SET statement missing =</td>
</tr>
<tr>
<td>EQA3068E</td>
<td>JCLLIB missing ORDER=</td>
</tr>
<tr>
<td>EQA3070E</td>
<td>EXEC card has null first parameter</td>
</tr>
<tr>
<td>EQA3071E</td>
<td>EXEC card PGM= has null value</td>
</tr>
<tr>
<td>EQA3072E</td>
<td>DD concatenation without DD statement</td>
</tr>
<tr>
<td>EQA3073E</td>
<td>Missing JOB card</td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EQA3074E</td>
<td>DD after JOB not JOBLIB or JOBCAT or PROCLIB</td>
</tr>
<tr>
<td>EQA3075E</td>
<td>Multiple JOBCAT statements</td>
</tr>
<tr>
<td>EQA3076E</td>
<td>Multiple JOBLIB statements</td>
</tr>
<tr>
<td>EQA3078E</td>
<td>Illegal statement in procedure</td>
</tr>
<tr>
<td>EQA3079E</td>
<td>PROC nesting exceeded limit</td>
</tr>
<tr>
<td>EQA3080E</td>
<td>Query empty context stack</td>
</tr>
<tr>
<td>EQA3081E</td>
<td>PROC stack underflow</td>
</tr>
<tr>
<td>EQA3082I</td>
<td>Statement:</td>
</tr>
<tr>
<td>EQA3083I</td>
<td>Substitution:</td>
</tr>
<tr>
<td>EQA3084E</td>
<td>Too many steps in job</td>
</tr>
<tr>
<td>EQA3086E</td>
<td>StepName.DDName too long</td>
</tr>
<tr>
<td>EQA3087E</td>
<td>Could not find step named on override DD</td>
</tr>
<tr>
<td>EQA3088E</td>
<td>Could not add DD; no PGM= steps found</td>
</tr>
<tr>
<td>EQA3089I</td>
<td>Added to Proc:</td>
</tr>
<tr>
<td>EQA3090I</td>
<td>Override Proc:</td>
</tr>
<tr>
<td>EQA3091E</td>
<td>Fatal Error:</td>
</tr>
<tr>
<td>EQA3092E</td>
<td>Statement label is too long</td>
</tr>
<tr>
<td>EQA3093E</td>
<td>Statement parameter text is too long</td>
</tr>
<tr>
<td>EQA3095E</td>
<td>Instream PROC has no name</td>
</tr>
<tr>
<td>EQA3096E</td>
<td>Instream PROC PEND not found before end of file</td>
</tr>
<tr>
<td>EQA3097I</td>
<td>Input parameters:</td>
</tr>
<tr>
<td>EQA3098E</td>
<td>Internal error</td>
</tr>
<tr>
<td>EQA3100W</td>
<td>Could not substitute DB2 run unit for SYSTSIN</td>
</tr>
<tr>
<td>EQA3101W</td>
<td>Could not open</td>
</tr>
<tr>
<td>EQA3102W</td>
<td>Could not find OUTPUT spec named!</td>
</tr>
<tr>
<td>EQA3103W</td>
<td>Could not open DD DDITV02 for DB2/IMS</td>
</tr>
<tr>
<td>EQA3104W</td>
<td>Could not read DD DDITV02</td>
</tr>
<tr>
<td>EQA3105W</td>
<td>Could not substitute DB2 run unit for DSNMVT01</td>
</tr>
<tr>
<td>EQA3108W</td>
<td>Could not find IMS program name</td>
</tr>
<tr>
<td>EQA3109W</td>
<td>Could not find IMS PSB name!</td>
</tr>
<tr>
<td>EQA3110E</td>
<td>Internal error</td>
</tr>
<tr>
<td>EQA3111E</td>
<td>Override not proceeded by named DD stmt</td>
</tr>
<tr>
<td>EQA3119E</td>
<td>Data set name is too long</td>
</tr>
<tr>
<td>EQA3120I</td>
<td>5655-V50: Debug Tool for z/OS</td>
</tr>
<tr>
<td>EQA3130I</td>
<td>Warning found in line</td>
</tr>
<tr>
<td>EQA3131I</td>
<td>Warning found in line</td>
</tr>
<tr>
<td>EQA4000E</td>
<td>The length of CSECT CsectName in memory is X’LMLength’ which does not match the length of X’LangXLength’ found in the EQALANGX data</td>
</tr>
<tr>
<td></td>
<td>Explanation: The EQALANGX data does not correspond to the CSECT in the loaded load module because the CSECT length does not match.</td>
</tr>
<tr>
<td></td>
<td>Programmer response: Regenerate the EQALANGX data or ensure that the matching object has been link-edited into the current load module.</td>
</tr>
<tr>
<td>EQA4001E</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Programmer response: Regenerate the EQALANGX data or ensure that the matching object has been link-edited into the current load module.</td>
</tr>
<tr>
<td>EQA4002E</td>
<td>The EQALANGX data for language CSECT CsectName is for a different programming language.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
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 QUER'ed 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 CeeCpiTest-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. Debug Tool 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).

EQA4015E 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: Debug Tool 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, Debug Tool 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: Debug Tool 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: Debug Tool 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 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: Debug Tool 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: Debug Tool 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. Debug Tool 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, Debug Tool 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 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.

EQA4022I 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.

EQA4023I 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.

EQA4024I The expression or variable & & & & & & is not valid, undefined, or not correctly qualified.

Explanation: Debug Tool 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.

EQA4025I & & & & & &

Explanation: This message contains the output from the DESCRIBE LOADMOS command.

EQA4026E Load module loadmod_name could not be found.

Explanation: The indicated load module was specified as an operand of the DESCRIBE LOADMOS 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.

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.

EQA4045E You can not 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 • EQA9929E

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.

EQA4700E A parse error was detected by the z/OS XML System Services parser: Return Code=x, Reason Code=X'raison', 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=1,context1
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 cursor1
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: Debug Tool 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.

EQA4706I Its address is &;&;& and its length is &;&.&.
Explanation: Text of a DESCRIBE CU message.

EQA9924U The current user does not have RACF access to use Debug Tool.
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.
Programmer response: Have the person controlling the RACF Facility in effect give your user-id a RACF access of READ or higher.

EQA9925I C/C++ does not support browse mode.
Explanation: An attempt was made to execute a C or C++ program in browse mode. This is not currently supported. The Debug Tool session will Abend with a U1991 Abend.
Programmer response: Do not attempt to execute C or C++ programs in browse mode.

EQA9926I ***Allocate attempted from OPEN exit.
Explanation: Debug Tool 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 Debug Tool.
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 Debug Tool support.
Programmer response: Report this message to Debug Tool support.

EQA9928W ***No ESPIE with CeeCaaXHCL on
Explanation: Debug Tool was entered for an SVC or overlay hook after Language Environment routines had set the CeeCaaXHCL flag but Debug Tool was not able to establish and ESPIE.
Programmer response: If this message is followed by unexpected Debug Tool behavior, report the message to Debug Tool support.

EQA9929E Debug Tool failed Product Registration. IFAEDREG RC = 00000004. The enclave will be terminated.
Explanation: Debug Tool called IFAEDREG to check for an enabled product registration for Debug Tool. IFAEDREG indicated that the check did not succeed.
Programmer response: If Debug Tool is licensed to run on this
machine, the system programmer should register and enable the product registration per the appropriate Program Directory.

---

**EQA9930W**  
**Explanation:** Debug Tool 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**  
**Explanation:** The requested user interface is not available so Debug Tool will quit processing events and the application will continue to run. (Note that any calls to restart Debug Tool are ignored.) This behavior was specified in the EQAOPTS customization module via the EQAXOPT macro invocation option NO displaY, QUIT DEBUG.

**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 NO displaY, 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 Debug Tool Terminal Interface Manager.

**Programmer response:** Use the Debug Tool 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**  
**Debug Tool EQA00CIC: Error loading Program. See **.SEQASAMP(EQA0ACCDS)**

**Explanation:** Debug Tool Program EQA00CIC in load module EQA00OSX was unable to load the specified program.

**Programmer response:** Ensure that the group (EQA) that contains the Debug Tool run time routines is in the group list used during CICS start-up. If required, rerun the EQACCSD job and restart the CICS region. Check the Debug Tool website for any applicable updates your system might require. If the problem persists, contact Debug Tool support.

---

**EQA9935E**  
**Explanation:** Debug Tool has experienced a problem, and is reporting diagnostic information (usually return codes given to Debug Tool by a subsystem.)

**Programmer response:** Check the log for further diagnostic messages. If there is no obvious cause for the problem, contact IBM Support.

---

**EQA9936I**  
**EQA00CIC Bad response from EXEC CICS cmd.**

**Explanation:** Debug Tool has issued an EXEC CICS command and has received an unexpected response.

**Programmer response:** Check the log for further diagnostic messages. If there is no obvious cause for the problem, contact IBM Support.

---

**EQA9937W**  
**XPCFTCH MEA conflict- XXXXXXXX: YYYYYYYY**

**Explanation:** Debug Tool is reporting that another CICS XPCFTCH global user exit has set a modified entry address (MEA) and prevented Debug Tool from any possible debugging of a specific non-Language Environment program. XXXXXXXX can be either: 'Prior MEA' or 'Program' or 'Transid' and YYYYYYYY is the data associated with each. Note this message is only issued once when this occurs the first time after the Debug Tool CICS exits are activated. Subsequent conflicts are not written to the CICS JES log.

**Programmer response:** Multiple XPCFTCH exits running in the same CICS region which can each set the MEA and return to CICS is usually not recommended. For Debug Tool, you will be unable to debug any non-Language Environment programs when the MEA was changed by another XPCFTCH exit. The behavior of Debug Tool in this kind of scenario will likely be unpredictable.

---

**EQA9938E**  
**Error in deactivate of NewProg exits.**

**Explanation:** Debug Tool detected an error in attempting to deactivate the NewProg supporting exits.

**Programmer response:** An error has likely occurred during Debug Tool CICS region initialization. Ensure that Debug Tool is properly installed in the CICS region. Also verify that the Debug Tool h1q.SEQAMOD data set is in the region DFHRPL DD and the CICS resource definitions from h1q.SEQAmp(EQA0CCLD) have been added.

---

**EQA9939I**  
**IBM Debug Tool NewProg support deactivated.**

**Explanation:** Debug Tool is reporting that NewProg support is now disabled in the current CICS region after a DTCP transaction was issued with the ‘F’ parameter. This support is to allow multi-region CICS configurations (for example, TOR/AOR), where DTCP is used, to work properly when DTCP 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 DTCP does not run.

---

**EQA9940I**  
**IBM Debug Tool NonLE exits enabled.**

**Explanation:** Debug Tool is reporting that the non-Language Environment-supporting CICS exits are now enabled in the current CICS region. This was accomplished by using PLTPI program EQA0CPLT and starting with INITPARM=EQA0CPLT=*NLE*.

---

Chapter 8. Debug Tool messages 365
EQA9941I • EQA9951E

EQA9941I IBM Debug Tool NewProg support activated.
Explanation: Debug Tool is reporting that NewProg support is now enabled in the current CICS region. This support is to allow multi-region CICS configurations (for example, TOR/AOR), where DTCN is used, to work properly when DTCN is executed in one region (TOR) and tasks to be debugged might be routed to an alternate region (AOR). This is only required in the regions where DTCN does not execute.

EQA9942I IBM Debug Tool Screen stack exits enabled.
Explanation: Debug Tool is reporting that its single-terminal mode screen stacking exits are now enabled. This is to support installations where starting CICS exits is restricted by an external security manager (for example, RACF) and prevents Debug Tool from starting the exits when it starts a debug session for a user. This was accomplished using PLTPI program EQAOCLT and starting with INITPARM=(EQAOCLT='STK').

EQA9943E Error in activate of NonLE exits.
Explanation: Debug Tool detected an error while attempting to activate the non-Language Environment supporting exits.
Programmer response: The error most likely occurred during Debug Tool CICS region initialization. Verify that the Debug Tool hlq.SEQAMOD data set is the in the region DFHRPL and the CICS resource definitions from hlq.SEQASAMP(EQACCSD) have been added.

EQA9944E Error in activate of NewProg exits.
Explanation: Debug Tool detected an error in attempting to activate the NewProg supporting exits.
Programmer response: An error has likely occurred during Debug Tool CICS region initialization. Ensure that Language Environment is properly installed in the CICS region. Also verify that the Debug Tool hlq.SEQAMOD data set is in the region DFHRPL DD and the CICS resource definitions from hlq.SEQASAMP(EQACCSD) have been added.

EQA9945S Debug Tool DTRCB Unavailable.
Explanation: The Debug Tool non-Language Environment CICS global user exits were made active in a CICS region where Debug Tool did not successfully initialize during CICS region startup.
Programmer response: Ensure that Language Environment is installed in the CICS region and verify that the Tool installation steps were executed properly. For example, the Debug Tool hlq.SEQAMOD data set is in the DFHRPL DD and that the hlq.SEQASAMP(EQACCSD) job was run to add Debug Tool resource definitions to the CICS region.

EQA9946S EQA015VC TCBSVCA2 invalid - xxx where xxx=start, stop, term, strX, stopX
Explanation: Internal Debug Tool SVC Screening error or Debug Tool SVC (109 extended code 51) issued outside Debug Tool. The SVC will abend.
Programmer response: If using Debug Tool contact your IBM representative.

EQA9947S EQA015VC EQASVCSCREEN N/T create
Explanation: Internal Debug Tool SVC Screening error or Debug Tool SVC (109 extended code 51) issued outside Debug Tool. The SVC will abend.
Programmer response: If using Debug Tool contact your IBM representative.

EQA9948S EQA015VC No DTRCB at InitScreen
Explanation: Internal Debug Tool SVC Screening error or Debug Tool SVC (109 extended code 51) issued outside Debug Tool. The SVC will abend.
Programmer response: If using Debug Tool contact your IBM representative.

EQA9949S or EQA9949I TCB SVC Screening already active and NOOVERRIDE is specified by EQAOPTS.
Explanation for Severe error: SVC Screening is in use by another product and SVC Screening CONFLICT=NOOVERRIDE is specified by EQAOPTS. Handling of non-Language Environment events is not available. Debugging of non-Language Environment programs will be restricted in this Debug Tool session.
Explanation for Informational message: The Debug Tool was invoked with an EQAOPTS options module that specified CONFLICT=NOOVERRIDE. Debug Tool will not intercept non-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 Debug Tool or have your installer provide an EQAOPTS that specified CONFLICT=OVERRIDE. CONFLICT=OVERRIDE allow Debug Tool to save and restore the previous use of SVC SCREENING (TCBSVCS, TCBSVCSP, TCBSVCA2).

EQA9950E Error enabling XEIHIN screen exit.
Explanation: Debug Tool detected an error during the ENABLE of a required CICS exit program.
Programmer response: Determine if the Debug Tool hlq.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 XEIOOUT screen exit.
Explanation: Debug Tool detected an error during the ENABLE of a required CICS exit program.
Programmer response: Determine if the Debug Tool hlq.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 Debug Tool RCB.
Explanation: Debug Tool CICS PLT program EQA0CPLT detected an error during the search for Debug Tool region-level resources.
Programmer response: An error has likely occurred during Debug Tool CICS region initialization. Ensure that Language Environment is properly installed in the CICS region. Also, verify that the Debug Tool hlq.SEQAM00 data set is in the region DFHRPL DD and the CICS resource definitions from hlq.SEQASAMP(EQACSD) have been added.

