

Debug Tool Utilities and Advanced Functions for z/OS



Coverage Utility User's Guide and Messages

Version 7.1

Debug Tool Utilities and Advanced Functions for z/OS



Coverage Utility User's Guide and Messages

Version 7.1

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 223.

First Edition (September 2006)

| This edition applies to Debug Tool Utilities and Advanced Functions for z/OS, Version 7.1 (Program Number 5655-R45), which supports the following compilers:

- IBM Enterprise COBOL for z/OS and OS/390, Version 3 (5655-G53)
- IBM COBOL for OS/390 & VM, Version 2 (5648-A25)
- VisualAge COBOL Millennium Language Extensions for OS/390 & VM, Version 1.0 (5648-MLE)
- IBM COBOL for MVS & VM, Version 1.2 (5688-197)
- VisualAge COBOL Millennium Language Extensions for MVS & VM, Version 1.0 (5654-MLE)
- VS COBOL II, Version 1 Release 4.0 (5668-958, 5688-023)
- OS/VS COBOL, Version 1 Release 2.4 (5740-CB1, 5740-LM1)
- IBM Enterprise PL/I for z/OS and OS/390, Version 3 (5655-H31)
- VisualAge PL/I Compiler, Version 2 Release 2 (5655-B22)
- IBM PL/I for MVS & VM, Version 1.1.1 (5688-235)
- VisualAge PL/I Millennium Language Extensions for MVS & VM, Version 1.0 (5648-MLX)
- OS PL/I Optimizing Compiler, Version 2.3.0 (5668-909, 5668-910, 5668-911)
- OS PL/I Optimizing Compiler, Version 1.5.1 (5734-PL1, 5734-PL3, 5734-LM4, 5734-LM5)
- IBM OS/390 C/C++, Version 2.6 through OS/390 C/C++, Version 2.10 (5647-A01)
- High Level Assembler (HLASM), Version 1 Release 2, 3 or 4 (5696-234)
- Assembler H (HASM), Version 2 (5668-962)

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: www.ibm.com/software/awdtools/debugtool

© Copyright International Business Machines Corporation 1997, 2006. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

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

Summary of changes	xiii
Changes introduced with Debug Tool V7.1	xiii
Changes introduced with Debug Tool Utilities and	
Advanced Functions V7.1	xiv

Part 1. Overview of Debug Tool Coverage Utility 1

Chapter 1. Introduction to Debug Tool Coverage Utility	3
Monitoring coverage: an overview	3
Setup	4
Execution	5
Report	5
Supported compilers and assemblers	6
Requirements	6
Where you can find more information	6
Chapter 2. Getting started	9
Starting the Coverage Utility ISPF dialog	9
Modifying your Coverage Utility defaults	9
Editing your user defaults	10
Resetting your user defaults to the site defaults	13

Part 2. Learning to use Coverage Utility 15

Chapter 3. Learning to use the product	17
Using the supplied samples	17
Running the samples	17
Allocating sample data sets	19
Preparing to produce a sample report	19
Compiling the sample	21
Editing the sample control file	21
Producing a sample summary	22
Creating sample setup JCL	22
Creating sample JCL to start a monitor session	23
Creating JCL for a sample summary report	23
Editing sample JCL to link and run	24
Running the summary sample JCL	24
Example: Summary report for COB01.	26
Example: Summary report for PLI01	27

Example: Summary report for C01.	28
Example: Summary report for ASM01	29
Producing a sample annotated listing report	29
Creating JCL for an annotated listing report	30
Running the annotated sample JCL	30
Example: COBOL annotated listing report	32
Example: PL/I annotated listing report	34
Example: C/C++ annotated listing report	35
Example: Assembler annotated listing report	37

Chapter 4. Samples that are provided with Coverage Utility. 41

User data sets that are required to run the samples	41
COBOL samples	43
PL/I samples	43
C/C++ samples	44
Assembler samples	44

Part 3. Preparing the programs to be monitored 47

Chapter 5. Describing the compile units to be analyzed 49

Editing a control file	49
Contents of the control file	50
Syntax of control file statements	51
INCLUDE statement	52
DEFAULTS statement	52
COBOL statement (compilation unit definition)	53
PL/I statement (compilation unit definition)	55
C statement (compilation unit definition)	56
ASM statement (compilation unit definition)	58
Examples: Control files	60
Example: Control file for a single compilation unit	60
Example: Control file for multiple compilation units.	60
Example: Control file for load module	60

Chapter 6. Preparing to monitor a program 61

Supplying setup input.	61
Setup processing	61
Restrictions on setup input	62
Instrumenting object modules or load modules	62
Creating the setup JCL by using the panels.	63
Determining when to create or submit setup JCL.	65
Compiler options required by Coverage Utility	65
COBOL compiler options required by Coverage Utility	65
PL/I compiler options required by Coverage Utility	66
C/C++ compiler options required by Coverage Utility	67

Assembler options required by Coverage Utility	68
Compiler restrictions imposed by Coverage Utility	68
COBOL compiler restrictions imposed by Coverage Utility	68
PL/I compiler restrictions imposed by Coverage Utility	69
C/C++ compiler restrictions imposed by Coverage Utility	69
Assembler restrictions imposed by Coverage Utility	69
Setup JCL for the compile job stream	70
Parameters for the setup programs	70
EQACUSET	70
EQACUZPT	72
EQACUZPL	72
EQACUZPP	72

Part 4. Running a Coverage Utility monitor session 75

Chapter 7. Monitoring a program 77

Creating the start monitor JCL by using the panels	77
Parameters for the monitor	79
Running multiple user sessions	80
Changing and using IDs	80
Coverage of common modules with multiple user sessions	80
Using performance mode to reduce monitor overhead	82
Monitoring a program that is executing under control of the Debug Tool debugger	83
Restrictions on monitoring programs	83
Restrictions on programs that reside in read-only storage	84
Restrictions on system modes	84

Chapter 8. Monitor commands 85

Issuing commands	85
EQACUOBP (Display breakpoint status)	86
EQACUOID (Add ID)	89
EQACUOPF (Performance mode off)	90
EQACUOPN (Performance mode on)	91
EQACUOQT (Quit)	92
EQACUORE (Reset)	93
EQACUOSA (Display statistics)	94
EQACUOSE (Display sessions)	96
EQACUOSL (Display listings)	97
EQACUOSN (Snapshot)	99
EQACUOSP (Stop)	100

Part 5. Obtaining Coverage Utility reports. 103

Chapter 9. Creating reports 105

Creating summary report JCL by using the panels	106
Creating annotated listing report JCL by using the panels	107
Creating export JCL by using the panels	109

Chapter 10. Summary report 113

Sections of the summary report	113
PROGRAM AREA DATA section	114
UNEXECUTED CODE section	114
BRANCHES THAT HAVE NOT GONE BOTH WAYS section	115
Example: COBOL summary report	115
Example: PL/I summary report	117
Example: C summary report	118
Example: Assembler summary report	119
Suppression of conditional branch coverage with performance mode	119
Example: Summary report with performance mode enabled during setup	120

Chapter 11. Annotated listing report 123

Selecting specific listings to annotate	123
Reducing the size of an annotated listing report	124
Changes in annotation symbols with performance mode	125
Displaying execution counts in an annotated listing report	126
Example: COBOL annotated listing report	127
Example: PL/I annotated listing report	129
Example: C annotated listing report	130
Example: Assembler annotated listing report	131

Chapter 12. Report differences for optimized C/C++ code. 135

The effects of code motion	135
The effects of dead code elimination	135
The effects of statement decomposition	136
The effects of inlining	136
Summary report with inline code	137
Annotated listing report with inline code	138

Chapter 13. Report program parameters 139

Parameters for the summary and report programs	139
Summary program parameters	139
Report program parameters	140
Parameters for the export data program	140

Chapter 14. HTML reports 143

HTML Annotated Listing Report	143
Creating an HTML Annotated Listing Report by using the panel interface	143
Creating an HTML Annotated Listing Report by using the command interface	144
Restrictions on creating an HTML Annotated Listing Report	145
Format of the HTML Annotated Listing Report	145
HTML Targeted Coverage report	146
Creating an HTML Targeted Coverage Report by using the panel interface	147
Creating an HTML Targeted Coverage Report by using the command interface	148
Format of the HTML Targeted Coverage Report	149

	Restrictions on creating an HTML Targeted Coverage Report	149
--	---	-----

Part 6. Dealing with special situations 151

Chapter 15. Using Coverage Utility in a project environment. 153

Creating Coverage Utility files during code development	153
For the coder	153
For the tester	154
Combining test case coverage results	155
Creating the combine JCL by using the panels	156
Rules for combining results.	157
Measuring coverage for individual test cases	158

Chapter 16. Diagnosing monitor problems 159

Solving system 047 abend	159
Solving system 7C1 abend in a user program.	159
Solving protection exception 0C4 (reason code 4) in a user program.	160
Solving system 0F8 abend in a user program.	160
Solving system Fnn abend in a user program.	160
Solving lack of SQA or ECSA space	161
Solving poor performance when measuring conditional branch coverage	161

Part 7. Appendixes 163

Appendix A. Messages 165

Appendix B. Resources and requirements 187

Coverage Utility resources	187
Setup resources.	187
Monitor CSA, SQA, and ECSA usage	188
Report programs	188
Coverage Utility requirements.	188
DDNAME requirements.	189
Data set attributes.	189

Appendix C. DBCS support 191

DBCS requirements for Coverage Utility compilers and assemblers.	191
--	-----

DBCS support with Coverage Utility	191
--	-----

Appendix D. FastPath 193

Creating quick start JCL from the panels	193
Quick start parameters	195
Creating snapshot summary JCL from the panels	198
Creating quick stop JCL from the panels	200
Quick stop parameters	201

Appendix E. Parameters that are common to multiple routines 205

LINECOUNT	206
LOCALE	206
NATLANG	208
Example: Common parameters	208

Appendix F. Exported XML data 209

XML file description	209
Statement or line numbers	214
Execution of statements with breakpoints	214
XML output for in-lined routines.	214
Optional sections	214
XML DTD	215
Example: Exported XML file	217

Accessibility. 221

Using assistive technologies	221
Keyboard navigation of the user interface	221
Accessibility of this document.	221

Notices 223

Copyright license	224
Programming interface information	224
Trademarks and service marks	224

Bibliography. 225

Debug Tool publications.	225
High level language publications.	225
Related publications	225
Softcopy publications.	226

Glossary 227

Index 231

About this document

This document contains information about the code-coverage utility of Debug Tool Utilities and Advanced Functions. The code-coverage utility measures code coverage for programs written in COBOL, PL/I, C/C++, and assembler. This document describes how to set up programs for analysis, how to prepare to monitor a program, the monitor commands, and the reports that you can obtain. It also contains the messages for the code-coverage utility.

The Debug Tool Utilities and Advanced Functions Coverage Utility is referred to throughout this document as the Debug Tool Coverage Utility or Coverage Utility.

Who might use this document

This document is intended for application programmers who use the code-coverage utility of Debug Tool Utilities and Advanced Functions to monitor the code coverage for applications that are written in high-level languages (HLLs) and assembler. Throughout this document, HLLs are referred to as C and C++, COBOL, and PL/I.

To use this document and determine code coverage for a program that is written in one of the supported languages, you need to know how to compile and run such a program.

Accessing z/OS licensed documents on the Internet

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

<http://www.ibm.com/servers/resourceLink>

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

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

<http://www.ibm.com/servers/resourceLink>

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

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

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

Printed licensed documents are not available from IBM.

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

Using LookAt to look up message explanations

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

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

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

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

How this document is organized

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

- Part 1 gives an overview of the Coverage Utility and describes how to start using it.
- Part 2 steps you through the procedures for the utility by using supplied samples and is provided as a means to learn the tool.
- Part 3 describes the control file that you use to describe the programs to be analyzed and monitored and how to modify the file to prepare to monitor a program.
- Part 4 describes how to monitor a program and provides the monitor commands.
- Part 5 describes all the reports that can be produced and how to obtain them.
- Part 6 describes how to deal with special situations, in particular using the Coverage Utility for large projects, incorporating the monitoring of programs into project procedures, and diagnosing monitor problems.
- Part 7 provides reference material: messages, requirements for data sets and resources, DBCS support, the FastPath function, common parameters for routines, and a description of the XML that is exported.

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

How to read syntax diagrams

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

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

Symbols

The following symbols may be displayed in syntax diagrams:

Symbol	Definition
▶—	Indicates the beginning of the syntax diagram.
—→	Indicates that the syntax diagram is continued to the next line.
▶—	Indicates that the syntax is continued from the previous line.
—▶	Indicates the end of the syntax diagram.

Syntax items

Syntax diagrams contain many different items. Syntax items include:

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

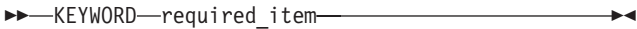
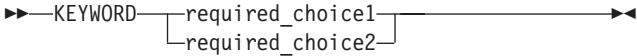
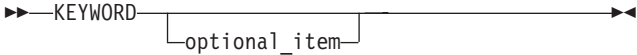
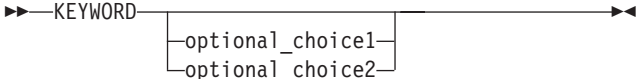
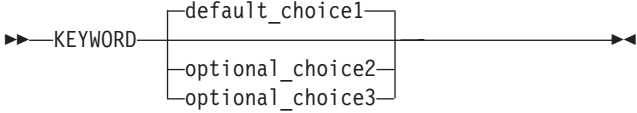

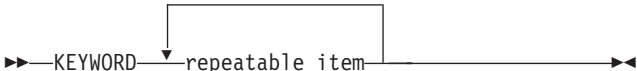
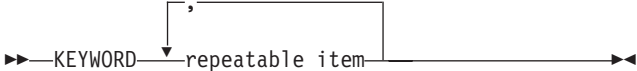

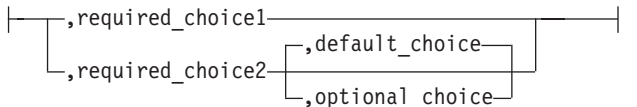
Keywords, variables, and operators may be displayed as required, optional, or default. Fragments, separators, and delimiters may be displayed as required or optional.

Item type	Definition
Required	Required items are displayed on the main path of the horizontal line.
Optional	Optional items are displayed below the main path of the horizontal line.
Default	Default items are displayed above the main path of the horizontal line.

Syntax examples

The following table provides syntax examples.

Table 1. Syntax examples

Item	Syntax example
Required item.	
Required items appear on the main path of the horizontal line. You must specify these items.	
Required choice.	
A required choice (two or more items) appears in a vertical stack on the main path of the horizontal line. You must choose one of the items in the stack.	
Optional item.	
Optional items appear below the main path of the horizontal line.	
Optional choice.	
An optional choice (two or more items) appears in a vertical stack below the main path of the horizontal line. You may choose one of the items in the stack.	
Default.	
Default items appear above the main path of the horizontal line. The remaining items (required or optional) appear on (required) or below (optional) the main path of the horizontal line. The following example displays a default with optional items.	
Variable.	
Variables appear in lowercase italics. They represent names or values.	
Repeatable item.	
An arrow returning to the left above the main path of the horizontal line indicates an item that can be repeated.	
A character within the arrow means you must separate repeated items with that character.	
An arrow returning to the left above a group of repeatable items indicates that one of the items can be selected, or a single item can be repeated.	
Fragment.	
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.	<p data-bbox="797 1562 922 1589">fragment:</p> 

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 Coverage Utility and how they affect this document.