EQA9953E  NOTAUTH Error issuing CICS EXTRACT EXIT.
Explanation: Debug Tool detected a NOTAUTH condition during an EXTRACT EXIT call to CICS.
Programmer response: Determine if the current Debug Tool user has external security-manager (RACF) access to the EXITPROGRAM CICS CLASS. This includes the ability to issue the EXEC CICS EXTRACT/ENABLE/DISABLE EXIT commands. If this is not permitted, then use of Debug Tool PLT initialization routine, EQA0CPLT, is recommended. Refer to the Debug Tool Customization Guide or information on EQA0CPLT.

EQA9954E  Invalid EXIT ENABLE request.
Explanation: Debug Tool 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 Debug Tool screen-stack exits. Contact IBM Support Center and report the error.

EQA9955E  User not authorized for EXIT ENABLE.
Explanation: Debug Tool CICS PLT program 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: Debug Tool CICS PLT program EQA0CPLT detected an error during the ENABLE of a required CICS exit program.
Programmer response: Determine if the Debug Tool hlq.SEQAM00 library is available in the DFHRPL concatenation of the CICS region and the resource definitions provided in hlq.SEQASAMP(EQACSD) have been added to the CICS region that is initializing.

EQA9957E  Invalid CICS release. Latest CICS used.
Explanation: Debug Tool detected an unsupported release of CICS and defaults to the latest release of CICS that this version of Debug Tool supports.
Programmer response: Determine if Debug Tool is starting on a supported release of CICS. See the Debug Tool Program Directory for the list of CICS releases that are supported.

EQA9958I  IBM Debug Tool CICS PLT init start.
Explanation: Debug Tool program EQAICPPLT is starting. This program activates various Debug Tool resources during CICS region startup. This includes starting Debug Tool support for running in CICS multi-region configurations (INITPARM=(EQA0CPLT=’NMP’)) and starting Debug Tool screen stack exits once at region initialization (INITPARM=(EQA0CPLT=’STK’)) and starting Debug Tool non-Language Environment-supporting exits (INITPARM=(EQA0CPLT=’NLE’)). Combinations of these selections are also supported. For example: INITPARM=(EQA0CPLT=’NMP,STK,NLE’)

EQA9959I  IBM Debug Tool CICS PLT init end.
Explanation: Debug Tool program EQAICPPLT is ending. This program activates various Debug Tool resources during CICS region startup. This includes starting up Debug Tool support for running in CICS multi-region configurations (INITPARM=(EQA0CPLT=’NMP’)) and starting Debug Tool screen stack exits once at region initialization (INITPARM=(EQA0CPLT=’STK’)) and starting Debug Tool non-Language Environment-supporting exits (INITPARM=(EQA0CPLT=’NLE’)).

EQA9960I  Program abend: Abcode Prog: Abprogram Ret#: XXXXXXX
Explanation: Debug Tool has detected abend Abcode while processing program Abprogram under CICS. Ret# is the address of the location where the abend was issued.
Programmer response: This message occurs when the non-Language Environment Debug Tool CICS exits are active and an abend has occurred in the application currently being debugged. It is trapped as a result of the TRAP(ON) runtime option. The default behavior for the STEP or GO command at this time is for Debug Tool to display the abend and allow the task to terminate or allow any active CICS HANDLE abend routines to run or, if applicable, allow any Language Environment user handlers or signal catchers to run. Use the TRAP(OFF) runtime option if you do not want Debug Tool to capture abends. This message is written to the CICS region’s JES message log.

EQA9961I  Program interrupt: Intcd Prog: Abprogram Int#: XXXXXXX
Explanation: Debug Tool 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 Debug Tool CICS exits are active and a program check has occurred in the application currently being debugged. It is trapped as a result of the TRAP(ON) runtime option. The default behavior for the STEP or GO command at this time is for Debug Tool to display the abend and allow the task to terminate or allow any active CICS HANDLE abend routines to run or, if applicable, allow any Language Environment user handlers or signal catchers to run. Use the TRAP(OFF) runtime option if you do not want Debug Tool to capture program checks. This message is written to the CICS region’s JES message log.

EQA9962I  IBM Debug Tool Exit Activation PLT start.
Explanation: Debug Tool program EQANCPLT is starting. This program activates the Debug Tool non-Language Environment CICS global exits which must be executed as either a stage 2 or 3 PLT post initialization program.
<table>
<thead>
<tr>
<th>Message ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EQA9963I</strong></td>
<td>IBM Debug Tool Exit Activation PLT end.</td>
</tr>
<tr>
<td><strong>Explanation:</strong></td>
<td>Debug Tool program EQANCPLT is ending. This program activates the Debug Tool non-Language Environment CICS global exits which must be executed as either a stage 2 or 3 PLT post initialization program.</td>
</tr>
<tr>
<td><strong>Programmer response:</strong></td>
<td>Contact IBM support center and report the error.</td>
</tr>
</tbody>
</table>

| **EQA9964E** | Create EQADTA name/token error. RC: RC |
| **Explanation:** | Debug Tool is unable to initialize for a non-Language Environment assembler program under CICS. |
| **Programmer response:** | Verify that the latest version of the Debug Tool Dynamic Debug facility SVCs are installed. The level of the SVCs can be checked by running the exec in Tool Dynamic Debug facility SVCs are installed. The level of the Debug Tool is unable to initialize for a non-Language Environment assembler program under CICS. |

| **EQA9965E** | CEEDBGEVTEXT Error. RC: RC |
| **Explanation:** | Debug Tool is unable to initialize for a non-Language Environment assembler program under CICS. |
| **Programmer response:** | Contact IBM support center and report the error. If this message occurs repeatedly, disable the non-Language Environment CICS exits using transaction DTCX (DTCXXF) or by removing the EQANCPLT from the CICS PLT. |

| **EQA9966E** | Back-level Debug Tool SVC detected. V5R1 or later SVCs required for Non-LE support. |
| **Explanation:** | Debug Tool is unable to initialize for a non-Language Environment assembler program under CICS due to back-level Dynamic Debug facility SVCs. |
| **Programmer response:** | Verify that the latest version of the Debug Tool Dynamic Debug facility SVCs are installed. The level of the SVCs can be checked by running the exec in dtexitname,SEQAEXEC(EQADTSVC). |

| **EQA9967I** | EQA00SVC Level:Eqad00svcVersion EQA01SVC Level:Eqad01svcVersion |
| **Explanation:** | Debug Tool is unable to initialize for a non-Language Environment assembler program under CICS due to back-level Dynamic Debug facility SVCs. |
| **Programmer response:** | Verify that the latest version of the Debug Tool Dynamic Debug facility SVCs are installed. The level of the SVCs can be checked by running the exec in dtexitname,SEQAEXEC(EQADTSVC). For non-Language Environment support, EQA00SVC must be at least 04 and EQA01SVC at least 05. |

| **EQA9968E** | Invalid Exit Type ... |
| **Explanation:** | Debug Tool CICS exit activation transaction DTCX is unable to determine a valid exit type to start or stop. |
| **Programmer response:** | Verify that DTCX is issued with an exit type of X=all exits or F=XPCFTCH exit or E=E=XEIN or A=XPCTA or H=XPCCHAIR. 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:** | Debug Tool CICS exit activation transaction DTCX or Newprog activation transaction DTCP is unable to determine a valid action to take, O=ON or F=OFF. |
| **Programmer response:** | Re-enter the transaction with an O or F parameter where O=On and F=Off. |

| **EQA9970I** | CICS exit activation successful. |
| **Explanation:** | Debug Tool CICS global user exits activated successfully. |

| **EQA9971I** | CICS exit deactivation successful. |
| **Explanation:** | Debug Tool CICS global user exits deactivated successfully. |

| **EQA9972I** | DT glueexitname CICS exit now ON. |
| **Explanation:** | Debug Tool CICS exit activation transaction DTCX successfully started the glueexitname exit where glueexitname is either XPCFTCH, XEIIN, XEIOUT, XPCTA, or XPCHAIR. |

| **EQA9973I** | DT glueexitname CICS exit now OFF. |
| **Explanation:** | Debug Tool CICS exit activation transaction DTCX successfully stopped the glueexitname exit where glueexitname is either XPCFTCH, XEIIN, XEIOUT, XPCTA, or XPCHAIR. |

| **EQA9974E** | Error enabling glueexitname - dxexitname. |
| **Explanation:** | Debug Tool CICS exit activation transaction DTCX was unable to activate glueexitname - dxexitname where glueexitname is either XPCFTCH, XEIIN, XEIOUT, XPCTA, or XPCHAIR and dxexitname is either EQANCFTC, EQANCXEI, EQANCXAB, or EQANCXHA. |
| **Programmer response:** | Verify that the latest dtexitname,SEQAEXEC(EQADTSVC) CICS resource definitions are installed and the Debug Tool dtexitname,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 - dxexitname. |
| **Explanation:** | Debug Tool CICS exit activation transaction DTCX was unable to deactivate glueexitname - dxexitname where glueexitname is either XPCFTCH, XEIIN, XEIOUT, XPCTA, or XPCHAIR and dxexitname is either EQANCFTC, EQANCXEI, EQANCXAB, or EQANCXHA. |
| **Programmer response:** | Verify that the latest dtexitname,SEQAEXEC(EQADTSVC) CICS resource definitions are installed and the Debug Tool V5.1 dtexitname,SEQAMOD library is in the CICS DFHRPL DD concatenation. If this has already been done, contact IBM support center and report the error. |

| **EQA9976I** | DT 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  *disvcname is backlevel. Exits not enabled.*

**Explanation:** Debug Tool is unable to activate the non-Language Environment CICS global exits due to back-level Dynamic Debug facility SVCs. *disvcname* is either EQA08SVC or EQA01SVC.

**Programmer response:** Verify that the latest version of the Debug Tool Dynamic Debug facility SVCs are installed. The level of the SVCs can be checked by running the exec in dthlq.SEQAEXEC(EQA01SVC). For non-Language Environment support, EQA08SVC 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:** Debug Tool is unable to set a hook and stop in this program because the separate debug file cannot be located. This program was specified using a DTCN or CADP profile.

**Programmer response:** Verify that the debug data file exists and make its location known to Debug Tool by using the SET DEFAULT LISTINGS or SET SOURCE command, the 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:

- Debug Tool unrecoverable CICS task error.
- CICS abend code is *aaaa*
- Debug Tool 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 Debug Tool 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 Debug Tool storage. Check the Debug Tool web site for any applicable service updates your system might require. If the problem persists, contact Debug Tool support.

---

EQA9982E  A non-zero response code was returned from EXEC CICS "command". Resp value = EIBRESP

**Explanation:** Debug Tool 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 Debug Tool.

**Explanation:** The non-Language Environment version of Debug Tool (EQANMDBG) was invoked but the Debug Tool Dynamic Debug facility SVCs required for dynamic debug support have not been installed.

**Programmer response:** Have your system programmer complete the installation of the required Debug Tool Dynamic Debug facility SVCs.

---

EQA9986E  Error in CEEEV006 loading OSI

**Explanation:** One of the required Debug Tool load modules is missing.

**Programmer response:** Contact your system programmer to verify the proper installation of Debug Tool.

---

EQA9987I  First parameter to *load_module* is not addressable. Cannot debug this module.

**Explanation:** A non-Language Environment program issued a LINK SVC for the specified load module. However, an invalid address was specified for the first positional parameter that is used to specify Language Environment runtime parameters and user parameters. This parameter must be valid, so that Debug Tool can add the TEST parameter.

**Programmer response:** Correct the parameter address passed to the LINK SVC.

---

EQA9988S  Debug Tool has terminated the enclave.

**Explanation:** A Debug Tool QUIT command was issued in a multi-enclave environment.

**Programmer response:** No action is necessary.

---

EQA9989I  EQANMDBG requires Debug Tool V5R1 or later SVC.

**Explanation:** The Debug Tool V5R1 (or later) non-Language Environment program was invoked but the installed version of the Debug Tool Dynamic Debug facility SVCs were from a earlier version of Debug Tool.

**Programmer response:** Have your system programmer install the current V5R1 or later Debug Tool Dynamic Debug facility SVCs.
EQA9990I  LOAD detect.

Programmer response: This is an internal Debug Tool message. No user response is required.

EQA9991E Error loading load_module

Explanation: An error was encountered loading the specified load module.

Programmer response: Retry the Debug Tool session. Check the Debug Tool website for any applicable service updates your system might require. If the problem persists, contact Debug Tool support.

EQA9992E Internal error processing Language Environment service FunctionCode

Explanation: An internal Debug Tool error has occurred.

Programmer response: Check the Debug Tool website for any applicable service updates your system might require. If the problem persists, contact Debug Tool support.

EQA9993I EQA00OHT: Failed to set hook in R/O storage

Explanation: The Dynamic Debug facility was unable to successfully use the Authorized Debug facility to place a hook into an application that has been loaded into protected (read only) storage.

Debug Tool Action: If the application has been compiled with hooks (e.g., TEST(ALL,...)) then you will be able to debug this application. If the application has been compiled with TEST(NONE,...) then you will not be able to STEP or set breakpoints.

Programmer response: Ensure that both the Dynamic Debug facility and Authorized Debug facility have been activated. Ensure that you have the access through your security system to resource EQADTOOL.AUTHDEBUG in CLASS(FACILITY).

EQA9994E No storage for DT RCB

Explanation: There is insufficient storage for Debug Tool to initialize.

Programmer response: Increase the region size available to the program and rerun.

EQA9995E REQUIRED TEXT

Explanation: All EQA9995E messages signify a severe error has occurred in the Dynamic Debug facility SVC routine while processing an O91 instruction.

Programmer response:
1. Make sure none of the applications you are debugging issue the reserved O91 (SVC 145) instruction.
2. If you have non-IBM products installed on your system, make sure none of them issue the reserved O91 (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 EQAW.SEQASAMP.
4. Have your system support person re-install the Dynamic Debug facility SVC using member EQAWISVC of data set EQAW.SEQASAMP and then run the IVP (see step 5).
5. Check the Debug Tool website for any applicable service updates your system might require.

6. If the problem persists, report the error message text, return code, and reason code to your IBM representative.

EQA9996E ERROR DESCRIPTION

Explanation: A severe error has occurred in the Debug Tool Authorized Debug Facility SVC routine EQA01SVC. EQA01SVC is SVC 109 with extended function code 51.

Programmer response: Check the Debug Tool website for any applicable service updates your system might require. If the problem persists, report the error message text, return code, and reason code to your IBM representative.

EQA9997E ERROR DESCRIPTION

Explanation: The ASMADOP module could not be loaded. Debugging via a Disassembly View cannot be supported.

Programmer response:
1. Have your system support person verify that the ASMADOP module is available (in the search path).
2. Check the Debug Tool website for any applicable service updates your system might require.
3. If the problem persists, report the error message text to your IBM representative.