Changes introduced with Debug Tool V7.1

- The following enhancements have been made to the monitoring functions:
 - For COBOL programs, Debug Tool does not prefix the program name to the output, allowing more data to be displayed on the same line.
 - You can now display the value of variables, including members of an array or structure, in a columnar format. Debug Tool provides a new command, SET MONITOR COLUMN, which you can use to indicate that you want the Monitor window to display information in columnar format.
 - Debug Tool now displays a ruler, which indicates the offset from the start of the display to current cursor position.
 - You can now update large variables directly in the Monitor window.
 - You can now use the HEX prefix command on only one member of an array or a sublevel of a structure. Previously, you could use the HEX prefix command only on the entire array or structure.
 - You can now update an array or a structure member without making the full name of the array or structure visible. Previously, you could update an array or structure member only if the full name of the array or structure was visible.
- You can now access source code (C, C++, Enterprise PL/I if a separate debug file is not used) stored in library systems that require data sets to be allocated as a DSORG DA or VSAM data set with the SUBSYS=ssss allocation parameter, where ssss is a subsystem provided by the library system vendor.
Debug Tool provides a method (by using EQAOPTS) that instructs Debug Tool to use the SUBSYS=ssss allocation parameter when it allocates the data set.
- You can now indicate that you want COPE facilities to continue operating while Debug Tool is active. Standardware Corporation's COPE product is used in an IMS environment to deliver some additional capabilities for applications and systems administrators. You use a new option (through EQAOPTS) to enable this behavior.
- You can now indicate what Debug Tool should do if the terminal using the full-screen mode through a VTAM terminal facility or the remote debugger is not available. Use a new option (through EQAOPTS) to select the new behavior.
- Debug Tool has enhanced the LIST, CLEAR, ENABLE, and DISABLE commands to support suspended breakpoints.
- The LIST STORAGE and STORAGE commands have been enhanced so that you can provide a starting byte offset. Previously, Debug Tool used the start of the area of storage allocated to the variable as the starting byte.
- You can now set COBOL level 88 condition variables to TRUE.
- The Debug Tool CICS control utility (DTCN) has been enhanced so that you can temporarily inactivate a profile, then reactivate it at a later time.
- DTCN has been enhanced so you can select CICS tasks to debug based on the client IP name or address.

- A new option TASK has been added to the QUIT DEBUG command to help you terminate debugging sessions that involve pseudo-conversational applications. If you specify the TASK option, Debug Tool terminates immediately. It does not wait until the end of the current CICS pseudo-conversational task, which can be indicated by, for example, an EXEC CICS RETURN TRANSID. When a new task is started in the pseudo-conversation, Debug Tool resumes debugging.
- An option is added to the Debug Tool Utilities primary panel to invoke IBM File Manager (FM) for z/OS functions — FM base, FM/DB2, and FM/IMS. (Note that File Manager is a separately installed product.)
- When you specify a preference or command file in the TEST run time option, you can specify whether Debug Tool interprets the data set name as fully or partially qualified.
- Debug Tool now supports the DEBUG compile option with FORMAT(ISD), which is available with the z/OS C/C++ Version 1.6 (and later) compiler. This option helps you specify the granularity of the compiled-in hooks that the compiler inserts and the amount of debug data to save.
- You can now view the Debug Tool Setup Utility's File Allocation Panel as a full screen panel by using the ShowDD command.
- Two new HTML reports can now be generated. See Chapter 14, "HTML reports," on page 143 for more information about how to create these reports.
- Miscellaneous updates.

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

- The following enhancements have been made to the monitoring functions:
 - You can use a new command, SET MONITOR WRAP, to indicate how you want to display the value of a variable, which is being monitored or automonitored, in the Monitor window. Debug Tool can display the value of a variable in either a wrapped format or on a scrollable line. In a wrapped format, if the value exceeds the length of the display, Debug Tool continues the display on the next line. On a scrollable line, if the value exceed the length of the display, you can scroll left or right to see the rest of the value.
 - You can use a new command, SET MONITOR DATATYPE, to indicate whether you want to display the data type of a variable that is being monitored or automonitored.
- You can display the source for a compile unit or CSECT (CU) before load module containing the CU has been loaded or run. You can work with breakpoints (for example, examine existing breakpoints or set new breakpoints) as you would for a CU that has been loaded or run. Debug Tool applies these breakpoints when the CU becomes active. This feature is also available on WebSphere Developer for zSeries and WebSphere Developer Debugger for zSeries.
- In Debug Tool Utilities, the layout of the panels and arrangement of parameters in the Create Private Message Regions function of the Manage IMS Programs section has been improved.
- You can specify which interface (Main Frame Interface (MFI), WebSphere Developer for zSeries, or WebSphere Developer Debugger for zSeries) to start when you want to debug a DB2 Stored Procedure, IMS Transaction Manager (TM), or batch program. Through an user exit, you can specify a TEST run-time option string that indicates which interface you want to start.
- You can use a utility, EQALANGP, to create a readable listing from a Fault Analyzer side file (IDILANGX or EQALANGX) or a SYSDEBUG file, which is

| generated by using the COBOL TEST(,,SEPARATE) compiler option. If you do not
| keep compiler listings in order to conserve DASD space, EQALANGP can help
| you create a compiler listing that resembles the original compiler listing.

| **Note:** EQALANGP, which is shipped as a component of Debug Tool Utilities
| and Advanced Functions, is functionally equivalent to the IDILANGP
| program shipped as a component of Fault Analyzer for z/OS.

- | • The Debug Tool Coverage Utility (DTCU) SVC installer has been updated to
| help you ensure that the SVC numbers that you choose to use for the DTCU
| breakpoint SVCs do not conflict with the SVC numbers chosen for another
| program.
- | • The Coverage Utility Annotated Listing report is updated for COBOL programs
| so that you can add an HTML version of the report. The HTML version contains
| colored lines that indicate statements that were not executed and recomputed
| statistics based on the annotations in the listing instead of the raw coverage
| data. In addition, a new HTML Targeted Coverage Report, which contains an
| Annotated Listing with lines that were changed between two versions of source
| files, is available for COBOL programs
- | • You can use a new command, SET LIST TABULAR, to indicate how you want the
| output of the LIST displayed. This helps you format the display so it matches
| the display of the MONITOR LIST command.
- | • You can use a new command, DESCRIBE LOADMODS, to indicate how you want to
| display information about all load modules or a specific load module, which are
| known to Debug Tool. Debug Tool displays information about where the load
| module or load modules are loaded, the size, the name, and the programs and
| CSECTs that contained each load module.
- | • Enhancements have been added to better integrate Debug Tool Utilities and
| Advanced Functions with WebSphere Developer for zSeries and WebSphere
| Developer Debugger for zSeries.

Part 1. Overview of Debug Tool Coverage Utility

Chapter 1. Introduction to Debug Tool Coverage

Utility	3
Monitoring coverage: an overview	3
Setup	4
Execution	5
Report	5
Supported compilers and assemblers	6
Requirements	6

Where you can find more information	6
---	---

Chapter 2. Getting started 9

Starting the Coverage Utility ISPF dialog	9
Modifying your Coverage Utility defaults	9
Editing your user defaults	10
Resetting your user defaults to the site defaults	13

This information introduces you to Debug Tool Coverage Utility: what the tool provides, what it requires, and how you can use it to measure code coverage in your application testing. In this part you can learn the basics of running Coverage Utility from setup to generating reports. New users are encouraged to read this part to learn the basics of the tool, including how to use the ISPF dialog and panels and to set defaults.

Chapter 1. Introduction to Debug Tool Coverage Utility

Debug Tool Coverage Utility (Coverage Utility) is a tool that measures test coverage in application programs that have the following characteristics:

- Written in the COBOL, PL/I, C/C++, and assembler languages
- Compiled by certain IBM® COBOL, PL/I, and C/C++ compilers or assembled by the High Level Assembler or Assembler H

Coverage Utility enables you to run application programs in a test environment and retrieve information to determine which code statements have been executed. This process is called measuring test case coverage.

Coverage Utility has the following advantages:

- Low overhead

For a test case coverage run, Coverage Utility typically adds very little to the execution time of your program. Coverage Utility inserts SVC instructions into your application object modules as *breakpoints*, and then is given control by MVS™ when these SVCs are executed. Most breakpoints are removed after their first execution. The increase in test program execution time is minimal, because of this technique.

- Panel-driven user interface

You can use an ISPF panel-driven interface to create JCL for executing Coverage Utility programs.