EQA9998I Debug Tool DTCN profile skipped.

Explanation: The Debug Tool profile has been skipped because a more qualified profile has been found or an older, equally qualified, profile has been found.

Programmer response: Additional EQA9998I messages will follow. See the details of these additional messages for the appropriate response.

EQA9999E ERROR DESCRIPTION

Explanation: Severe Internal Error in Debug Tool Module
Failure address - \_

Programmer response: See details of message issued for appropriate response or indication of potential problem. Check the Debug Tool website for any applicable service updates your system might require. If the problem persists, contact Debug Tool support.

EQA9999W ERROR DESCRIPTION

Explanation: Warning Message issued by Debug Tool Module

Programmer response: See details of message issued for appropriate response or indication of potential problem. Check the Debug Tool website for any applicable service updates your system might require.
Chapter 9. Non-Language Environment IMS messages

Each message has a number of the form EQAI\textsubscript{nnnx}, where EQAI indicates that the message is non-Language Environment IMS message, \textit{nnn} is the number of the message, and \textit{x} indicates the severity level of each message. The variable \textit{x} can be any of the following values:

- \textbf{I} An \textit{informational} message calls attention to some aspect of a command response that might assist the programmer.
- \textbf{W} A \textit{warning} message calls attention to a situation that might not be what is expected or to a situation that Debug Tool attempted to fix.
- \textbf{E} An \textit{error} message describes an error that Debug Tool detected or cannot fix.
- \textbf{S} A \textit{severe} error message describes an error that indicates a command referring to bad data, control blocks, program structure, or something similar.
- \textbf{U} An \textit{unrecoverable} error message describes an error that prevents Debug Tool from continuing.

Symbols in messages

Many of the Debug Tool messages contain information that is inserted by the system when the message is issued. In this publication, such inserted information is indicated by italicized symbols, as in the following:

\texttt{EQAI1046I The breakpoint-id breakpoint is replaced.}

The portion of Debug Tool located on the host notifies you of errors associated with debugging functions carried out by the host.

<table>
<thead>
<tr>
<th>Message ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAI1002S</td>
<td>CSVQUERY failure</td>
</tr>
<tr>
<td>Explanation:</td>
<td>Query application MPP failed.</td>
</tr>
<tr>
<td>User response:</td>
<td>Consult your system programmer.</td>
</tr>
<tr>
<td>EQAI1003S</td>
<td>Environment AIB call failed</td>
</tr>
<tr>
<td>Explanation:</td>
<td>Query environment AIB call failed.</td>
</tr>
<tr>
<td>User response:</td>
<td>Consult your system programmer.</td>
</tr>
<tr>
<td>EQAI1004S</td>
<td>Environment IOPCB call failed</td>
</tr>
<tr>
<td>Explanation:</td>
<td>Query environment IOPCB call failed.</td>
</tr>
<tr>
<td>User response:</td>
<td>Consult your system programmer.</td>
</tr>
<tr>
<td>EQAI1008S</td>
<td>Error loading EQANIDBG</td>
</tr>
<tr>
<td>Explanation:</td>
<td>Load Debug Tool non-Language Environment front end failed.</td>
</tr>
<tr>
<td>User response:</td>
<td>Consult your system programmer.</td>
</tr>
<tr>
<td>EQAI1009S</td>
<td>Invalid EQASET specification</td>
</tr>
<tr>
<td>Explanation:</td>
<td>Load Debug Tool non-Language Environment front end failed.</td>
</tr>
</tbody>
</table>

User response: Consult your system programmer.

<table>
<thead>
<tr>
<th>Message ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAI1013S</td>
<td>Retrieve of token failed with returncode</td>
</tr>
<tr>
<td>Explanation:</td>
<td>Fail to retrieve token (EQA\texttt{user_ID}). The return code is returncode.</td>
</tr>
<tr>
<td>User response:</td>
<td>Run the EQASET transaction from the terminal where the application transaction is invoked.</td>
</tr>
<tr>
<td>EQAI1020S</td>
<td>Retrieve token failed</td>
</tr>
<tr>
<td>Explanation:</td>
<td>Fail to retrieve token.</td>
</tr>
<tr>
<td>User response:</td>
<td>Start EQASET transaction with a valid keyword (MFI, TCP, VTCP, VTAM).</td>
</tr>
<tr>
<td>EQAI12005I</td>
<td>DEBUG SET ON FOR MFI SETTING</td>
</tr>
<tr>
<td>Explanation:</td>
<td>Debugging is turned on and setting is MFI.</td>
</tr>
<tr>
<td>User response:</td>
<td>None.</td>
</tr>
<tr>
<td>EQAI12006I</td>
<td>DEBUG SET ON FOR TCP SETTING</td>
</tr>
<tr>
<td>Explanation:</td>
<td>Debugging is turned on and setting is TCP.</td>
</tr>
<tr>
<td>User response:</td>
<td>None.</td>
</tr>
</tbody>
</table>
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 MFI LU name.
User response:  None.

EQAI2011I  VALID KEYWORDS:  ON, OFF, MFI=, TCP=, VTCP, VTAM=, STATUS
Explanation:  Valid keywords for EQASET transaction.
User response:  None.

EQAI2012E  USERID FROM IMS ENVIRONMENT IS BLANK
Explanation:  User ID is blank or cannot be found.
User response:  Contact your system programmer.

EQAI2013E  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.
Chapter 10. Load Module Analyzer Messages

All messages are shown in this section are in mixed case English. The uppercase English message text is the same, but is in uppercase letters.

Each message has a number of the form EQALMnnnx, where EQA indicates that the message is an Load Module Analyzer message, nnn is the number of the message, and x indicates the severity level of each message. The variable x can be any of the following values:

I An informational message calls attention to some aspect of a command response that might assist the programmer.

W A warning message calls attention to a situation that might not be what is expected or to a situation that Debug Tool attempted to fix.

E An error message describes an error that Load Module Analyzer detected or cannot fix.

S A severe error message describes an error that indicates a command referring to bad data, control blocks, program structure, or something similar.

U An unrecoverable error message describes an error that prevents Load Module Analyzer from continuing.

Symbols in messages

Many of the Load Module Analyzer messages contain information that is inserted by the system when the message is issued. In this publication, such inserted information is indicated by italicized symbols, as in the following:

EQALM006E **** Work area overflow ****
Explanation: An internal work area has overflowed.

The portion of Load Module Analyzer located on the host notifies you of errors associated with debugging functions carried out by the host.

**EQALM001** **** Unable to open filename ****
Explanation: This indicated file cannot be opened.

**EQALM002** **** 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.

**EQALM003** **** Unknown member specified in SELECT statement ****
Explanation: The member specified on the SELECT statement was not found in the EQALIB concatenation.

**EQALM004** **** Unrecognized control statement ****
Explanation: An unrecognized control statement was encountered while processing the EQAIN file.

**EQALM005** **** Work area overflow ****
Explanation: An internal work area has overflowed.

**EQALM006** **** 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.

**EQALM007** **** text is an unrecognized option ****
Explanation: The specified text is not a supported option.

**EQALM008** **** Debug Tool 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 Debug Tool.

**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 11. Debug Tool Language Environment user exit messages

Each message has a number of the form EQAU\textit{nnn}x, where EQAU indicates that the message is a Debug Tool Language Environment user exit message, \textit{nnn} is the number of the message, and \textit{x} indicates the severity level of each message.

One small set of messages apply to all environments. The rest of the messages are divided into the following categories with a specific prefix:

- “DB2 user exit messages” on page 376 begin with EQAU0
- “IMS user exit messages” on page 377 begin with EQAU1
- “Batch user exit messages” on page 378 begin with EQAU2

The variable \textit{x} can be any of the following values:

I An \textit{informational} message calls attention to some aspect of a command response that might assist the programmer.

W A \textit{warning} message calls attention to a situation that might not be what is expected or to a situation that Debug Tool attempted to fix.

E An \textit{error} message describes an error that Debug Tool detected or cannot fix.

S A \textit{severe} error message describes an error that indicates a command referring to bad data, control blocks, program structure, or something similar.

U An \textit{unrecoverable} error message describes an error that prevents Debug Tool from continuing.

Symbols in messages

Many of the Debug Tool messages contain information that is inserted by the system when the message is issued. In this publication, such inserted information is indicated by italicized symbols, as in the following:

\textbf{EQAU046I} The \textit{breakpoint-id} breakpoint is replaced.

The portion of Debug Tool located on the host notifies you of errors associated with debugging functions carried out by the host.

The following messages apply to all environments:

\begin{tabular}{|l|l|}
\hline
\textbf{EQAU099I} & \textbf{NO READ ACCESS OF DEBUG TOOL USER EXIT DATA SET} \\
\textbf{Explanation:} & Debug Tool tries to read the user exit data set but the access is denied because of RACF or other security protection. \\
\textbf{System action:} & No debug session is started. \\
\textbf{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. \\
\hline
\textbf{EQAU099I} & \textbf{SEE ICH408I MESSAGE} \\
\textbf{Explanation:} & Direct the user to the ICH408I message for more details of the NO READ ACCESS OF DEBUG TOOL USER EXIT DATA SET problem. \\
\textbf{System action:} & None. \\
\textbf{User response:} & Look for ICH408I message in the system log. \\
\hline
\end{tabular}
**DB2 user exit messages**

The following messages are displayed in the DB2 environment:

**EQAU001E** CSVQuery call failed.

**Explanation:** CSVQuery failed to return a valid program stored external name. EQADDCXT returns without a TEST run-time option.

**User response:** Consult your system programmer.

**EQAU002E** DSNWL call failed. Return code is: returnCode.

**Explanation:** DSNWL call failed. EQADDCXT returns without a TEST run-time option.

**User response:** Consult your system programmer.

**EQAU003E** DSNWL call failed.

**Explanation:** DSNWL failed to return a valid authorization ID. EQADDCXT returns without a TEST run-time option.

**User response:** Consult your system programmer.

**EQAU004W** DATA SET dsnname WAS NOT ALLOCATED. DYNAMIC ALLOCATION RETURNED THE FOLLOWING INFORMATION: RETURN CODE=X'\(\text{returnCode},\) REASON CODE=X'\(\text{reasonCode},\) INFO CODE=X'\(\text{infocode}.\)

**Explanation:** The user exit processing for EQADDCXT was not able to retrieve the TEST runtime options because the data set might not exist or cannot be accessed. No debug session was started.

**User response:** If the data set does not exist, start Debug Tool Utilities, select the Manage TEST runtime option, and then fill out the panel with TEST runtime options to create a data set. See the z/OS MVS Programming: Authorized Assembler Services Guide for a description of return code, reason code and info code of dynamic allocation.

**EQAU005W** Data set open failed. Data set name is: dsnname.

**Explanation:** Data set open action failed. EQADDCXT returns without a TEST run-time option.

**User response:** Consult your system programmer.

**EQAU006W** Invalid program name found in data set – pgmname.

**Explanation:** A program name of more than eight characters is defined in the data set. EQADDCXT returns without a TEST run-time option.

**User response:** Use DTU option: Manage TEST run-time option to edit the name on the name list.

**EQAU007W** Data set is empty.

**Explanation:** The data set is empty. EQADDCXT returns without a TEST run-time option.

**User response:** Use DTU option: Manage TEST run-time option to add contents to the data set.

**EQAU008W** No TEST run-time option in data set.

**Explanation:** The data set does not contain TEST run-time option.

**User response:** Use DTU option: Manage TEST run-time option to create a sequential data set with proper content.

**EQAU009E** Data set get failed. Data set name is: dsnname.

**Explanation:** Data set get action failed. EQADDCXT returns without a TEST run-time option.

**User response:** Consult your system programmer.

**EQAU010W** Data set name is not correct. Data set name is: dsnname.

**Explanation:** Data set name is not correct. EQADDCXT returns without a TEST run-time option.

**User response:** Verify and correct data set naming pattern in EQADDCXT. Assemble and re-link it into your application.

**EQAU011W** Data set allocation failed. DD name is not valid.

**Explanation:** DD name is not invalid. EQADDCXT returns without a TEST run-time option.

**User response:** Verify and correct data set name pattern in EQADDCXT. Assemble and re-link it into your application. If problem persists, consult your system programmer.

**EQAU012W** Data set is not physical sequential. Data set name is: dsnname.

**Explanation:** Data set is not a sequential data set. EQADDCXT returns without a TEST run-time option.

**User response:** Use DTU option: Manage TEST run-time option to create a sequential data set.

**EQAU013W** First record type is not \(<PGM>\) in data set. Record is: datarecord.

**Explanation:** The type of the first record is not \(<PGM>\). A Program list record must be the first record in the data set. EQADDCXT returns without a TEST run-time option.

**User response:** Use DTU option: Manage TEST run-time option to create a sequential data set with proper content.

**EQAU014W** Record type is not \(<TST>\) or \(<RTO>\) in data set. Record is: datarecord.

**Explanation:** The type of the record following \(<PGM>\) record is not \(<TST>\) TEST run-time option or \(<RTO>\) other Language Environment run-time option. EQADDCXT returns without a TEST run-time option.