This section contains the following topics:

- “Monitoring coverage: an overview”
- “Supported compilers and assemblers” on page 6
- “Requirements” on page 6
- “Where you can find more information” on page 6

Monitoring coverage: an overview

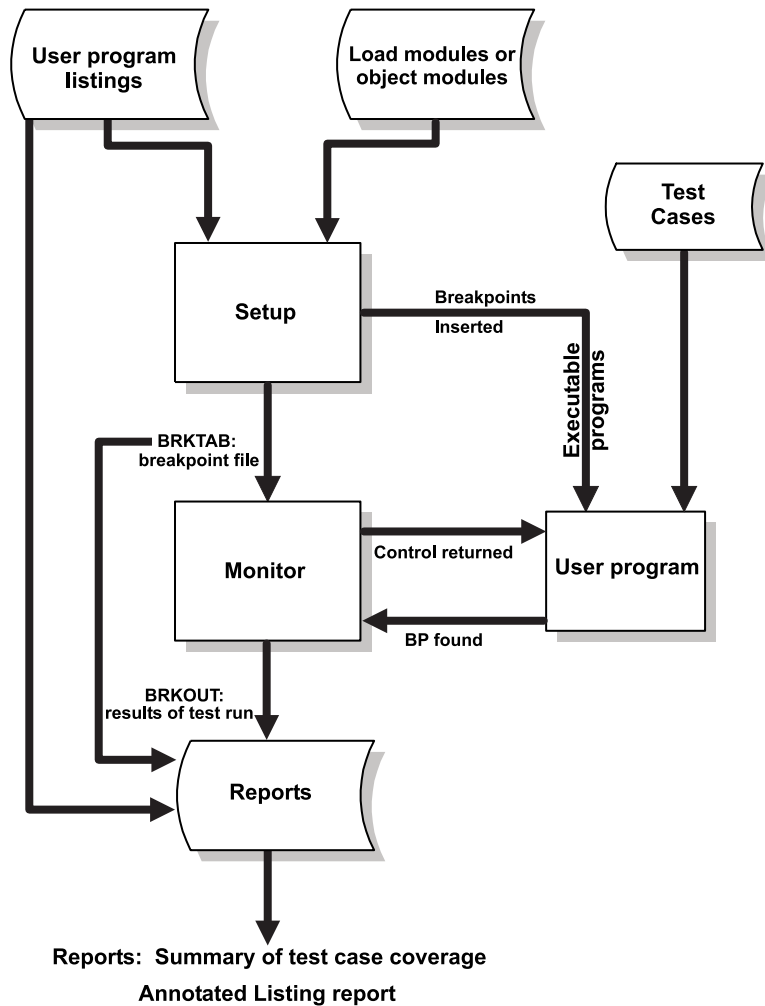
Running Coverage Utility consists of the following steps.

1. Setup — Prepare to monitor programs.
 - a. Compile the source code that you want to analyze, using the required compiler options.
 - b. Generate Coverage Utility JCL, by using the Coverage Utility ISPF dialog:
 - 1) Edit the Coverage Utility control file.
 - 2) Create the setup JCL.
 - 3) Create the start monitor JCL.
 - 4) Create the report or summary JCL.
 - c. Edit the link-edit JCL to include the modified object modules that are created when the setup JCL is run. Alternatively, you can instrument load modules after your build process.
 - d. Edit the GO JCL (or program invocation) to point to the instrumented load module that was provided in step 1c.
2. Execution — Run a monitor session.
 - a. Run the setup JCL (created in step 1b2).
 - b. Run the link-edit JCL (created in step 1c).
 - c. Run the JCL to start a monitor session (created in step 1b3).

- d. Run your application using the load modules from step 2b.
- e. Stop the monitor session (with the EQACUOSP command).
3. Report — Obtain Coverage Utility reports.
 - a. Run the report or summary JCL (created in step 1b4).
 - b. Optional: Run the export utility to save the output in XML format for use by other programs.

If you change your program and want to rerun the test cases, you must repeat step 1a using the changed source code, and then complete steps 1b through 3a again.

This is a flow diagram of the entire process:



Setup

Coverage Utility setup analyzes assembler statements that are included in the compiler or assembler output listings. From this analysis, Coverage Utility determines where to insert breakpoints in disk-resident copies of the object or load modules that you want to examine, and then inserts them.

To run setup, you need:

- Source/assembler listings of the object modules
- The object modules or load modules you want to test

The setup step produces:

- Modified test programs that contain breakpoints
- A file of breakpoint-related information (called a BRKTAB in this book) that is required for the monitor program in the execution step

Execution

If you instrument object modules, you must link the modified object modules into an executable load module.

To run your program under Coverage Utility, you must first start a monitor session, and then run your test case programs. As breakpoints are encountered, the monitor gains control, updates test case coverage statistics, and then returns control to your program. After your test cases have completed, stop the monitor session. The results are written to a file called BRKOUT.

Setup inserts reserved supervisor call (SVC) instructions as breakpoints. The monitor is given control by MVS when these SVC instructions are executed in a program.

Two SVC instructions are used, one for two-byte instructions, and one for four- or six-byte instructions. During installation, the monitor is installed as the handler for the two SVC instructions used as breakpoints.

Report

The Coverage Utility report programs use the results from the monitor to produce a summary report and an annotated listing report. You can print a summary report of overall test case coverage and an annotated listing report for each module tested.

To run reports you need:

- The BRKTAB data set from the setup step
- The BRKOUT data set from the execution step
- Any listings that you want to annotate

For each COBOL paragraph; PL/I procedure, ON-unit, Begin-block; C/C++ function; or assembler CSECT, Coverage Utility can provide you with the following information:

- The percentage of statements that have executed and a list of unexecuted statements
- The percentage of conditional branches that have executed and a list of conditional branches that have not executed in both directions.
- Annotated listing report that shows the execution status of each statement.

To process the monitor output by using another program, you must first export the coverage data in a form that the program understands. You can use the export utility to export the coverage data in an XML format.

To create this XML format output you need:

- The BRKTAB data set from the setup step
- The BRKOUT data set from the execution step

Related tasks

Part 5, "Obtaining Coverage Utility reports," on page 103

Supported compilers and assemblers

Coverage Utility supports applications that are generated by the following compilers and assemblers:

- IBM Enterprise COBOL for z/OS and OS/390®, Version 3 (5655-G53)
- IBM COBOL for OS/390 & VM, Version 2 (5648-A25)
- VisualAge® COBOL Millennium Language Extensions for OS/390 & VM, Version 1.0 (5648-MLE)
- IBM COBOL for MVS & VM, Version 1.2 (5688-197)
- VisualAge COBOL Millennium Language Extensions for MVS & VM, Version 1.0 (5654-MLE)
- VS COBOL II, Version 1 Release 4.0 (5668-958 , 5688-023)
- OS/VS COBOL, Version 1 Release 2.4 (5740-CB1, 5740-LM1)
- IBM Enterprise PL/I for z/OS and OS/390, Version 3 (5655-H31)
- VisualAge PL/I Compiler, Version 2 Release 2 (5655-B22)
- IBM PL/I for MVS & VM, Version 1.1.1 (5688-235)
- VisualAge PL/I Millennium Language Extensions for MVS & VM, Version 1.0 (5648-MLX)
- OS PL/I Optimizing Compiler, Version 2.3.0 (5668-909, 5668-910, 5668-911)
- OS PL/I Optimizing Compiler, Version 1.5.1 (5734-PL1, 5734-PL3, 5734-LM4, 5734-LM5)
- IBM OS/390 C/C++, Version 2.6 through Version 2.10 (5647-A01)
- High Level Assembler (HLASM), Version 1 Release 2, 3 or 4 (5696-234)
- Assembler H (HASM), Version 2 (5668-962)

Requirements

Coverage Utility has the following requirements:

- Coverage Utility runs under MVS and uses ISPF services to display dialogs and to produce the JCL to run the Coverage Utility steps.
- As part of its input, Coverage Utility requires listings that have been created by the application program compilers and assemblers that it supports. These compilers and assemblers offer options that enable you to include assembler statements in the listings. Coverage Utility uses these statements to determine where to insert breakpoints.
- Coverage Utility also requires the application program object modules or load modules as input. Coverage Utility creates copies of these object modules or load modules with breakpoints inserted into them.

Related references

Appendix B, “Resources and requirements,” on page 187

Where you can find more information

Refer to the following sections for additional information.

For information about ...	See ...
Installing Debug Tool Coverage Utility on your system	<i>Debug Tool Customization Guide</i>
Samples of Coverage Utility test case coverage, including sample reports	Chapter 3, “Learning to use the product,” on page 17

For information about ...	See ...
Setting up the table of breakpoints from the listings	Chapter 6, "Preparing to monitor a program," on page 61
Starting the monitor session and running test cases on your programs	Chapter 7, "Monitoring a program," on page 77
Commands that control the monitor program	Chapter 8, "Monitor commands," on page 85
Reports about the test run	Part 5, "Obtaining Coverage Utility reports," on page 103
Using Coverage Utility in a large project environment	Chapter 15, "Using Coverage Utility in a project environment," on page 153
Coverage Utility messages	Appendix A, "Messages," on page 165
System resources needed by Coverage Utility	Appendix B, "Resources and requirements," on page 187
DBCS support	Appendix C, "DBCS support," on page 191

Chapter 2. Getting started

This section contains topics describing how to access the Coverage Utility ISPF dialog and the Coverage Utility user customization procedures. These topics are of most interest when you are new to Coverage Utility.