**User response:** Use DTU option: Manage TEST run-time option to create a sequential data set with proper content.

**EQAU015W** Record type is out of sequence in data set. Record is: datarecord.

**Explanation:** The \(<TST>\) record is found after the \(<RTO>\) record. All \(<TST>\) records must be grouped together and placed before first \(<RTO>\) record. EQADDCXT returns without a TEST run-time option.
**IMS user exit messages**

The following messages are displayed in the IMS environment:

<table>
<thead>
<tr>
<th>Message ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAU101E</td>
<td>CSVQuery call failed. Explanation: CSVQuery failed to return a valid program name. EQADICXT returns without a TEST run-time option. User response: Consult your system programmer.</td>
</tr>
<tr>
<td>EQAU104W</td>
<td>DATA SET dsnname WAS NOT ALLOCATED. DYNAMIC ALLOCATION RETURNED THE FOLLOWING INFORMATION: RETURN CODE=X'returncode', REASON CODE=X'reasoncode', INFO CODE=X'infocode'. Explanation: The user exit processing for EQADICXT was not able to retrieve the TEST runtime options because the data set might not exist or cannot be accessed. No debug session was started. User response: If the data set does not exist, start Debug Tool Utilities, select the Manage TEST run-time option, and then fill out the panel with TEST runtime options to create a data set. See the z/OS MVS Programming: Authorized Assembler Services Guide for a description of return code, reason code and info code of dynamic allocation.</td>
</tr>
<tr>
<td>EQAU106W</td>
<td>Invalid program name found in data set – pgmname. Explanation: A program name of more than eight characters is defined in the data set. EQADICXT returns without a TEST run-time option. User response: Use DTU option: Manage TEST run-time option to edit the name on the name list.</td>
</tr>
<tr>
<td>EQAU107W</td>
<td>Data set is empty. Explanation: The data set is empty. EQADICXT returns without a TEST run-time option. User response: Use DTU option: Manage TEST run-time option to add contents to the data set.</td>
</tr>
<tr>
<td>EQAU108W</td>
<td>No TEST run-time option in data set. Explanation: The data set does not contain TEST run-time option. EQADICXT returns without a TEST run-time option. User response: Use DTU option: Manage TEST run-time option to add TEST run-time option to the data set.</td>
</tr>
<tr>
<td>EQAU110W</td>
<td>Data set name is not correct. Data set name is: dsnname. Explanation: Data set name is not correct. EQADICXT returns without a TEST run-time option. User response: Verify and correct data set naming pattern in EQADICXT. Assemble and re-link it into your application.</td>
</tr>
<tr>
<td>EQAU111W</td>
<td>Data set allocation failed. DD name is not valid. Explanation: DD name is not invalid. EQADICXT returns without a TEST run-time option. User response: Verify and correct data set name in EQADICXT. Assemble and re-link it into your application. If problem persists, consult your system programmer.</td>
</tr>
<tr>
<td>EQAU112W</td>
<td>Data set is not physical sequential. Data set name is: dsnname. Explanation: Data set is not a sequential data set. EQADICXT returns without a TEST run-time option. User response: Use DTU option: Manage TEST run-time option to create a sequential data set.</td>
</tr>
<tr>
<td>EQAU113W</td>
<td>First record type is not &lt;PGM&gt; in data set. Record is: datarecord. Explanation: The type of the first record is not &lt;PGM&gt;. A Program list record must be the first record in the data set. EQADDCXT returns without a TEST run-time option. User response: Use DTU option: Manage TEST run-time option to create a sequential data set with proper content.</td>
</tr>
<tr>
<td>EQAU114W</td>
<td>Record type is not &lt;TST&gt; or &lt;RTO&gt; in data set. Record is: datarecord. Explanation: The type of the record following &lt;PGM&gt; record is not &lt;TST&gt; TEST run-time option or &lt;RTO&gt; other Language Environment run-time option. EQADDCXT returns without a TEST run-time option. User response: Use DTU option: Manage TEST run-time option to create a sequential data set with proper content.</td>
</tr>
<tr>
<td>EQAU115W</td>
<td>Record type is out of sequence in data set. Record is: datarecord. Explanation: &lt;TST&gt; record is found after &lt;RTO&gt; record. All &lt;TST&gt; records must be grouped together and placed before first &lt;RTO&gt; record. EQADDCXT returns without a TEST run-time option. User response: Use DTU option: Manage TEST run-time option to create a sequential data set with proper content.</td>
</tr>
</tbody>
</table>
Batch user exit messages

The following messages are displayed with batch programs:

**EQAU201E**  CSVQuery call failed.

**Explanation:** CSVQuery failed to return a valid program name. EQADBCXT returns without a TEST run-time option.

**User response:** Consult your system programmer.

**EQAU204W**  DATA SET dsnname WAS NOT ALLOCATED. DYNAMIC ALLOCATION RETURNED THE FOLLOWING INFORMATION: RETURN CODE=X'returncode', REASON CODE=X'reasoncode', INFO CODE=X'infocode'.

**Explanation:** The user exit processing for EQADBCXT was not able to retrieve the TEST runtime options because the data set might not exist or cannot be accessed. No debug session was started.

**User response:** If the data set does not exist, start Debug Tool Utilities, select the Manage TEST runtime option, and then fill out the panel with TEST runtime options to create a data set. See the z/OS MVS Programming: Authorized Assembler Services Guide for a description of return code, reason code and info code of dynamic allocation.

**EQAU205W**  Data set open failed. Data set name is: dsnname.

**Explanation:** Data set open action failed. EQADBCXT returns without a TEST run-time option.

**User response:** Consult your system programmer.

**EQAU206W**  Invalid program name found in data set – pgmname.

**Explanation:** A program name of more than 8 characters is defined in the data set. EQADBCXT returns without a TEST run-time option.

**User response:** Use DTU option: Manage TEST run-time option to edit the name on the name list.

**EQAU207W**  Data set is empty.

**Explanation:** The data set is empty. EQADBCXT returns without a TEST run-time option.

**User response:** Use DTU option: Manage TEST run-time option to add contents to the data set.

**EQAU208W**  No TEST run-time option in data set.

**Explanation:** The data set does not contain TEST run-time option. EQADBCXT returns without a TEST run-time option.

**User response:** Use DTU option: Manage TEST run-time option to add TEST run-time option to the data set.

**EQAU209E**  Data set get failed. Data set name is: dsnname.

**Explanation:** Data set get action failed. EQADBCXT returns without a TEST run-time option.

**User response:** Consult your system programmer.

**EQAU210W**  Data set name is not correct. Data set name is: dsnname.

**Explanation:** Data set name is not correct. EQADBCXT returns without a TEST run-time option.

**User response:** Verify and correct data set name in EQADBCXT. Assemble and re-link it into your application.

**EQAU211W**  Data set allocation failed. DD name is not valid.

**Explanation:** DD name is not invalid. EQADBCXT returns without a TEST run-time option.

**User response:** Verify and correct data set name in EQADBCXT. Assemble and re-link it into your application. If problem persists, consult your system programmer.

**EQAU212W**  Data set is not physical sequential. Data set name is: dsnname.

**Explanation:** Data set is not a sequential data set. EQADBCXT returns without a TEST run-time option.

**User response:** Use DTU option: Manage TEST run-time option to create a sequential data set.

**EQAU213W**  First record type is not <PGM> in data set. Record is: datarecord.

**Explanation:** The type of the first record is not <PGM>. A Program list record must be the first record in the data set. EQADDCXT returns without a TEST run-time option.

**User response:** Use DTU option: Manage TEST run-time option to create a sequential data set with proper content.

**EQAU214W**  Record type is not <TST> or <RTO> in data set. Record is: datarecord.

**Explanation:** The type of the record following <PGM> record is not <TST> TEST run-time option or <RTO> other Language Environment run-time option. EQADDCXT returns without a TEST run-time option.

**User response:** Use DTU option: Manage TEST run-time option to create a sequential data set with proper content.

**EQAU215W**  Record type is out of sequence in data set. Record is: datarecord.

**Explanation:** <TST> record is found after <RTO> record. All <TST> records must be grouped together and placed before first <RTO> record. EQADDCXT returns without a TEST run-time option.

**User response:** Use DTU option: Manage TEST run-time option to create a sequential data set with proper content.
Chapter 12. Debug Tool Terminal Interface Manager messages

Each message has a number of the form EQAYnnnx, where EQAY indicates that the message is a Debug Tool Terminal Interface Manager message, nnn is the number of the message, and x indicates the severity level of each message. The variable x can be any of the following values:

I An informational message calls attention to some aspect of a command response that might assist the programmer.

W A warning message calls attention to a situation that might not be what is expected or to a situation that Debug Tool attempted to fix.

E An error message describes an error that Debug Tool detected or cannot fix.

S A severe error message describes an error that indicates a command referring to bad data, control blocks, program structure, or something similar.

U An unrecoverable error message describes an error that prevents Debug Tool from continuing.

Symbols in messages

Many of the Debug Tool messages contain information that is inserted by the system when the message is issued. In this publication, such inserted information is indicated by italicized symbols, as in the following:

EQAY999S The breakpoint-id breakpoint is replaced.

The portion of Debug Tool located on the host notifies you of errors associated with debugging functions carried out by the host.

<table>
<thead>
<tr>
<th>Message Code</th>
<th>Message Description</th>
<th>Explanation</th>
<th>Programmer response</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAY999S</td>
<td>Invalid userid/password. Respecify.</td>
<td>An invalid userid or incorrect password has been specified to Debug Tool Terminal Interface Manager.</td>
<td>Respecify the userid and/or password.</td>
</tr>
<tr>
<td>EQAY999S</td>
<td>Error receiving lu name</td>
<td>Severe internal error (VTAM 3270) in Debug Tool Terminal Interface Manager Please contact your IBM representative.</td>
<td></td>
</tr>
<tr>
<td>EQAY999S</td>
<td>Screen dimensions could not be determined</td>
<td>Severe internal error (VTAM 3270) in Debug Tool Terminal Interface Manager Please contact your IBM representative.</td>
<td></td>
</tr>
<tr>
<td>EQAY999S</td>
<td>Logon message not available</td>
<td>Severe internal error (VTAM 3270) in Debug Tool Terminal Interface Manager Please contact your IBM representative.</td>
<td></td>
</tr>
<tr>
<td>EQAY999S</td>
<td>Session parameters inquiry error</td>
<td>Severe internal error (VTAM 3270) in Debug Tool Terminal Interface Manager Please contact your IBM representative.</td>
<td></td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 1992, 2009
EQAY999I Shutting down Debug Tool Terminal Interface Manager

Explanation: Debug Tool Terminal Interface Manager has been requested to shut down.
Chapter 13. Debug Tool Utilities messages

All messages are shown in this section are in mixed case English. The uppercase English message text is the same, but is in uppercase letters.

Each message has a number of the form EQAZnnnx, where EQAZ indicates that the message is an Debug Tool Utilities message, nnn is the number of the message, and x indicates the severity level of each message. The value of x is I, W, E, S, or U, as described below:

I An informational message calls attention to some aspect of a command response that might assist the programmer.

W A warning message calls attention to a situation that might not be what is expected or to a situation that Debug Tool Utilities attempted to fix.

E An error message describes an error that Debug Tool Utilities detected or cannot fix.

S A severe error message describes an error that indicates a command referring to bad data, control blocks, program structure, or something similar.

U An unrecoverable error message describes an error that prevents Debug Tool Utilities from continuing.

Symbols in messages

Many of the Debug Tool Utilities messages contain information that is inserted by the system when the message is issued. In this publication, such inserted information is indicated by italicized symbols, as in the following:

EQA1046I The breakpoint-id breakpoint is replaced.

EQAZ005S Install Error cmdName has no value for hlq
Explanation: cmdName exec detects that no value is assigned for variable hlq.
System action: hlq is used as the high level qualifiers for data set names.
User response: Follow the instructions "Customizing Debug Tool Utilities" in DT Customization Guide to modify EQASTART to customize data set names.

EQAZ006S Install DSN Error cmdName. Missing 'dataSet'
Explanation: cmdName cannot find dataSet.
System action: cmdName exec ends.
User response: Follow the instructions "Customizing Debug Tool Utilities" in DT Customization Guide to modify EQASTART to customize data set names.

EQAZ007S libType LIBDEF Failed for lib
Explanation: Allocation of application library of type libType failed for lib.
System action: Debug Tool Utility ends.
User response: Follow the instructions "Customizing Debug Tool Utilities" in DT Customization Guide to modify EQASTART to customize data set names.

EQAZ010W Allocation Error: dsnName
Explanation: Allocation failed for dsnName.
System action: dsnName is not processed.
User response: Make sure that dsnName exists.

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.
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQAZ027E</td>
<td>Invalid line command detected. <code>cmd</code> is not allowed for <code>ddn</code> DD statement.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>Invalid line command entered.</td>
</tr>
<tr>
<td>System action:</td>
<td><code>cmd</code> is not executed.</td>
</tr>
<tr>
<td>User response:</td>
<td>Enter a valid command.</td>
</tr>
<tr>
<td>EQAZ029W</td>
<td><code>cmd</code> return code <code>rc</code></td>
</tr>
<tr>
<td>Explanation:</td>
<td><code>cmd</code> (edit, view, or browse) command has a return code of <code>rc</code>.</td>
</tr>
<tr>
<td>System action:</td>
<td>Processing continues.</td>
</tr>
<tr>
<td>User response:</td>
<td>Refer to the ISPF documentation for return code meaning.</td>
</tr>
<tr>
<td>EQAZ030W</td>
<td>Member specified for <code>fileName</code> but type is not PDSE or PDS.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>A member is specified for <code>fileName</code>, but its type is not PDSE or PDS.</td>
</tr>
<tr>
<td>System action:</td>
<td>You are prompted to correct the problem.</td>
</tr>
<tr>
<td>User response:</td>
<td>Remove the member name, or specify a PDSE or PDS.</td>
</tr>
<tr>
<td>EQAZ031W</td>
<td>Member not specified for <code>fileName</code> for type PDSE or PDS.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>No member is specified for <code>fileName</code> of type PDSE or PDS.</td>
</tr>
<tr>
<td>System action:</td>
<td>You are prompted to correct the problem.</td>
</tr>
<tr>
<td>User response:</td>
<td>Enter a member name.</td>
</tr>
<tr>
<td>EQAZ033W</td>
<td>An invalid data set pattern character <code>pChar</code> was used.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>An invalid character <code>pChar</code> was used in a data set pattern in the program preparation.</td>
</tr>
<tr>
<td>System action:</td>
<td>You are prompted to correct the problem.</td>
</tr>
<tr>
<td>User response:</td>
<td>Use field help to choose a valid pattern character. (To see field help, press the HELP key with the cursor positioned in the field.)</td>
</tr>
<tr>
<td>EQAZ034S</td>
<td><code>fileName</code> is multiply included. Circular definitions are not allowed.</td>
</tr>
<tr>
<td>Explanation:</td>
<td><code>fileName</code> is included multiple times in the settings file or files.</td>
</tr>
<tr>
<td>System action:</td>
<td>Include statement is not processed.</td>
</tr>
<tr>
<td>User response:</td>
<td>Contact your system administrator. Make sure that <code>fileName</code> is not included more than once in the same settings file or nested settings file.</td>
</tr>
<tr>
<td>EQAZ035S</td>
<td>Too many INCLUDE statements were found in the settings file</td>
</tr>
<tr>
<td>Explanation:</td>
<td>More than sixteen INCLUDE statements were found in the settings file or files.</td>
</tr>
<tr>
<td>System action:</td>
<td>Include statements starting with the sixteenth are not processed.</td>
</tr>
<tr>
<td>User response:</td>
<td>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.</td>
</tr>
<tr>
<td>EQAZ036E</td>
<td>End of file for member <code>fileName</code> while processing statement</td>
</tr>
<tr>
<td>Explanation:</td>
<td>Incomplete statement is found in <code>fileName</code> settings file.</td>
</tr>
<tr>
<td>System action:</td>
<td>The statement is not processed.</td>
</tr>
<tr>
<td>User response:</td>
<td>Contact your system administrator. Make sure that the statement in <code>fileName</code> is properly ended with a semicolon.</td>
</tr>
<tr>
<td>EQAZ037E</td>
<td>Invalid keyword <code>kwd</code> found in member <code>fileName</code>.lineNo</td>
</tr>
<tr>
<td>Explanation:</td>
<td>Invalid keyword <code>kwd</code> found in line <code>lineNo</code> of settings file <code>fileName</code>.</td>
</tr>
<tr>
<td>System action:</td>
<td>Include statement is not processed.</td>
</tr>
<tr>
<td>User response:</td>
<td>Contact your system administrator. Make sure that the statement in line <code>lineNo</code> of settings file member <code>fileName</code> is a valid statement.</td>
</tr>
<tr>
<td>EQAZ038I</td>
<td>You must supply a valid job card to use batch.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>A request to run the application in batch is submitted but no valid job card is found.</td>
</tr>
<tr>
<td>System action:</td>
<td>You are prompted for job card information.</td>
</tr>
<tr>
<td>User response:</td>
<td>Enter job card information.</td>
</tr>
<tr>
<td>EQAZ039W</td>
<td>An invalid sequence number <code>seqValue</code> was entered.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>A non-numeric value <code>seqValue</code> is entered in the sequence number field.</td>
</tr>
<tr>
<td>System action:</td>
<td><code>seqValue</code> is removed from the field.</td>
</tr>
<tr>
<td>User response:</td>
<td>Enter a numeric value.</td>
</tr>
<tr>
<td>EQAZ040I</td>
<td>Action started for <code>srcName</code> using <code>inName</code>.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>Action, which can be any program preparation action (compile, assemble, link-edit, create FA side file, or convert old COBOL program), is started for source <code>srcName</code> using input <code>inName</code>.</td>
</tr>
<tr>
<td>System action:</td>
<td>Processing continues.</td>
</tr>
<tr>
<td>User response:</td>
<td>None.</td>
</tr>
<tr>
<td>EQAZ041I</td>
<td>loadlib library; <code>modName</code> invoked.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>The load module <code>modName</code> in library <code>loadlib</code>, which is the data set where a compiler or assembler resides, is invoked.</td>
</tr>
<tr>
<td>System action:</td>
<td>Processing continues.</td>
</tr>
<tr>
<td>User response:</td>
<td>None.</td>
</tr>
<tr>
<td>EQAZ042I</td>
<td>CICS translator started for <code>fileName</code>.</td>
</tr>
<tr>
<td>Explanation:</td>
<td>CICS translator starts to translate <code>fileName</code>.</td>
</tr>
<tr>
<td>System action:</td>
<td>Processing continues.</td>
</tr>
<tr>
<td>User response:</td>
<td>None.</td>
</tr>
<tr>
<td>Message Code</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>EQAZ043I</td>
<td>CICS translator invoked from complib.</td>
</tr>
<tr>
<td>EQAZ044I</td>
<td>Converter invoked from complib.</td>
</tr>
<tr>
<td>EQAZ045I</td>
<td>DB2 preprocessor started for file.</td>
</tr>
<tr>
<td>EQAZ046I</td>
<td>DB2 Preprocessor invoked from complib.</td>
</tr>
<tr>
<td>EQAZ047I</td>
<td>EQALANGX started for file.</td>
</tr>
<tr>
<td>EQAZ048I</td>
<td>EQALANGX invoked from complib.</td>
</tr>
<tr>
<td>EQAZ049W</td>
<td>You must specify the parameter string format.</td>
</tr>
<tr>
<td>EQAZ050W</td>
<td>Allocation of a temporary data set failed.</td>
</tr>
<tr>
<td>EQAZ051S</td>
<td>Internal error in exec. Invalid panel = panel.</td>
</tr>
<tr>
<td>EQAZ052S</td>
<td>Internal error in exec. Invalid command = cmd.</td>
</tr>
<tr>
<td>EQAZ053W</td>
<td>Invalid DSN type entered - dsnType.</td>
</tr>
<tr>
<td>EQAZ054W</td>
<td>LISTDSI failed for dsnName Level1ErrorMsg Level2ErrorMsg.</td>
</tr>
<tr>
<td>EQAZ055S</td>
<td>Internal error in exec. Too many variable types, varTypeList.</td>
</tr>
<tr>
<td>EQAZ056S</td>
<td>Internal error in exec. Invalid variable type = varType.</td>
</tr>
<tr>
<td>EQAZ057W</td>
<td>Data set dsnName is not available - errorMsg.</td>
</tr>
</tbody>
</table>
EQAZ058W  
**dsnUse Data set dsnName is not available - errorMsg.**

**Explanation:** The data set is not available for the reason specified in `errorMsg`.

**System action:** The action on the data set is not performed. The program preparation panel is presented.

**User response:** Check the program preparation return code on the panel. Report the problem to the system administrator.

---

EQAZ059W  
**No IMSplex ID**

**Explanation:** IMSplex ID is required.

**System action:** You are prompted for an IMSplex ID.

**User response:** Enter a valid IMSplex ID. Contact your system administrator if you do not have an ID.

---

EQAZ060E  
**No REXX IMS SPOC**

**Explanation:** REXX IMS SPOC environment is not available. Return Code = RC.

**System action:** The action on LE runtime options is not performed.

**User response:** Contact your system administrator and verify that IMS V8 is installed on your system and that Debug Tool Utility is properly installed and configured. See “Debug Tool Customization Guide” for details.

---

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 Debug Tool Utility is properly installed and configured. See “Debug Tool 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 Debug Tool Utility is properly installed and configured. See “Debug Tool Customization Guide” for details.

---

EQAZ065E  
**Non-LE program cannot have load module name = EQANMDBG.**

**Explanation:** EQANMDBG is a reserved load module name for Debug Tool 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 run-time option data set must be a sequential data set.

**System action:** GDSname data set is not allocated.

**User response:** Provide a valid generation data set name.

---

EQAZ070E  
**GDSname is not a valid generation data set name.**

**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 GnnnnVnn 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 step type 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.
Appendix A. Reference card: Frequently used Debug Tool commands

The following reference card provides a list of frequently used Debug Tool commands. For best results, print the card from the PDF file.
Setting breakpoints (stopping points) at statements in a program

A
A is the abbreviation for AT. Enter through the prefix area of the Source window. Sets a breakpoint on line where A is entered.

PF6
Sets a breakpoint on line where cursor is located.

AT 509
Sets a breakpoint on line 509.

AT LABEL label_name
Sets a breakpoint on a label, paragraph, or section name.

Clearing (removing) breakpoints set at statements in a program

C
C is the abbreviation for CLEAR AT. Type a C in the prefix area of the Source window. When you press Enter, Debug Tool removes the breakpoint on the line where C is in the prefix area.

PF6
Removes a breakpoint on line where cursor is located.

CLEAR AT 509
Removes the breakpoint on line 509.

CLEAR AT LABEL label_name
Clears a breakpoint from a label, paragraph, or section name.

Setting breakpoints (stopping points) triggered by a change in the value of a variable

AT CHANGE ITEMNO
Sets a breakpoint that stops the program when the value of ITEMNO changes.

Clearing (removing) breakpoints triggered by a change in the value of a variable

CLEAR AT CHANGE ITEMNO
Removes the breakpoint that stops the program when the value of ITEMNO changes.

Setting breakpoints (stopping points) at the entrance or exit of a program

AT ENTRY cu_name
Sets a breakpoint that stops the program when it enters cu_name.

AT ENTRY *
Sets breakpoints that stop a program whenever Debug Tool enters a known program.

AT EXIT cu_name
Sets a breakpoint that stops the program when it exits cu_name.

AT EXIT *
Sets breakpoints that stop a program whenever Debug Tool exits a known program.

Clearing (removing) breakpoints set at the entrance or exit of a program

CLEAR AT ENTRY cu_name
Clears the breakpoint that stops the program when it enters cu_name.

CLEAR AT ENTRY *
Clears all breakpoints that stop a program whenever Debug Tool enters a known program.

CLEAR AT EXIT cu_name
Clears a breakpoint that stops the program when it exits cu_name.

CLEAR AT EXIT *
Removes the breakpoints at every exit point in every program.

Making breakpoints conditional

Add a WHEN clause to make breakpoints conditional.

AT CHANGE ITEMNO WHEN ITEMNO = '0805'
Stop after the value of ITEMNO changes, but only if ITEMNO is equal to the specified value.

AT CHANGE CUSTID WHEN ACCT-BAL > 100
Stop after the value of CUSTID changes, but only if ACCT-BAL is greater than the specified value.

AT 509 WHEN ITEMNO = '0805'
Stop at statement 509, but only if ITEMNO is equal to the specified value.

Commands that work on all breakpoints

LIST AT
Displays all breakpoints in the Log window.

CLEAR AT
Cleans all breakpoints.

DISABLE AT
Temporarily disables (deactivates) all breakpoints.

ENABLE AT
Enables (activates) all disabled breakpoints.

Identifying and loading a program’s source and debug information

SET DEFAULT LISTINGS source.info.library
Identifies a source library (PDS or PDSE) where Debug Tool searches for source files and debug information files. For example, SYSDEBUG files, LANGX files, and compiler listings. Debug Tool displays this information in the Source window.

SET DEFAULT LISTINGS (source.info.lib1, source.info.lib2, ...)
Identifies a concatenation of source libraries (PDS or PDSE) where Debug Tool searches for source files and debug information files.

LISTING or LIST
Displays a list of programs known to Debug Tool. Then, you can specify the name of the source file or debug information file for each program.

LDD assembler_CSECT or LDO nonLECOBOL_program
Load debug information about assembler_CSECT or nonLECOBOL_program from the EQALANGX file into the Source window.
Displaying variables in the Monitor window

- **SET AUTOMONITOR ON**
  Automatically displays the values of variables referenced by the current statement in the Monitor window.

- **SET AUTOMONITOR ON BOTH**
  Automatically displays the values of variables referenced by both the current statement and the previously run statement in the Monitor window.

- **MONITOR LIST ITEMNO**
  Adds the ITEMNO variable and its value to the Monitor window.

- **SET MONITOR DATATYPE ON**
  Display the data types of variables.

- **SET MON WRAP OFF**
  Displays values on a single line. If the value is longer than the visible area, Debug Tool displays a scale to indicate that there is more to see.

- **CLEAR MONITOR**
  Clears all items from the Monitor window.

C
  C is the abbreviation for the CLEAR MONITOR command. Type in the letter C in the prefix area of the monitor window. When you press Enter, Debug Tool removes the variable on the line where C is in the prefix area.

Changing values of variables

Type over value displayed in the Monitor window

- Move cursor to value displayed in Monitor window, type in new value, then press Enter.

- **MOVE 24 to ACCUM-X**
  For COBOL programs, replace the value of ACCUM-X with 24.

- **ACCUMX = 24**
  For some languages, replaces the value of ACCUMX with 24.

Displaying variables in Log window and controlling Log window options

- **LIST CUST-ID** or **LIST TITLED CUST-ID**
  Displays the value of a variable. Only some programming languages require TITLED.

- **PF4** or **LIST**
  Displays the value of a variable identified by the location of the cursor.

- **LIST TITLED WSS or LS or FS or LOS**
  Display contents of specific SECTIONS for COBOL programs. WSS means Working-Storage Section, LS means Linkage Section, FS means File Section, and LOS means Local-Storage Section.

- **LIST TITLED ***
  Displays the values of all variables.

- **SET ECHO OFF**
  Debug Tool does not display STEP and GO commands in the Log window. However, if a log file is open, Debug Tool writes them to the log file.

- **SET LOG ON FILE file_name**
  Opens a log file. When Debug Tool opens the log file, all items it writes to the Log window are also written to the log file.

- **QUALIFY program_name**
  Displays the program program_name in the Source window. When the program is displayed in the Source window, you can set a breakpoint or work with variables in that program.

- **QUALIFY RESET**
  Repositions to the current program and the current line.

- **QUALIFY**
  Repositions source in the Source window so that Debug Tool displays the current program and current statement.

Controlling program execution

- **STEP** or **PF2**
  Run one statement or line.

- **GO** or **PF9**
  Run the program until Debug Tool encounters a breakpoint, the program finishes, or an abend occurs.

- **RUNTO 27**
  Runs the program and then stops before it runs line 27.

- **R**
  R is the abbreviation of RUNTO. Type in the command in the prefix area of the Source window. When you press Enter, Debug Tool runs the program until it reaches the line with the R in the prefix area.

- **GO BYPASS**
  Resume running a program after encountering an abend. Enter this command immediately after an abend occurs. Debug Tool skips the statement that caused the abend and continues running the program from the next logical statement.

Skipping (do not run) over program statements

- **JUMPTO 27**
  Moves the point at which the program resumes execution to line 27, does not run any statements between the current point and line 27, and then pauses at line 27. When you enter a GO or STEP command, the program resumes running at line 27.

- **GOTO 27**
  Moves the point at which the program resumes execution to line 27, does not run any statements between the current point and line 27, and then resumes running the program at line 27.
Commands that work in the prefix area of the Source window

A
A is the abbreviation for AT. Sets a breakpoint on the line.

C
C is the abbreviation for CLEAR AT. Clears the breakpoint from the line.

D
D is the abbreviation for DISABLE AT. Disables the breakpoint on the line.

E
E is the abbreviation for ENABLE AT. Enables the breakpoint on the line.

L
L is the abbreviation for LIST. Displays all variables referenced by the statement in the log. This prefix command works only for programs compiled with specific compilers.

L1, L2, L3,...
L is the abbreviation for LIST. Displays the first, second, third, and so on variable referenced by the statement in the log. This prefix command works only for programs compiled with specific compilers.

M
M is the abbreviation for MONITOR LIST. Displays all variables reference by the statement in the Monitor window. This prefix command works only for programs compiled with specific compilers.

M1, M2, M3,...
M is the abbreviation for MONITOR LIST. Displays first, second, third, and so on variable referenced by the statement in the Monitor window. This prefix command works only for programs compiled with specific compilers.

Working with PF keys

QUERY PFKEYS
Displays the PF key settings in the log.

SET KEYS ON
Debug Tool displays the PF key settings for PF keys 1-12 at the bottom of the screen.

SET KEYS ON 24
Debug Tool displays the PF key settings for PF keys 13-24 at the bottom of the screen.

SET KEYS OFF
Debug Tool removes the PF key settings from the bottom of the screen.

SET PF16 "MON" = MONITOR LIST
Example of assigning a command to a PF key. In this example, you assign the MONITOR LIST command to the PF16 key. When Debug Tool displays PF keys 13-24 at the bottom of the screen, it shows "PF16=MON".

Default PF key settings

PF1 or PF13
? (HELP)

PF2 or PF14
STEP

PF3 or PF15
END

PF4 or PF16
LIST

PF5 or PF17
FIND

PF6 or PF18
AT/CLEAR

PF7 or PF19
UP

PF8 or PF20
DOWN

PF9 or PF21
GO

PF10 or PF22
ZOOM

PF11 or PF23
ZOOM LOG

PF12 or PF24
RETRIEVE

Displaying help for commands

? Displays a list of commands

AT ? Example of displaying help for the AT command. Enter all or part of a command, followed by a question mark (?) to display keywords that are valid at the location of the question mark.

Continuing a long command

- (dash at the end of a line)
To continue a long command (for example, a command that exceeds the size of the command line), type a dash at the end of a partial command and then press Enter. Debug Tool prompts you to enter the rest of the command.

Abbreviating commands

(use partial keywords)
You can abbreviate keywords in Debug Tool commands to the least number of letters that make the keyword unambiguous. For example, you can abbreviate the command MONITOR LIST VARX to MON LIST VARX or MO LIS VARX.

Ending a debugging session

QUIT Ends the debugging session and prompts you to verify that you want to end the debugging session.

QUIT Ends the debugging session without prompting you.

QUIT DEBUG Ends the debugging session but program continues to run. Debug Tool will not be restarted.

QUIT DEBUG TASK
This command works only for CICS. Ends debugging session but the program continues to run. Debug Tool will not be restarted. To start Debug Tool, start another iteration of a pseudo-conversational task.