- “Starting the Coverage Utility ISPF dialog”
- “Modifying your Coverage Utility defaults”

Starting the Coverage Utility ISPF dialog

To start the Coverage Utility ISPF dialog, use one of the following methods:

- If an option was installed to access the Debug Tool primary options ISPF panel from an existing panel, select that option by using instructions from the installer.
- If the Debug Tool Utilities and Advanced Functions data sets were installed into your normal logon procedure, issue the following command from ISPF option 6:

```
EQASTART common_parameters
```

- If Debug Tool was not installed in your ISPF environment, issue this command from ISPF option 6:

```
EX 'hi_lev_qual.SEQAEXEC(EQASTART)' 'common_parameters'
```

Then select the option for Coverage Utility.

You can use *common_parameters* to specify any of the parameters that are common to multiple routines (LINECOUNT, LOCALE, and NATLANG). Separate multiple parameters by blanks. If you specify any of these *common_parameters*, your settings are remembered by EQASTART, and they become the default on subsequent invocations of EQASTART when you do not specify parameters.

The next panel that you see is this Debug Tool Coverage Utility panel:

```
----- Debug Tool Coverage Utility -----  
Option ==>  
  
0 Defaults      Manipulate defaults  
1 CntlFile     Work with the Control File  
2 Setup        Create JCL for Setup  
3 StartMon     Create JCL to Start the Monitor  
4 Reports      Create Reports  
5 Monitor      Control the Monitor  
6 FastPath     FastPath  
  
Enter X to Terminate
```

Related references

Appendix E, “Parameters that are common to multiple routines,” on page 205

Modifying your Coverage Utility defaults

When Debug Tool Coverage Utility is installed, defaults are established for information such as the high-level qualifier to use to construct names for user and project data sets. Typically, you do not have to change these defaults for your user ID. However, you can use the Coverage Utility panels to change your personal defaults in the following ways:

- Edit your defaults.
- Reset your defaults to the site defaults.

In addition, you can save your current defaults or set all of your defaults from a previously saved copy by using the following functions:

- Export defaults to a sequential data set.
- Import defaults from a sequential data set.

New release: If you are changing from one release of Coverage Utility to another, it is recommended that you use Defaults RESET (Coverage Utility option 0.2) and reenter any personal changes.

To change your Coverage Utility user defaults, complete the following steps:

1. Start the Coverage Utility ISPF dialog.
2. Select option 0 from the Debug Tool Coverage Utility panel to specify your Coverage Utility user default values. The Manipulate Defaults panel is displayed:

```

----- Manipulate Defaults -----
Option ==>

1  EDIT           Edit defaults
2  RESET          Reset defaults to site defaults
3  IMPORT         Import defaults from a sequential dataset
4  EXPORT         Export defaults to a sequential dataset

Enter END to Terminate

Import | Export Dataset (Options 3 and 4 only):
Data Set Name . . . .

```

You can change your Coverage Utility user defaults by using the options and **data set name** field on this panel.

Related tasks

“Starting the Coverage Utility ISPF dialog” on page 9

Editing your user defaults

To edit your user defaults do these steps:

1. Select option 1 on the Manipulate Defaults panel. The scrollable Edit Defaults panel is displayed:

```

----- Edit Defaults -----
Command ==>

Enter END (to Exit and Save changes) or CANCEL (to Exit without saving)
----- General Defaults -----
Project Qualifier . . . . YOUNG.SAMPLE
Use Pgm Name for File Name YES (Yes|No)
Program Name . . . . . COB01
JCL Output Dsn . . . . . 'YOUNG.SAMPLE.JCL'
  Type . . . . . JCL
  DSORG . . . . . PDS (SEQ|PDS)
  Alloc Parms. . . . . LRECL(80) RECFM(F B) BLKSIZE(0)
                        TRACKS SPACE(10 10)
1st JOBLIB Dsn . . . . . 'EQAW.SEQAMOD'
2nd Alternate JOBLIB Dsn .
3rd Alternate JOBLIB Dsn .
4th Alternate JOBLIB Dsn .
5th Alternate JOBLIB Dsn .
6th Alternate JOBLIB Dsn .
REXX Dsn . . . . . 'EQAW.SEQAEXEC'
Sample Dsn . . . . . 'EQAW.SEQASAMP'
Display Messages . . . . I (S|E|W|R|I)
Log Messages . . . . . I (S|E|W|R|I)
Control File Dsn . . . . 'YOUNG.SAMPLE.DTCU(COB01)'
  Type . . . . . DTCU
  DSORG . . . . . PDS (SEQ|PDS)
  Alloc Parms. . . . . LRECL(255) RECFM(V B) BLKSIZE(0)
                        TRACKS SPACE(10 10)
  DD Parms . . . . . SPACE=(TRK,(2,2)),
                        DCB=(RECFM=VB,LRECL=255,BLKSIZE=0)
Type of Control File . . COBOL (COBOL|PL/I|C|ASM)
----- Setup Defaults -----
Jobcard Name . . . . . YOUNG
Jobcard Operands . . . . (12345678),
                        YOUNG,NOTIFY=YOUNG,USER=YOUNG,
                        MSGCLASS=H,CLASS=A,REGION=32M,TIME=1

JES Control Cards. . . . .

Breakpoint Table Dsn . . 'YOUNG.SAMPLE.COB01.BRKTAB'
  Type . . . . . BRKTAB
  DSORG . . . . . SEQ (SEQ|PDS)
  Alloc Parms. . . . . LRECL(256) RECFM(F B) BLKSIZE(0)
                        TRACKS SPACE(2 2)
  DD Parms . . . . . SPACE=(TRK,(2,2)),
                        DCB=(DSORG=PS,RECFM=FB,LRECL=256,BLKSIZE=0)
SVC number for 4 byte BP . FF (in HEX)
Performance Mode . . . . YES (Yes|No)
Debug Mode . . . . . NO (Yes|No)
Frequency Count Mode . . NO (Yes|No)
----- Monitor Defaults -----
Loadlib Dsn . . . . . 'EQAW.SEQAMOD'
Jobcard Name . . . . . YOUNG
Jobcard Operands . . . . (12345678),
                        YOUNG,NOTIFY=YOUNG,USER=YOUNG,TIME=1
                        MSGCLASS=H,CLASS=A,REGION=32M,MSGLEVEL=(1,1)

```

```

SVC number for 2 byte BP . FE          (in HEX)
JES Control Cards. . . . .

Breakout Table Dsn . . . . 'YOUNG.SAMPLE.COB01.BRKOUT'
  Type . . . . . BRKOUT
  DSORG. . . . . SEQ          (SEQ|PDS)
  Alloc Parms. . . . . LRECL(256) RECFM(F B) BLKSIZE(0)
                        TRACKS SPACE(3 3)
  DD Parms . . . . . SPACE=(TRK,(3,3)),
                        DCB=(DSORG=PS,RECFM=FB,LRECL=256,BLKSIZE=0)
----- Report Defaults -----
Combined Cntl Dsn. . . . 'YOUNG.SAMPLE.CBCTL(COB01)'
  Type . . . . . CBCTL
  DSORG. . . . . PDS          (SEQ|PDS)
Combined Breakout Dsn. . . 'YOUNG.SAMPLE.COB01.CMBOUT'
  Type . . . . . CMBOUT
  DSORG. . . . . SEQ          (SEQ|PDS)
Jobcard Name . . . . . YOUNG
Jobcard Operands . . . . . (12345678),
                        YOUNG,NOTIFY=YOUNG,USER=YOUNG,
                        MSGCLASS=H,CLASS=A,REGION=32M,TIME=1

JES Control Cards. . . . .

Report File Dsn. . . . . 'YOUNG.SAMPLE.COB01.REPORT'
Summary File Dsn . . . . 'YOUNG.SAMPLE.COB01.SUMMARY'
  Report File Type . . . . REPORT
  Summary File Type. . . . SUMMARY
  DSORG. . . . . SEQ          (SEQ|PDS)
  Alloc Parms. . . . . LRECL(133) RECFM(F B A) BLKSIZE(27930)
                        TRACKS SPACE(10 10)
  DD Parms . . . . . SPACE=(TRK,(10,10)),
                        DCB=(DSORG=PS,RECFM=FBA,LRECL=133,BLKSIZE=27930)
Summary Type . . . . . INTERNAL (Internal|External)
Summary Assembler Stmts. . YES
Summary Inline . . . . . N      (I|N)
Annotation Symbols . . . . :->V%0& (* for default)
Report User Options. . . . A    (A|U)
Print Report File Dataset. YES (Yes|No)
Exported XML Dsn . . . . 'YOUNG.SAMPLE.COB01.XML'
  XML File Type. . . . . XML
  DSORG. . . . . SEQ          (SEQ|PDS)
  Alloc Parms. . . . . LRECL(32756) RECFM(V B) BLKSIZE(32760)
                        TRACKS SPACE(10 10)
  DD Parms . . . . . SPACE=(TRK,(10,10)),
                        DCB=(DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760)
XML Branch Analysis. . . . NO    (Yes|No)
XML Frequency. . . . . NO      (Yes|No)
XML BP Details . . . . . NO    (Yes|No)