QUIT ABEND
Ends the debugging session and terminates the program with an abend at the current location.
Appendix B. Debug Tool commands supported in remote debug mode

You can use some Debug Tool commands in remote debug 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. You must have the Compiled Language Debugger component of Rational Developer for System z, Version 7.5, to use this feature.
- Use them in a commands or preferences file.

Using any of these methods, you can use the following commands in remote debug mode:

- “AT CHANGE command (remote debug mode)” on page 53
- “AT ENTRY command (remote debug mode)” on page 59
- “AT LOAD command (remote debug mode)” on page 67
- “AT STATEMENT command (remote debug mode)” on page 75
- “CALL %VER command” on page 85
- “CHKSTGV command” on page 88
- CLEAR LOAD, which is described in “CLEAR command” on page 88
- DESCRIBE CHANNEL, which is described in “DESCRIBE command” on page 102
- DESCRIBE CUS, which is described in “DESCRIBE command” on page 102
- DESCRIBE LOADMODS, which is described in “DESCRIBE command” on page 102
- DISABLE CADP, which is described in “DISABLE command” on page 106
- DISABLE DTCN, which is described in “DISABLE command” on page 106
- ENABLE CADP, which is described in “ENABLE command” on page 112
- ENABLE DTCN, which is described in “ENABLE command” on page 112
- LIST CADP, which is described in “LIST DTCN or CADP command” on page 148
- “LIST CONTAINER command” on page 146
- LIST DTCN, which is described in “LIST DTCN or CADP command” on page 148
- “LOAD command” on page 162
- “LOADDEBUGDATA command” on page 164 (for assembler only)
- “NAMES DISPLAY command” on page 175
- “NAMES EXCLUDE command” on page 176
- “NAMES INCLUDE command” on page 177
- QUERY ASSEMBLER, which is described in “QUERY command” on page 190
- QUERY AUTOMONITOR, which is described in “QUERY command” on page 190
• QUERY BROWSE MODE, which is described in “QUERY command” on page 190

• QUERY CURRENT VIEW, which is described in “QUERY command” on page 190

• QUERY DEFAULT DBG, which is described in “QUERY command” on page 190

• QUERY DEFAULT LISTINGS, which is described in “QUERY command” on page 190

• QUERY DEFAULT MDBG, which is described in “QUERY command” on page 190

• QUERY DEFAULT VIEW, which is described in “QUERY command” on page 190

• QUERY DISASSEMBLY, which is described in “QUERY command” on page 190

• QUERY DYNDDEBUG, which is described in “QUERY command” on page 190

• QUERY IGNORELINK, which is described in “QUERY command” on page 190

• QUERY INTERCEPT, which is described in “QUERY command” on page 190

• QUERY LDD, which is described in “QUERY command” on page 190

• QUERY LOCATION, which is described in “QUERY command” on page 190

• QUERY LOG, which is described in “QUERY command” on page 190

• QUERY REWRITE, which is described in “QUERY command” on page 190

• QUERY WARNING, which is described in “QUERY command” on page 190

• QUIT, which is described in “QUIT command” on page 196

• QUIT ABEND, which is described in “QUIT command” on page 196

• QUIT DEBUG, which is described in “QUIT command” on page 196

• QUIT command” on page 197

• “SET ASSEMBLER ON/OFF command” on page 207

• “SET ASSEMBLER STEPOVER command” on page 208

• “SET AUTOMONITOR command” on page 209

• “SET DEFAULT DBG command” on page 216

• “SET DEFAULT LISTINGS command” on page 217

• “SET DEFAULT MDBG command” on page 218

• “SET DEFAULT VIEW command” on page 220

• “SET DISASSEMBLY command” on page 221

• “SET DYNDDEBUG command” on page 222

• “SET IGNORELINK command” on page 228

• “SET INTERCEPT command (COBOL, remote debug mode)” on page 231

• “SET LDD command” on page 232

• SET LOG OFF, which is described in “SET LOG command” on page 233

• SET LOG ON, which is described in “SET LOG command” on page 233

• SET QUALIFY CU, which is described in “SET QUALIFY command” on page 244

• SET QUALIFY LOAD, which is described in “SET QUALIFY command” on page 244

• “SET REWRITE command (remote debug mode)” on page 249

• “SET WARNING command (C, C++, COBOL, and PL/I)” on page 258

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:
Through the **Debug Engine Command** field or the **Debug Console Commands** window, you can view a list of Debug Tool 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 Debug Tool commands that begin with those same letters is displayed.

To use this feature, you must use the Compiled Language Debugger component of Rational Developer for System z, Version 7.5.

### Specifying the location of source, listing, or separate debug file in remote debug mode by using environment variables

Debug Tool 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.