```

2. Change the Project Qualifier value to the high-level qualifier that you want Coverage Utility to use to construct names for user and project data sets.
3. To generate or build any data set names automatically, ensure that Use Pgm Name For File Name is set to Yes. Coverage Utility uses the project qualifier, the program name, and the specified values for type and DSORG for each data set to build names of the following forms:

- Sequential data sets:

```
'proj_qual.program_name.file_type'
```

For example: 'YOUNG.SAMPLE.COB01.BRKTAB'

- Partitioned data sets:

```
'proj_qual.file_type(program_name)'
```

For example: 'YOUNG.SAMPLE.BRKTAB(COB01)'

If you specify No for Use Pgm Name For File Name, Coverage Utility does not automatically build or change any data set names.

Resetting your user defaults to the site defaults

To reset your user defaults to the site defaults follow these steps:

1. Select option 2 from the Manipulate Defaults panel. The Reset Defaults to Site Defaults panel is displayed:

```
----- Reset Defaults to Site Defaults -----
Command ==>

Project Qualifier. . . . . YOUNG.SAMPLE
Program Name . . . . . COB01

Enter ENTER to Reset Defaults
Enter END to Cancel and Terminate
```

The fields for the panel are as follows:

Project Qualifier

The qualifier to be used to construct data set names for user and project files when you set **Use Program Name for File Name** to Yes.

Program Name

The program-specific qualifier to be used when constructing data set names for user and project files when you set **Use Program Name for File Name** to Yes.

2. If you want to reset all of your user defaults to the site defaults using the project high-level qualifier and the program name specified in the panel, press Enter. If you do not want to reset your defaults, press the End key (PF3) to return to the previous panel.

Part 2. Learning to use Coverage Utility

Chapter 3. Learning to use the product	17	Producing a sample annotated listing report	29
Using the supplied samples	17	Creating JCL for an annotated listing report	30
Running the samples	17	Running the annotated sample JCL	30
Allocating sample data sets	19	Example: COBOL annotated listing report	32
Preparing to produce a sample report	19	Example: PL/I annotated listing report	34
Compiling the sample	21	Example: C/C++ annotated listing report	35
Editing the sample control file	21	Example: Assembler annotated listing report	37
Producing a sample summary	22		
Creating sample setup JCL	22	Chapter 4. Samples that are provided with	
Creating sample JCL to start a monitor session	23	Coverage Utility	41
Creating JCL for a sample summary report	23	User data sets that are required to run the samples	41
Editing sample JCL to link and run	24	COBOL samples	43
Running the summary sample JCL	24	PL/I samples	43
Example: Summary report for COB01.	26	C/C++ samples	44
Example: Summary report for PLI01	27	Assembler samples	44
Example: Summary report for C01.	28		
Example: Summary report for ASM01	29		

Coverage Utility comes with samples that you can use to generate test case coverage reports. Use these samples to learn the functions and capabilities of the tool, to become familiar with how the process works, and to learn what reports you can generate. This part also describes the samples and how to set up your own environment in which to run the samples.

Chapter 3. Learning to use the product

This section describes generating Coverage Utility test case coverage reports by using the samples provided with Coverage Utility:

- “Using the supplied samples”
- “Preparing to produce a sample report” on page 19
- “Producing a sample summary” on page 22
- “Producing a sample annotated listing report” on page 29

If you are a new user, you are encouraged to run one of the samples as a tutorial. In this way you can see how the process works and become familiar with the Coverage Utility outputs.

Coverage Utility provides statistics about each program area (PA) in your programs. A PA can be any of the following:

- A COBOL paragraph
- A PL/I external or internal procedure, an ON-unit, or a Begin-block
- A C or C++ function
- An assembler CSECT

Coverage Utility does not have to examine all object modules in your program. You can select which areas of the program you want to test. You can also test multiple programs (load modules) simultaneously.

Related references

Chapter 4, “Samples that are provided with Coverage Utility,” on page 41

Using the supplied samples

Running the samples generates a summary of test case coverage and annotated listing report for the sample that you choose from the following programs:

COB01	COBOL sample
PLI01	PL/I sample
C01	C/C++ sample
ASM01	Assembler sample

Use the appropriate program name from the above list for the sample name.

Related tasks

“Allocating sample data sets” on page 19

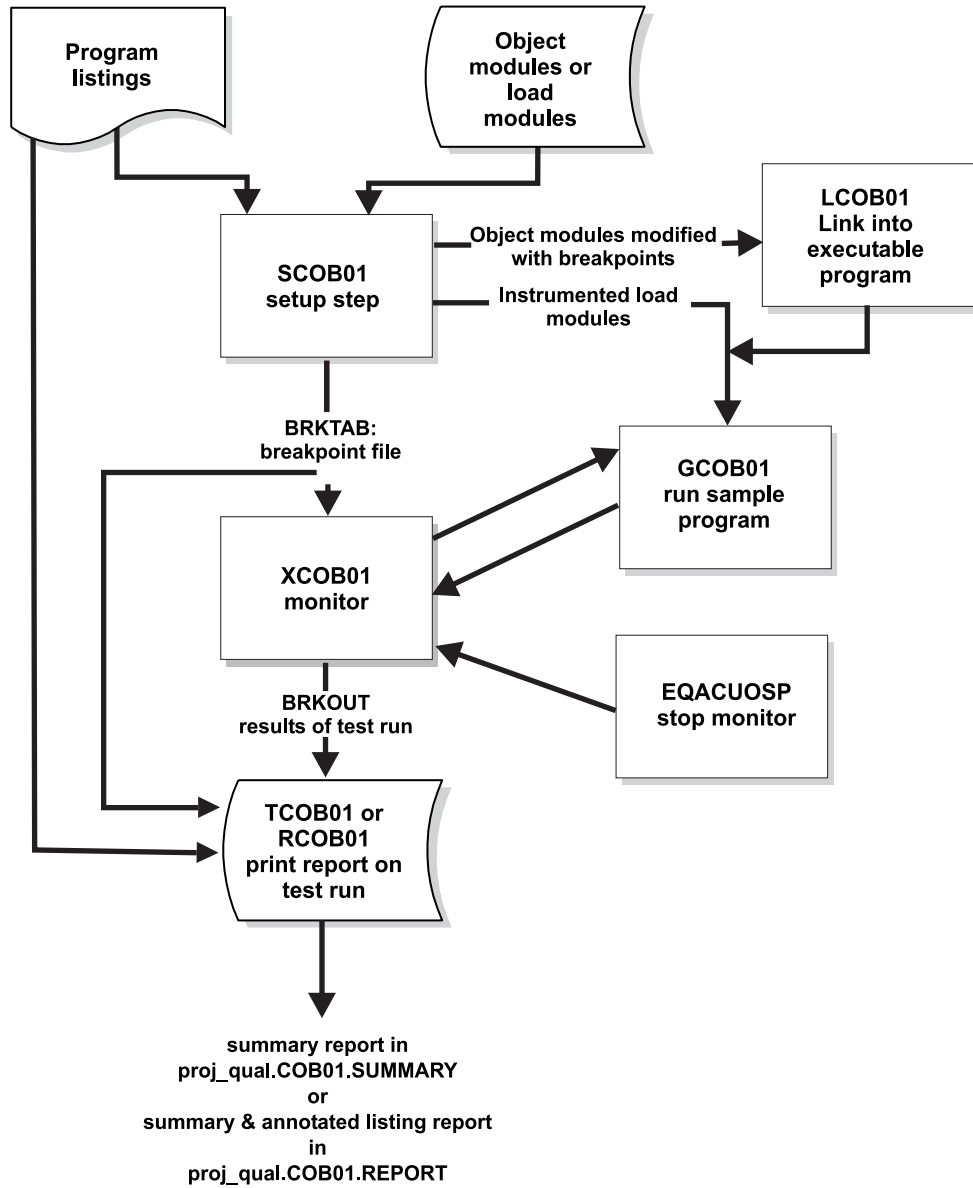
“Running the samples”

Related references

Chapter 4, “Samples that are provided with Coverage Utility,” on page 41

Running the samples

A flow diagram of the steps that you need to follow to run these samples is shown here. The names in the steps are the member names of the JCL for the step.