The following example shows you how to declare an environment variable that specifies the location of a listing file:

```bash
export EQA_SRC_PATH="/u/build1/tryp1:/u/build2/tryp2: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.

### Adding an Eclipse-based plug-in that can read, create, update, and delete DTCN profiles

You can download an example of an Eclipse-based plug-in that can manipulate DTCN profiles from the workstation by following the instructions in topic “Running a sample plug-in that uses the API”, which is in **Debug Tool API User’s Guide and Reference**.
Appendix C. Changes in behavior of some commands

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.

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 48
- “LIST STORAGE command” on page 160
- “MEMORY command” on page 165
- “WINDOW SWAP command” on page 279
- “STORAGE command” on page 266
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.


- “Searching IBM support Web sites for a solution”
- “Obtaining fixes” on page 401
- “Receiving support updates through e-mail notification” on page 401
- “Receiving support updates through RSS feeds” on page 402
- “If you need to contact IBM Software Support” on page 402

Searching IBM support Web sites for a solution

You can search the available knowledge bases to determine whether your problem was already encountered and is already documented.

- “Searching the information center”
- “Searching product support documents”
- “IBM Support Assistant” on page 400

Searching the information center

You can find this publication and documentation for many other products in the IBM System z Enterprise Development Tools & Compilers information center at http://publib.boulder.ibm.com/infocenter/pdthelp/v1r1/index.jsp. Using the information center, you can search product documentation in a variety of ways. You can search across the documentation for multiple products, search across a subset of the product documentation that you specify, or search a specific set of topics that you specify within a document. Search terms can include exact words or phrases, wild cards, and Boolean operators.

To learn more about how to use the search facility provided in the IBM System z Enterprise Development Tools & Compilers information center, you can view the multimedia presentation at http://publib.boulder.ibm.com/infocenter/pdthelp/v1r1/index.jsp?topic=/com.ibm.help.doc/InfoCenterTour800600.htm.

Searching product support documents

Use the System z Enterprise Development Tools & Compilers information center or the product support page to search the Internet for the latest, most complete information that might help you resolve your problem.
Specific IBM Software Support sites for the System z Enterprise Development Tools and Compilers products include:

- Application Performance Analyzer for z/OS Support
- Debug Tool for z/OS Support
- Enterprise COBOL for z/OS Support
- Enterprise PL/I for z/OS Support
- Fault Analyzer for z/OS Support
- File Export for z/OS Support
- File Manager for z/OS Support
- Optim™ Move for DB2 Support
- WebSphere Developer Debugger for System z Support
- WebSphere Studio Asset Analyzer for Multiplatforms Support
- Workload Simulator for z/OS and OS/390 Support

To search multiple Internet resources for your product using the information center, click Troubleshooting and support in the left navigation pane and select Searching IBM support Web sites for a solution. You can select one or more products, specify keywords, and search a variety of resources, including the following:

- IBM technotes
- IBM downloads and fixes
- IBM problem reports (APARs) and flashes
- IBM Redbooks®, whitepapers, articles, and tutorials
- IBM developerWorks®
- Forums and newsgroups
- Google

There is also a search facility provided on the product support page. The search facility provided on the product support page allows you to narrow the search scope and search only product support documents for that product.

**IBM Support Assistant**

The IBM Support Assistant (also referred to as ISA) is a free local software serviceability workbench that helps you resolve questions and problems with IBM software products. It provides quick access to support-related information. You can use the IBM Support Assistant to help you in the following ways:

- Search through IBM and non-IBM knowledge and information sources across multiple IBM products to answer a question or solve a problem.
- Find additional information through product and support pages, customer news groups and forums, skills and training resources and information about troubleshooting and commonly asked questions.

In addition, you can use the built in Updater facility in IBM Support Assistant to obtain IBM Support Assistant upgrades and new features to add support for additional software products and capabilities as they become available.

General information about the IBM Support Assistant can be found on the IBM Support Assistant home page at http://www.ibm.com/software/support/isa.

Obtaining fixes

A product fix might be available to resolve your problem. To determine what fixes and other updates are available, the following information is available on the respective product support site:

- Latest PTFs for Application Performance Analyzer for z/OS
- Latest PTFs for Debug Tool for z/OS
- Latest PTFs for Fault Analyzer for z/OS
- Latest PTFs for File Export for z/OS
- Latest PTFs for File Manager for z/OS
- Latest fixes for Optim Move for DB2
- Latest PTFs for WebSphere Studio Asset Analyzer for Multiplatforms
- Latest PTFs for Workload Simulator for z/OS and OS/390

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.

For more information about the types of fixes that are available, see the IBM Software Support Handbook at http://techsupport.services.ibm.com/guides/handbook.html.

Receiving support updates through e-mail notification

To receive e-mail notifications about fixes and other software support news, follow the steps below. Additional information is provided at http://www.ibm.com/support/docview.wss?rs=615&uid=swg21172598.

2. Click My notifications in the upper right corner of the page.
3. If you have already registered for My notifications, sign in and skip to the next step. If you have not registered, click register now. Complete the registration form using your e-mail address as your IBM ID and click Submit.
4. In the My notifications tool, click the Subscribe tab to specify products for which you want to receive e-mail updates.
5. To specify Problem Determination Tools products, click Other software and then select the products for which you want to receive e-mail updates, for example, Debug Tool for z/OS and File Manager for z/OS.
6. To specify a COBOL or PL/I compiler, click Rational® and then select the products for which you want to receive e-mail updates, for example, Enterprise COBOL for z/OS.
7. After selecting all products that are of interest to you, scroll to the bottom of the list and click Continue.
8. Determine how you want to save your subscription. You can use the default subscription name or create your own by entering a new name in the Name field. It is recommended that you create your own unique subscription name using a something easily recognized by you. You can create a new folder by entering a folder name in the New field or select an existing folder from the pulldown list. A folder is a container for multiple subscriptions.
9. Specify the types of documents you want and the e-mail notification frequency.

10. Scroll to the bottom of the page and click Submit.

To view your current subscriptions and subscription folders, click My subscriptions.

If you experience problems with the My notifications feature, click the Feedback link in the left navigation panel and follow the instructions provided.

---

**Receiving support updates through RSS feeds**

To receive RSS feeds about fixes and other software support news, go to one of the following web sites:

- RSS feed for Application Performance Analyzer for z/OS
- RSS feed for Debug Tool for z/OS
- RSS feed for Enterprise COBOL for z/OS
- RSS feed for Enterprise PL/I for z/OS
- RSS feed for Fault Analyzer for z/OS
- RSS feed for File Export for z/OS
- RSS feed for File Manager for z/OS
- RSS feed for WebSphere Studio Asset Analyzer
- RSS feed for Workload Simulator for z/OS and OS/390

---

**If you need to contact IBM Software Support**

IBM Software Support provides assistance with product defects.

Before contacting IBM Software Support, your company must have an active IBM software maintenance contract, and you must be authorized to submit problems to IBM. The type of software maintenance contract that you need depends on the type of product you have:

- For IBM distributed software products (including, but not limited to, Tivoli®, Lotus®, and Rational products, as well as DB2 and WebSphere products that run on Windows, or UNIX operating systems), enroll in Passport Advantage® in one of the following ways:
  - Online
    Go to the Passport Advantage Web site at [http://www.lotus.com/services/passport.nsf/WebDocs/Passport_Advantage_Home](http://www.lotus.com/services/passport.nsf/WebDocs/Passport_Advantage_Home) and click How to Enroll.
  - By phone
    For the phone number to call in your country, go to the IBM Software Support Web site at [http://techsupport.services.ibm.com/guides/contacts.html](http://techsupport.services.ibm.com/guides/contacts.html) and click the name of your geographic region.

- For customers with Subscription and Support (S & S) contracts, go to the Software Service Request Web site at [https://techsupport.services.ibm.com/ssr/login](https://techsupport.services.ibm.com/ssr/login)

• For IBM eServer™ software products (including, but not limited to, DB2 and WebSphere products that run in zSeries, pSeries, and iSeries environments), you can purchase a software maintenance agreement by working directly with an IBM sales representative or an IBM Business Partner. For more information about support for eServer software products, go to the IBM Technical Support Advantage Web site at http://www.ibm.com/servers/eserver/techsupport.html.

If you are not sure what type of software maintenance contract you need, call 1-800-IBMSERV (1-800-426-7378) in the United States. From other countries, go to the contacts page of the IBM Software Support Handbook on the Web at http://techsupport.services.ibm.com/guides/contacts.html and click the name of your geographic region for phone numbers of people who provide support for your location.

To contact IBM Software support, follow these steps:
1. “Determining the business impact”
2. “Describing problems and gathering information”
3. “Submitting problems” on page 404

**Determining the business impact**

When you report a problem to IBM, you are asked to supply a severity level. Therefore, you need to understand and assess the business impact of the problem that you are reporting. Use the following criteria:

**Severity 1**
The problem has a critical business impact. You are unable to use the program, resulting in a critical impact on operations. This condition requires an immediate solution.

**Severity 2**
The problem has a significant business impact. The program is usable, but it is severely limited.

**Severity 3**
The problem has some business impact. The program is usable, but less significant features (not critical to operations) are unavailable.

**Severity 4**
The problem has minimal business impact. The problem causes little impact on operations, or a reasonable circumvention to the problem was implemented.

**Describing problems and gathering information**

When describing a problem to IBM, be as specific as possible. Include all relevant background information so that IBM Software Support specialists can help you solve the problem efficiently.

To save time, 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 Application Performance Analyzer for z/OS: http://www.ibm.com/support/docview.wss?rs=2300\n<context=SSFMHB&q1=mustgather&uid=swg21265542&loc=en_US&cs=utf-8&lang=en


If the product does not have a Mustgather document, please 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.

Submitting problems

You can submit your problem to IBM Software Support in one of two ways:

Online
Click Open service request on the IBM Software Support site at http://www.ibm.com/software/support/probsub.html. In the Other support tools section, select IBMLink to open an Electronic Technical Response (ETR). Enter your information into the appropriate problem submission form.

By phone
Call 1-800-IBMSERV (1-800-426-7378) in the United States or, from other countries, go to the contacts page of the IBM Software Support Handbook at http://techsupport.services.ibm.com/guides/contacts.html and click the name of your geographic region.

If the problem you submit is for a software defect or for missing or inaccurate documentation, IBM Software Support creates an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Software Support provides a workaround that you can implement until the APAR is resolved and a fix is delivered. IBM publishes resolved APARs on the
Software Support Web site daily, so that other users who experience the same problem can benefit from the same resolution.

After a Problem Management Record (PMR) is open, you can submit diagnostic MustGather data to IBM using one of the following methods:

- FTP diagnostic data to IBM
- If FTP is not possible, email diagnostic data to techsupport@mainz.ibm.com. You must add PMR xxxxx bbb ccc in the subject line of your email. xxxxx is your PMR number, bbb is your branch office, and ccc is your IBM country code. Click here [http://itcenter.mainz.de.ibm.com/ecurep/mail/subject.html](http://itcenter.mainz.de.ibm.com/ecurep/mail/subject.html) for more details.

Always update your PMR to indicate that data has been sent. You can update your PMR online or by phone as described above.
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 Debug Tool.

The major accessibility features in z/OS enable users to:

- Use assistive technology products such as screen readers and screen magnifier software
- Operate specific or equivalent features by using only the keyboard
- Customize display attributes such as color, contrast, and font size

The IBM System z Enterprise Development Tools & Compilers Information Center, and its related publications, are accessibility-enabled. The accessibility features of the information center are described at http://publib.boulder.ibm.com/infocenter/pdthelp/v1r1/topic/com.ibm.help.doc/accessibility_info.html.

Using assistive technologies

Assistive technology products work with the user interfaces that are found in z/OS. For specific guidance information, consult the documentation for the assistive technology product that you use to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces by using TSO/E or ISPF. Refer to z/OS TSO/E Primer, z/OS TSO/E User’s Guide, and z/OS ISPF User’s Guide Volume 1 for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Accessibility of this document

Information in the following formats of this document is accessible to visually impaired individuals who use a screen reader:

- HTML format when viewed from the IBM System z Enterprise Development Tools & Compilers Information Center
- BookManager® format when viewed with IBM BookManager BookServer (except for syntax diagrams)

Syntax diagrams start with the word Format or the word Fragments. Each diagram is preceded by two images. For the first image, the screen reader will say "Read syntax diagram". The associated link leads to an accessible text diagram. When you return to the document at the second image, the screen reader will say "Skip visual syntax diagram" and has a link to skip around the visible diagram.

For BookManager users only: A screen reader might say the lines, symbols, and words in a diagram, but not in a meaningful way. For example, you might hear "question question dash dash MOVE dash dash plus dash dash"
literal-1 dash dash plus” for part of the MOVE statement. You can enter **Say Next Paragraph** to move quickly through syntax diagrams if your screen reader has that capability.
Notices

This information was developed for products and services offered in the U.S.A. IBM might not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

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 illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or functions of these programs.

Programming interface information

This book is intended to help you debug application programs. This publication documents intended Programming Interfaces that allow you to write programs to obtain the services of Debug Tool.

Trademarks and service marks

IBM, the IBM logo, and ibm.com® are 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, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

LINUX is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT®, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

MasterCraft is a trademark of Tata Consultancy Services Ltd.
Glossary

This glossary defines technical terms and abbreviations used in Debug Tool Reference and Messages documentation. If you do not find the term you are looking for, refer to the IBM Glossary of Computing Terms, located at the IBM Terminology web site:

http://www.ibm.com/ibm/terminology

D

DTCN. Debug Tool Control utility, a CICS transaction that enables the user to identify which CICS programs to debug.

distributing profile. Data that specifies a set of application programs which are to be debugged together.

I

index. A computer storage position or register, the contents of which identify a particular element in a table.

M

minor node. In VTAM, a uniquely defined resource within a major node.

multitasking. A mode of operation that provides for concurrent performance, or interleaved execution of two or more tasks.

O

offset. The number of measuring units from an arbitrary starting point to some other point.

S

Single Point of Control. The control interface that sends commands to one or more members of an IMSplex and receives command responses.

SPOC. See “Single Point of Control.”

statement. (1) An instruction in a program or procedure. (2) In programming languages, a language construct that represents a step in a sequence of actions or a set of declarations.
Bibliography

Debug Tool publications

Using CODE/370 with VS COBOL II and OS PL/I, SC09-1862

Debug Tool for z/OS

You can access Debug Tool publications through the IBM System z Enterprise Development Tools and Compilers information center. You can subscribe to RSS feeds about updates to the information center by following the instructions in the topic "Subscribe to information center updates," which is in the IBM System z Enterprise Development Tools and Compilers information center.

- Debug Tool API User’s Guide and Reference, SC14-7250
- Debug Tool Coverage Utility User’s Guide and Messages, SC14-7247
- Debug Tool Customization Guide, GC14-7257
- Debug Tool Reference and Messages, GC14-7255
- Debug Tool Reference Summary, GC14-7256
- Debug Tool User’s Guide, SC14-7245
- Program Directory for IBM Debug Tool for z/OS, GI11-9130

COBOL and CICS Command Level Conversion Aid for OS/390 & MVS & VM: User’s Guide, SC26-9400-02

- Program Directory for IBM COBOL and CICS Command Level Conversion Aid for OS/390 & MVS & VM, GI10-5080-04
- Japanese Program Directory for IBM COBOL and CICS Command Level Conversion Aid for OS/390 & MVS & VM, GI10-6976-02

High level language publications

z/OS C and C++

- Compiler and Run-Time Migration Guide, GC09-4913
- Curses, SC09-4815
- Language Reference, SC09-4765
- Programming Guide, SC09-4767
- Run-Time Library Reference, SA22-7821
- User’s Guide, SC09-4767

Enterprise COBOL for z/OS, Version 4

- Enterprise COBOL for z/OS Compiler and Runtime Migration Guide, GC27-1409
- Enterprise COBOL for z/OS Customization Guide, SC23-8526
- Enterprise COBOL for z/OS Licensed Program Specifications, GI11-7871
- Enterprise COBOL for z/OS Language Reference, SC23-8528
- Enterprise COBOL for z/OS Programming Guide, SC23-8529

Enterprise COBOL for z/OS and OS/390, Version 3

- Migration Guide, GC27-1409
- Customization, GC27-1410
- Licensed Program Specifications, GC27-1411
- Language Reference, SC27-1408
- Programming Guide, SC27-1412

COBOL for OS/390 & VM

- Compiler and Run-Time Migration Guide, GC26-4764
- Customization under OS/390, GC26-9045
- Language Reference, SC26-9046
- Programming Guide, SC26-9049

Enterprise PL/I for z/OS and OS/390