JCL member names: The JCL member names for the steps, such as SCOB01, depend on the sample that you are running. For example, the setup JCL for the PLI01 test case is SPLI01.

DD name: The names outside the boxes in the figure (for example, BRKTAB) correspond to the DD names in the created JCL.

The example scenarios in this section use the Coverage Utility ISPF dialog to create the JCL to run the Coverage Utility steps. This ISPF dialog is provided as an aid in creating the JCL. After you have created the JCL for a test environment, you do not have to recreate it from the dialog. You do not have to use the ISPF dialog to use Coverage Utility. In a typical user test environment, you can incorporate the creation of the JCL into your procedures.

These samples describe instrumenting object modules from listings. You can similarly instrument load modules.

Related tasks

Chapter 15, “Using Coverage Utility in a project environment,” on page 153
“Instrumenting object modules or load modules” on page 62

Allocating sample data sets

Before you can run a sample as described in the rest of this section, you first need to create data sets and copy members from *hi_lev_qual.SEQASAMP*, as described in the following steps:

1. Allocate your personal sample data sets as described in “User data sets that are required to run the samples” on page 41.

Allocate at least 10 tracks for each data set. Partitioned data sets should have at least 10 blocks for the directory.

2. Copy the sample members shown in tables in the following sections from *hi_lev_qual.SEQASAMP* into the appropriate data set, renaming the members as described.

- “COBOL samples” on page 43
- “C/C++ samples” on page 44
- “PL/I samples” on page 43
- “Assembler samples” on page 44

The naming convention shown in the tables assumes that you select only one compiler version for each language.

3. Edit the Coverage Utility control card members that you copied and change the high-level qualifier of each sample data set name to your TSO prefix.
4. Edit the JCL members that you copied and do the following:
 - a. Change the high-level qualifier of each sample data set name to your TSO prefix.
 - b. Change the compiler and link-edit library data set names to match the compiler and link-edit names of your system.
 - c. Change the job cards to match your local conventions.

Preparing to produce a sample report

To prepare a sample to produce a summary report or an annotated listing of the sample, perform the following steps. **These steps are described in more detail in topics that follow this one.** After you complete these steps, decide which report you want to generate, summary or annotated, and follow the steps in the appropriate topic.

1. Compile the sample source. The compilation produces listings that include the assembler statements that Coverage Utility needs.

Make sure to use the required compiler options.
2. Start the Coverage Utility ISPF dialog; the Debug Tool Coverage Utility panel is displayed:

```

----- Debug Tool Coverage Utility -----
Option ==>

0 Defaults      Manipulate defaults
1 CntlFile     Work with the Control File
2 Setup        Create JCL for Setup
3 StartMon     Create JCL to Start the Monitor
4 Reports      Create Reports
5 Monitor      Control the Monitor
6 FastPath     FastPath

Enter X to Terminate

```

- a. Edit the Coverage Utility control file.
Verify that the control file includes the listings of the object modules that you want to test.
 - b. Create the setup JCL.
Create the JCL that enables you to produce a file that contains breakpoint data and that modifies copies of your object modules by inserting breakpoints.
 - c. Create the JCL to start a monitor session.
 - d. Create the JCL for a report.
Create the JCL to produce a summary report or the annotated listing report after the sample runs.
3. End the Coverage Utility ISPF dialog by pressing the End key (PF3) to close the Debug Tool Coverage Utility panel and then the End key to close the Debug Tool primary options panel.
 4. Edit the JCL to link the modified object modules.
After the setup step, and before you start the monitor, you must link the modified object modules into an executable program that you can test. Edit the link JCL and specify the library that will contain the modified object modules for the OBJECT ddname and the library that will contain the modified load module for the SYSLMOD ddname.
 5. Edit the JCL to run the GO step.
Edit the JCL to run your program. Specify the same modified load module as in step 4.
 6. Run the JCL.
Run the created JCL files for the sample in the correct order.

Related tasks

- “Compiling the sample”
- “Editing the sample control file”
- “Creating sample setup JCL” on page 22
- “Creating sample JCL to start a monitor session” on page 23
- “Producing a sample summary” on page 22
- “Producing a sample annotated listing report” on page 29

Compiling the sample

To compile your source code, do these steps:

1. Edit the sample compile JCL that you copied from *hi_lev_qual.SEQASAMP* so that it will run on your system:
 - a. Change the job card to match your system requirements.
 - b. Update the compiler data set names to match the data set names for your local compiler.
 - c. Update the sample data set names to match the data set names for your sample data set.
 - d. Uncomment the member names for the sample that you are compiling.
2. Run the JCL.
3. Make sure that all steps run with a return code of zero and that the listings and object modules are created.

Editing the sample control file

Coverage Utility uses assembler statements from the compiler listings to determine where to insert breakpoints. You supply the names of the listing files in the Coverage Utility control file.

The Coverage Utility control file for the examples are shown here. These examples show how to instrument object modules.

COB01	PLI01
Defaults ListDsn=YOUNG.SAMPLE.COBOLST(*), LoadMod=COB01, FromObjDsn=YOUNG.SAMPLE.OBJ, ToObjDsn=YOUNG.SAMPLE.ZAPOBJ COBOL ListMember=COB01A COBOL ListMember=COB01B COBOL ListMember=COB01C COBOL ListMember=COB01D	Defaults ListDsn=YOUNG.SAMPLE.PLI01LST(*), LoadMod=PLI01, FromObjDsn=YOUNG.SAMPLE.OBJ, ToObjDsn=YOUNG.SAMPLE.ZAPOBJ PL/I ListMember=PLI01A PL/I ListMember=PLI01B PL/I ListMember=PLI01C PL/I ListMember=PLI01D
C01	ASM01
Defaults ListDsn=YOUNG.SAMPLE.CLST(*), LoadMod=C01, FromObjDsn=YOUNG.SAMPLE.OBJ, ToObjDsn=YOUNG.SAMPLE.ZAPOBJ C ListMember=C01A C ListMember=C01B C ListMember=C01C C ListMember=C01D	Defaults ListDsn=YOUNG.SAMPLE.ASMLST(*), LoadMod=ASM01, FromObjDsn=YOUNG.SAMPLE.OBJ, ToObjDsn=YOUNG.SAMPLE.ZAPOBJ ASM ListMember=ASM01A ASM ListMember=ASM01B ASM ListMember=ASM01C ASM ListMember=ASM01D

To edit the Coverage Utility control file for the selected summary, do the following steps. Use the program name and language appropriate for the sample that you are running; the COBOL sample information is shown here.

1. Select option 1 from the Debug Tool Coverage Utility panel.

The Work with the Control File panel is displayed.

2. Specify the following information:

Use Program Name for File Name: YES

Program Name: COB01

Listing Type: COBOL

An ISPF edit session for the Coverage Utility control file that you request is displayed.

The data in the control file consists of the following information:

- The type of listing file (COBOL)
- The names of the listing files for the programs that you want to test
- The names of the load modules that contain the code of each listing
- The copy to or from information for making copies of the object modules into which the breakpoints are inserted

If the control file that you request did not previously exist, it is created from a sample template that contains comments. The comments help you enter the appropriate information in the fields.

3. Verify the accuracy of the listing data set names and the object module names for copying to or from. Typically these are the same as the names that are shown in the above figure, except that your TSO prefix is used as the high-level qualifier for each data set.
4. Press the End key (PF3) to terminate the edit session.