- Diagnosis Guide, SC27-1459
- Language Reference, SC27-1460
- Licensed Program Specifications, GC27-1456
- Messages and Codes, SC27-1461
- Migration Guide, GC27-1458
- Programming Guide, SC27-1457

VisualAge PL/I for OS/390

- Compiler and Run-Time Migration Guide, SC26-9474
- Diagnosis Guide, SC26-9475
- Language Reference, SC26-9476
- Licensed Program Specifications, GC26-9471
- Messages and Codes, SC26-9478
- Programming Guide, SC26-9473

PL/I for MVS & VM
Related publications

CICS
- Application Programming Guide, SC34-6231
- Application Programming Primer, SC34-0674
- Application Programming Reference, SC34-6232

DB2 Universal Database™ for z/OS
- Administration Guide, SC18-7413
- Application Programming and SQL Guide, SC18-7415
- Command Reference, SC18-7416
- Data Sharing: Planning and Administration, SC18-7417
- Installation Guide, GC18-7418
- Messages and Codes, GC18-7422
- Reference for Remote DRDA® Requesters and Servers, SC18-7424
- Release Planning Guide, SC18-7425
- SQL Reference, SC18-7426
- Utility Guide and Reference, SC18-7427

IMS
- IMS Application Programming: Database Manager, SC27-1286
- IMS Application Programming: EXEC DLI Commands for CICS & IMS, SC27-1288
- IMS Application Programming: Transaction Manager, SC27-1289

TSO/E
- Command Reference, SA22-7782
- Programming Guide, SA22-7788
- System Programming Command Reference, SA22-7793
- User’s Guide, SA22-7794

z/OS
- MVS JCL Reference, SA22-7597
- MVS JCL User’s Guide, SA22-7598
- MVS System Commands, SA22-7627

z/OS Language Environment
- Concepts Guide, SA22-7567
- Customization, SA22-7564
- Debugging Guide, SA22-7560
- Programming Guide, SA22-7561
- Programming Reference, SA22-7562
- Run-Time Migration Guide, GA22-7565
- Vendor Interfaces, SA22-7568
- Writing Interlanguage Communication Applications, SA22-7563

Softcopy publications

Online publications are distributed on CD-ROMs and can be ordered through your IBM representative. Debug Tool User’s Guide, Debug Tool Customization Guide, and Debug Tool Reference and Messages are distributed on the following collection kit:

SK3T-4269

Online publications can also be downloaded from the IBM Web site. Visit the IBM Web site for each product to find online publications for that product.
Index

Special characters

? command 33
.dbg file
   SET DEFAULT DBG command 216
.mdbg file
   SET DEFAULT MDBG command 218
   SET MDBG command 236
%ADDRESS variable 289
%AMODE variable 289
description 289
%BLOCK variable 289
%CAAADDRESS variable 289
%CC variable 289
%CEBR command, CALL 79
%CECI command, CALL 79
%CONDITION variable 290
description 289
%COUNTRY variable 290
%CU variable 290
%DEC built-in function 281
%DUMP command, CALL 80
%EPA variable 290
%EPRBn variable 291
%EPRDn variable 291
%EPRn variable 290
%FA command, CALL 84
%FM command, CALL 85
%FPRBn variable 292
%FPRDn variable 292
%FPRn variable 291
%GENERATION built-in function 282
%GPRGn variable 293
%GPRn variable 293
%HARDWARE variable 294
%HEX built-in function 282
%IF command (programming language neutral) 128
%INSTANCES built-in function 283
   restriction with replay 283
%LINE variable 294
%LOAD variable 294
%LPRBn variable 295
%LPRDn variable 296
%LPRn variable 295
%NLANGUAGE variable 296
%PATHCODE variable 296
description 296
   restriction with replay 296
%PLANGUAGE variable 296
%port_id suboption of TEST run-time option 9
%PROGMASK variable 296
%PROGRAM variable 290, 297
%PSW variable 297
%RC variable 297
%RECURSION built-in function 284
%Rn variable 297
%RUNMODE variable 297
%STATEMENT variable 294
%SUBSYSTEM variable 297, 298
%SYSTEM variable 298
.%VER command, CALL 85
%WHERE built-in function 285

Numerics

64-bit operands, how Debug Tool handles 24

A

abbreviating commands 29
address description 11
ALL suboption of TEST run-time option 4
ALLOCATE command (PL/I) 33
ALLOCATE, AT command (PL/I) 43
allowable comparisons for Debug Tool IF command (COBOL) 132
allowable moves
   for Debug Tool MOVE command 172
   for Debug Tool SET command 261
ANALYZE command (PL/I) 34
APPEARANCE, AT command 44
assembler commands
   assignment 35
   WHERE built-in function 285
assembler expression syntax
   common syntax elements 21
   conditional operators 24
   operators 22
   assembler, definition of x
ASSEMBLER, QUERY command 192
ASSEMBLER, SET command 207
AUTOMONITOR, QUERY command 192
AUTOMONITOR, SET command 207

B

batch mode
   restrictions 194, 241, 249
BEGIN command 76
BEGIN command (PL/I)
   restriction with playback 76
block command (C and C++)
   restriction with playback 77
block_name, description 12
block_spec, description 12
break command (C and C++)
   restriction with playback 78
breakpoint
   AT command, setting with 39
   in unknown compile unit 45
   removing 88
breakpoints
   saving and restoring 39
   CICS-specific information 40
BROWSE MODE, QUERY command 192
C

C and C++

%INSTANCES built-in function 283
%STORAGE function 49
commands
block 77
break 77
do/while 109
for 123
if 130
INPUT 137
SET INTERCEPT 229
SET LONGCUNAME 235
SET WARNING 258
switch 268
while 276
declarations 96
expression 116
HEX built-in function 282
RECURSION built-in function 284
CALL %CEBR command 79
CALL %CECI command 79
CALL %DUMP command 80
CALL entry_name (COBOL) 86
CALL HOGAN 85
CALL procedure 87
summary 78
CALLS, LIST command 145
CHANGE, AT command 48
CHANGE, AT command (remote debug mode) 53
CHANGE, QUERY command 192
CHANGE, SET command 211
CHKSTGV command 88
CLEAR command 88
restriction with playback 91
CLEAR prefix (full-screen mode) 92
restriction with playback 92
CLOSE, WINDOW command 277
COBOL
%STORAGE function 49
AT DATE command 56
commands
CALL entry_name 86
COMPUTE 93
EVALUATE 114
IF 131
INPUT 137
MOVE 171
PERFORM 182
SET 260
SET INTERCEPT 230
SET INTERCEPT (remote debug mode) 231
SET, allowable moves 261
declarations 98

COBOL (continued)

HEX built-in function 282
SET WARNING, to modify variables in optimized code 258
CODEPAGE option vs. VADSCPnnn option 6
COLOR, SET command 212
COLORS, QUERY command 192
COLUMN, SET MONITOR command 238
command
syntax diagrams xi
command suboption of TEST run-time option 7
commands
delimiting 76
commands_file of TEST run-time option 5
commands_file_designator suboption of TEST run-time option 5
COMMENT command 93
common syntax elements 11
comparisons, allowable (IF command for COBOL) 132
compile unit
record of associations, Debug Tool 180
compile_unit_name, description 13
COMPUTE command (COBOL) 93
condition
constants, for C 272
for C 67
condition, description 13
COUNTRY, QUERY command 192
COUNTRY, SET command 215
cu_spec, description 14
CURRENT VIEW, QUERY command 192
CURSOR command 94
cursor commands
CURSOR, AT 55
CURSOR, LIST (full-screen mode) 147
customer support
See Software Support

D

DATATYPE, SET MONITOR command 238
DATE, AT command 56
DBCS
SET DBCS command 215
variable, assigning new value to 172
DBCS, QUERY command 192
DBCS, SET command 215
Debug Tool terminology ix
declarations
for assembler 95
for C and C++ 96
for COBOL 98
for disassembly 95
for non-Language Environment COBOL 95
declarations (C and C++)
restriction with playback 98
DECLARE command (PL/I) 100
restriction with playback 101
DEFAULT DBG, QUERY command 192
DEFAULT DBG, SET command 216
DEFAULT LISTINGS, QUERY command 192
DEFAULT LISTINGS, SET command 217
DEFAULT MDBG, QUERY command 192
DEFAULT MDBG, SET command 218
DEFAULT SCROLL, QUERY command 192
DEFAULT SCROLL, SET command 220
DEFAULT VIEW, QUERY command 192
DEFAULT VIEW, SET command 220
DEFAULT WINDOW, QUERY command 193
DEFAULT WINDOW, SET command 221
disassembly
commands not supported for
GOTO 125
JUMPTO 138
PLAYBACK 184
WHERE built-in function 285
disassembly commands
AT OFFSET 70
declarations 95
DO 108
IF 129
SET DISASSEMBLY 221
while 276
DISASSEMBLY, QUERY command 193
DISASSEMBLY, SET command 221
display point of execution 246
DO command (assembler, disassembly, non-Language Environment COBOL) 108
DO command (PL/I) 109
restriction with playback 111
do/while command (C and C++) 109
restriction with playback 109
disassembly commands
restriction with playback 112
documents, licensed vii
DTSU
See Debug Tool Setup Utility
.
DWARF file
See .dbg file
DYNDDEBUG, QUERY command 193
DYNDDEBUG, SET command 222

E

ECHO, QUERY command 193
ECHO, SET command 224
ENABLE command 112, 113
restriction with playback 112
F
FIND command 116
FINDBP command 121
FREQUENCY, LIST command 153
FREQUENCY, QUERY command 193
FREQUENCY, SET command 227
full-screen mode (continued)
prefix commands 188
QUERY prefix 195
RETRIEVE 199
SET COLOR 212
SET DEFAULT SCROLL 220
SET DEFAULT WINDOW 221
SET KEYS 232
SET LOG NUMBERS 235
SET MONITOR 237
SET MONITOR COLUMN 238
SET MONITOR DATATYPE 238
SET MONITOR LIMIT 238
SET MONITOR NUMBERS 238
SET MONITOR WRAP 238
SET PROMPT 244
SET SAVE 249
SET SCREEN 252
SET SCROLL DISPLAY 253
SET SUFFIX 256
SHOW prefix 263
WINDOW CLOSE 277
WINDOW OPEN 277
WINDOW SIZE 278
WINDOW SWAP 279
WINDOW ZOOM 280
function, Debug Tool
%DEC 281
%GENERATION 282
%HEX 282
%INSTANCES 283
%RECURSION 284
%WHERE 285
functions, Debug Tool
%STORAGE 49
summary 281
G
GENERATION built-in function 282
GLOBAL, AT command 61
GO command 124
GOTO command 125
restriction with COBOL 126
restriction with playback 125
GOTO LABEL command 127
restriction with playback 128
H
HEX built-in function 282
HISTORY, QUERY command 193
HISTORY, SET command 228
HOGAN command, CALL 85
I
IBM Support Assistant, searching for problem resolution 400
IF command
allowable comparisons (for COBOL) 132
for C and C++
restriction with playback 130
for PL/I
restriction with playback 136
IF command (assembler, disassembly, non-Language Environment COBOL) 129
IF command (C and C++) 130
IF command (COBOL) 131
IF command (PL/I) 135
IGNORELINK, QUERY command 193
IMMEDIATE command 136
information centers, searching for problem resolution 399
INPUT command
restriction with playback 137
INPUT command (C, C++, COBOL) 137
INSPPREF suboption of TEST run-time option 8
INSTANCES built-in function 283
INTERCEPT, QUERY command 193
Internet searching for problem resolution 399
invoking See starting
ISPF
SET REFRESH command 247
J
JUMP TO command 138
JUMPTO command 138
restriction with playback 138
JUMPTO LABEL command 139
restriction with playback 140
K
KEYS, SET command 232
knowledge bases, searching for problem resolution 399
L
LABEL, AT command 62
LANGUAGE, SET NATIONAL command 239
LANGUAGE, SET PROGRAMMING command 242
LAST, LIST command 154
LDD
See LOADDEBUGDATA
LDD, QUERY command 193
LDD, SET command 232
licensed documents vii
LIMIT, SET MONITOR command 238
line commands
See prefix commands
LINE NUMBERS, LIST command 159
LINE, AT command 65, 73
LINE, AT command (remote debug mode) 75
LINES, LIST command 160
LIST commands
LIST (blank) 142
LIST AT 142
restriction with playback 144
LIST CALLS 145
Index 417
LIST commands (continued)
   restriction with playback  145
LIST CONTAINER       146
LIST CURSOR (full-screen
   mode)  147
LIST DTCN or CADP    148
LIST expression      149
LIST FREQUENCY       153
LIST LAST            154
LIST MONITOR         155
LIST NAMES           155
LIST ON (PL/I)       157
LIST PROCEDURES     158
LIST REGISTERS
description 158
LIST STATEMENT NUMBERS 159
LIST STATEMENTS      160
LIST STORAGE
description 160
summary 141
LIST, QUERY command 193
LIST, SET command    233
listing
   SET DEFAULT LISTINGS
command 217
LOAD command 162
load_module_name, description 15
LOAD, AT command      65
LOAD, AT command (remote debug
   mode)  67
LOADDEBUGDATA command 164
loaded modules
   clearing 88
LOCATION, QUERY command 193
LOG NUMBERS, QUERY command 193
LOG NUMBERS, SET command 235
LOG, QUERY command    193
log, session
   clearing 88
LOG, SET command      233
Logical Unit Name     7
LONGCUNAME, QUERY command 193
LookAt message retrieval tool viii
loops
   for command (C and C++) 123
LU Name               7
lvalue for C          16

M
MDBG, QUERY command  193
MDBG, SET command    236
MEM
   See MEMORY
MEMORY command 165
message retrieval tool, LookAt viii
MFI suboption of TEST run-time
   option 7
MONITOR COLUMN, QUERY
   command 193
MONITOR COLUMN, SET
   command 238
MONITOR command
description 166
MONITOR DATATYPE, QUERY
   command 193
MONITOR DATATYPE, SET
   command 238
MONITOR LIMIT, QUERY
   command 194
MONITOR LIMIT, SET command 238
MONITOR NUMBERS, QUERY
   command 194
MONITOR NUMBERS, SET
   command 238
MONITOR WRAP, QUERY
   command 194
MONITOR WRAP, SET command 238
MONITOR, LIST command  237
monitors
   clearing 88
MOVE command (COBOL) 171
allowable moves 172
MSGID, QUERY command 194
MSGID, SET command 239
multiple AT conditions, how Debug Tool
   processes 42

N
NAMES DISPLAY command 175
NAMES EXCLUDE command 176
NAMES INCLUDE command 177
NAMES, LIST command 155
NATIONAL LANGUAGE, QUERY
   command 194
NATIONAL LANGUAGE, SET
   command 239
NATLANG run-time option 2
non-Language Environment COBOL
   commands
   assignment 38
   commands not supported for
   AT CALL 47, 61
non-Language Environment COBOL
   commands
   declarations 95
   DO 108
   IF 129
   NONE suboption of TEST run-time
   option 5
   NOPROMPT suboption of TEST run-time
   option 7
   NOTEST suboption of TEST run-time
   option 4
null command 178
NUMBERS, LIST STATEMENT
   command 159
NUMBERS, SET LOG command 235
NUMBERS, SET MONITOR
   command 238

O
OCURRENCE, AT command  67
offset_spec, description 16
OFFSET, AT command (disassembly) 70
ON command (PL/I)  178
   restriction with playback 179
ON, LIST command (PL/I) 157
   restriction with playback 157
OPEN, WINDOW command     277
optimized COBOL
   commands not supported for
   GOTO LABEL 128
JUMPTO LABEL 140
P
PACE, QUERY command    194
PACE, SET command      241
panel
   Source Identification 180
PANEL command (full-screen mode)
description 180
path point
   differences between languages 71
PATH, AT command      71
PERFORM command (COBOL) 182
PFKEY, SET command    241
PFKEYS, QUERY command 194
PL/I
   %INSTANCES built-in function 283
   %STORAGE function 49
commands
   ANALYZE 34
   assignment 38
   AT ALLOCATE 43
   DECLARE 100
   DO 109
   IF 135
   LIST ON 157
   ON 178
   SELECT 204
   SET LONGCUNAME 235
   HEX built-in function 282
   RECURSION built-in function 284
   SET WARNING 258
PL/I, definition of x
playback
   available commands 186
PLAYBACK commands
   BACKWARD 187
   DISABLE 187
   ENABLE 184
   FORWARD 187
   START 185
   STOP 187
   summary 184
PLAYBACK LOCATION, QUERY
   command 194
PLAYBACK, QUERY command 194
POPUP, QUERY command 194
preferences_file_designator suboption of
   TEST run-time option 8
prefix commands
   AT 72
   CLEAR 92
   description 188
   DISABLE 108
   ENABLE 113
   QUERY 195
   SHOW 263
problem determination
   describing problems 403
determining business impact 403
submitting problems 404
PROCEDURE command 189
procedure, CALL 87
PROCEDURES, LIST command 158
PROGRAMMING LANGUAGE, QUERY command 194
PROGRAMMING LANGUAGE, SET command 242
PROMPT suboption of TEST run-time option 7
PROMPT, QUERY command 194
PROMPT, SET command 244

Q
QUIT command 197
QUALIFY RESET command 190
QUALIFY, QUERY command 194
QUERY command 190
QUERY prefix 195
QUIT command 196

R
range of statements, specifying 18
RECURSION built-in function 284
reference
 C lvalue 16
COBOL data name 17
COBOL special register 17
description 16
REFRESH, QUERY command 194
REGISTERS, LIST command 158
remote debug mode 9
tcpip_workstation_id suboption 8
TCPIP& suboption 8
VADTCPIP& suboption 8
repeating breakpoints 42
RESTORE command 198
RESTORE, QUERY command 194
RESTORE, SET command 247
RETRIEVE command
description 199
return to point of execution 246
REWRITE for remote debug mode, SET command 249
REWRITE, QUERY command 194
REWRITE, SET command 249
RUNTO command 200

S
SAVE, QUERY command 194
SAVE, SET command 249
SCREEN, QUERY command 194
SCREEN, SET command 252
SCROLL command
description 201
SCROLL DISPLAY, QUERY command 194
SCROLL, SET DEFAULT command 220
SELECT command (PL/I) 204
separate debug file
 SET DEFAULT DBG command 216
 SET DEFAULT MDBG command 218
 SET MDBG command 236
SEQUENCE, QUERY command 194
SEQUENCE, SET command 253

SET commands
SET ASSEMBLER 207
SET AUTOMONITOR 209
SET CHANGE 211
SET COLOR 212
SET command (COBOL)
description 260
SET COUNTRY 215
SET DBCS 215
SET DEFAULT DBG 216
SET DEFAULT LISTINGS 217
SET DEFAULT MDBG 218
SET DEFAULT SCROLL
description 220
SET DEFAULT VIEW 220
SET DEFAULT WINDOW 221
SET DISASSEMBLY 221
SET DYNDDEBUG 222
SET ECHO 224
SET EQUATE
description 225
SET EXECUTE 226
SET FREQUENCY 227
SET HISTORY 228
SET IGNORELINK
description 228
SET INTERCEPT
description 229, 230, 231
restriction with playback 230, 231
SET KEYS 232
SET LDD 232
SET LIST TABULAR 233
SET LOG 233
SET LOG NUMBERS 235
SET LONGCUNAME
description 235
SET MDBG 236
SET MONITOR 237
SET MONITOR COLUMN 238
SET MONITOR DATATYPE 238
SET MONITOR LIMIT 238
SET MONITOR NUMBERS 238
SET MONITOR WRAP 238
SET MSGID 239
SET NATIONAL LANGUAGE 239
SET PAGE 241
SET PFKEY
description 241
SET PROGRAMMING LANGUAGE 242
SET PROMPT 244
SET QUALIFY
description 244
SET REFRESH
description 247
SET RESTORE 247
SET REWRITE 249
SET REWRITE for remote debug mode 249
SET SAVE 249
SET SCREEN 252
SET SCROLL DISPLAY
description 253
SET SEQUENCE (PL/I) 253
SET SOURCE 254
SET SUFFIX 256
SET TEST 257

SET commands (continued)
SET WARNING
description 258
summary 205
SETS, QUERY command 195
SHOW prefix command 263
SIZE, WINDOW command 278
Software Support
contacting 402
describing problems 403
determining business impact 403
receiving updates 401, 402
submitting problems 404
Source Identification panel, Debug Tool 180
SOURCE, QUERY command 195
SOURCE, SET command 254
Starting Debug Tool
TEST run-time option 3
STATEMENT NUMBERS, LIST command 159
statement_id_range, description 17
statement_id, description 17
STATEMENT, AT command 73
STATEMENT, AT command (remote debug mode) 75
statements
specifying a range 18
STATEMENTS, LIST command 160
STEP command
description 263
stmt_id_spec, description 17
STORAGE command 266
restriction with playback 267
string
searching for 120
SUFFIX, QUERY command 195
SUFFIX, SET command 256
SWAP, WINDOW command 279
switch command (C and C++) 268
restriction with playback 269
taxonomy diagrams
how to read xi
taxonomy, assembler expression
conditional operators 24
operators 22
taxonomy, assembler expressions
common syntax elements 21
syntax, common elements 11
SYSTEM command (z/OS) 270

T
TABULAR, SET LIST command 233
tcpip_workstation_id suboption of TEST run-time option 8
TCP/IP& suboption of TEST run-time option 8
terminal_id suboption of TEST run-time option 7
TERMINATION, AT command 75
terminology, Debug Tool ix
TEST run-time option
%port_id suboption 9
ALL suboption 4
command suboption 7
TEST run-time option (continued)
commands_file suboptions 5
commands_file_designator suboption 5
ERROR suboption 5
INSPPREF suboption 8
MFI suboption 7
NONE suboption 5
NOPROMPT suboption 7
NOTEST suboption 4
preferences_file_designator suboption 8
PROMPT suboption 7
syntax 3
tcpip_workstation_id suboption 8
TCPIP& suboption 8
terminal_id suboption 7
TEST suboption 4
VADTCP& suboption 8
VTAM_LU_id suboption 7
TEST suboption of TEST run-time option 4
TEST, QUERY command 195
TEST, SET command 257
TRAP run-time option 9
TRIGGER command 271
TSO command (z/OS)
description 274

U
USE command 274

V
VADSCPnnn
description of 6
VADSCPnnn option vs. CODEPAGE option 6
VADTCP& suboption of TEST run-time option 8
variable_name description 19
variables
DBCS, assigning new value to 172
modifiable Debug Tool
%EPRBn 291
%EPRDn 291
%EPRn 290
%FPRBn 292
%FPRDn 292
%FPRn 291
%GPRGn 293
%GPRn 293
%LPRBn 295
%LPRDn 296
%LPRn 295
nonmodifiable Debug Tool
%ADDRESS 289
%AMODE 289
%BLOCK 289
%CAAADDRESS 289
%CONDITION 290
%COUNTRY 290
%CU 290
%EPA 290
%HARDWARE 294

W
WARNING, QUERY command 195
WHERE built-in function 285
while command (C and C++) 276
restriction with playback 276
WINDOW command
description 276
WINDOW commands
CLOSE 277
OPEN 277
SIZE 278
SWAP 279
ZOOM 280
WINDOW SIZES, QUERY command 195
WINDOW, SET DEFAULT command 221
workstation debugging
See remote debug mode
WRAP, SET MONITOR command 238
writing assembler instructions,
introduction to 21

Z
z/OS
TSO command 274
ZOOM, WINDOW command 280
Readers’ Comments — We’d Like to Hear from You

Debug Tool for z/OS Reference and Messages Version 10.1

Publication No. GC14-7255-00

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.
Submit your comments using one of these channels:
• Send your comments to the address on the reverse side of this form.
• Send your comments via e-mail to: COMMENTS@US.IBM.COM

If you would like a response from IBM, please fill in the following information:

Name

Address

Company or Organization

Phone No.

E-mail address