Related tasks

Chapter 5, "Describing the compile units to be analyzed," on page 49

Producing a sample summary

To produce a summary for a sample program, do the following steps:

1. "Creating sample setup JCL"
2. "Creating sample JCL to start a monitor session" on page 23
3. "Creating JCL for a sample summary report" on page 23
4. "Editing sample JCL to link and run" on page 24
5. "Running the summary sample JCL" on page 24
"Example: Summary report for COB01" on page 26
"Example: Summary report for PLI01" on page 27
"Example: Summary report for C01" on page 28
"Example: Summary report for ASM01" on page 29

Related tasks

"Running the summary sample JCL" on page 24

"Starting the Coverage Utility ISPF dialog" on page 9

Related references

"Compiler options required by Coverage Utility" on page 65

Creating sample setup JCL

Before the sample program can be monitored, Coverage Utility must insert breakpoints into the test program. Coverage Utility does this by using the setup JCL.

When you run the setup JCL, the Coverage Utility setup program analyzes the assembler statements in the compiler listings and creates a table that contains breakpoint data (address, op code, and so on). Breakpoints are inserted in the instrumented object modules or load modules. If you instrumented object modules, you then link these modified object modules into a modified sample load module for Coverage Utility to use.

To create the setup JCL:

1. Select option 2 from the Debug Tool Coverage Utility panel.

The Create JCL for Setup panel is displayed.

All of the default values on this panel are correct for the sample. The defaults are usually correct for your test coverage run. The only field that you might need to change is the **Program Name** field.

2. If necessary, change the program name to the sample name.
3. Select option 1.

Informational messages are written to your screen as the JCL is created. The created JCL is put into the JCL library that is identified on the panel by the member name *Ssample*, where *sample* is the sample name, such as SCOB01.

4. Press the End key (PF3) to exit the panel.

Related tasks

Chapter 6, "Preparing to monitor a program," on page 61

Creating sample JCL to start a monitor session

JCL is required to start a Coverage Utility monitor session.

To create the JCL to start a monitor session, do these steps:

1. Select option 3 from the Debug Tool Coverage Utility panel.

The Create JCL to Start the Monitor panel is displayed.

2. If necessary, change the program name to the sample name.
3. Select option 1.

Informational messages are written to your screen as the JCL is created. The created JCL is put into the JCL library that is identified on the panel by the member name *Xsample*, where *sample* is the name of the sample, such as XCOB01.

4. Press the End key (PF3) to exit the panel.

Use the start monitor JCL to start a monitor session before you run your test case program. You can perform Coverage Utility execution on a system other than the one on which you have stored the listing.

Related tasks

Chapter 7, "Monitoring a program," on page 77

Creating JCL for a sample summary report

JCL is required to generate a summary report.

To create the summary report JCL:

1. Select option 4 from the Debug Tool Coverage Utility panel.

The Create Reports panel is displayed.

2. Select option 1.

The Create JCL for Summary Report panel is displayed. You create the JCL for generating the sample summary report from this panel.

3. If necessary, change Program Name to the sample name.
4. Select option 1.
Informational messages are written to your screen as the JCL is created. The created JCL is put into the JCL library that is identified on the panel by the member name *Tsample*, where *sample* is the sample name, such as TCOB01.
5. Press the End key (PF3) to exit the panel.

Related tasks

“Creating summary report JCL by using the panels” on page 106

Related references

Chapter 10, “Summary report,” on page 113

Editing sample JCL to link and run

You must link the modified object modules (modified by the setup step) into an executable program for testing. Edit the sample link-edit JCL that you copied from *hi_lev_qual.SEQASAMP* so that it will run on your system. Use your local library names and your sample data set names.

Edit the sample GO JCL that you copied from *hi_lev_qual.SEQASAMP* so that it will run on your system. Use your local library names and your sample data set names.

Running the summary sample JCL

When you have created all of the sample JCL, you can run the summary sample by running the following functions in the order listed, where *sample* is the sample name, such as COB01 or PLI01:

Function name	Purpose	Expected results
Compile ²	Performs the compile step	All JCL steps complete with condition code 0.
<i>Ssample</i> ¹	Performs the setup step	All JCL steps complete with condition code 0.
<i>Lsample</i> ²	Links the object modules	The object modules that were modified with breakpoints in the setup step are linked into the <i>sample</i> load module.
<i>Xsample</i> ¹	Starts the monitor	All JCL steps complete with condition code 0.
<i>Gsample</i> ²	Runs the sample program	The sample runs to completion with condition code 0.
EQACUOSA ³ (optional)	Displays statistics	A nonzero EVNTS count is in the TOTALS line
EQACUOSP ³	Stops the monitor session	Coverage Utility writes the statistics to disk
<i>Tsample</i> ¹	Creates the summary of the sample	The summary is in data set <i>proj_qual.sample.SUMMARY</i>

Function name	Purpose	Expected results
		<ol style="list-style-type: none"> 1. JCL is created from the panels and put into the JCL library. 2. JCL is supplied with the installation materials in <i>hlq.SEQASAMP</i>. 3. From either the Control the Monitor panel or the TSO command processor (ISPF option 6) enter the following command, where <i>dtucmd</i> is a monitor command such as EQACUOSA or EQACUOSP: EX '<i>hlq.SEQAEXEC(dtucmd)</i>'

Related tasks

“Using the supplied samples” on page 17

Example: Summary report for COB01

This is the summary report that is produced when you use the COBOL sample program COB01.

```
***** DTCU SUMMARY:                PROGRAM AREA DATA                *****
      DATE: 07/01/2002
      TIME: 10:07:13
      TEST CASE ID:
|<--|          PROGRAM IDENTIFICATION          |-->|
| PA LOAD MOD PROCEDURE | LISTING NAME | STATEMENTS: | BRANCHES: |
|-----|-----|-----|-----|
|          |          | TOTAL  EXEC  % | CPATH  TAKEN  % |
|-----|-----|-----|-----|
1 COB01  PROG      YOUNG.SAMPLE.COBOLST(COB01A)  6    6 100.0
2          PROGA          5    4  80.0
3          PROCA          1    0   0.0
4          LOOP1         3    3 100.0
5          LOOP2         2    2 100.0
6 COB01  PROGB      YOUNG.SAMPLE.COBOLST(COB01B)  6    5  83.3
7          PROCB          1    1 100.0
8          LOOP1         3    3 100.0
9 COB01  PROGC      YOUNG.SAMPLE.COBOLST(COB01C)  5    5 100.0
10         PROCC          3    2  66.7
11         LOOP1         4    3  75.0
12         LOOP2         2    2 100.0
13 COB01  PROGD      YOUNG.SAMPLE.COBOLST(COB01D)  4    0   0.0
14         PROCD          1    0   0.0
15         LOOP1         1    0   0.0
|-----|-----|-----|-----|
Summary for all PAs:                47    36 76.6    0    0 100.0
```

```
***** DTCU SUMMARY:                UNEXECUTED CODE                *****
      DATE: 07/01/2002
      TIME: 10:07:13
      TEST CASE ID:
|<--|          PROGRAM IDENTIFICATION          |-->|
| PA LOAD MOD PROCEDURE | LISTING NAME | start  end  | start  end  | start  end  |
|-----|-----|-----|-----|-----|
2 COB01  PROGA          YOUNG.SAMPLE.COBOLST(COB01A)  58   58
3          PROCA          68   68
6 COB01  PROGB          YOUNG.SAMPLE.COBOLST(COB01B)  40   40
10 COB01  PROCC          YOUNG.SAMPLE.COBOLST(COB01C)  46   46
11         LOOP1         55   55
13 COB01  PROGD          YOUNG.SAMPLE.COBOLST(COB01D)  32   37
14         PROCD          41   41
15         LOOP1         45   45
|-----|-----|-----|-----|
```

```
***** DTCU SUMMARY:                BRANCHES THAT HAVE NOT GONE BOTH WAYS *****
      DATE: 07/01/2002
      TIME: 10:07:13
      TEST CASE ID:
|<--|          PROGRAM IDENTIFICATION          |-->|
| PA LOAD MOD PROCEDURE | LISTING NAME | stmt  stmt  | stmt  stmt  | stmt  |
|-----|-----|-----|-----|-----|
```

Related references

Chapter 10, "Summary report," on page 113

Example: Summary report for PLI01

This is the summary report that is produced when you use the PL/I sample program PLI01.

```
***** DTCU SUMMARY:                PROGRAM AREA DATA                *****
      DATE: 07/01/2002
      TIME: 10:08:38
      TEST CASE ID:
<--  PROGRAM IDENTIFICATION  -->
| PA LOAD MOD PROCEDURE | LISTING NAME | STATEMENTS: | BRANCHES: |
|                       |              | TOTAL   EXEC  % | CPATH  TAKEN  % |
-----|-----|-----|-----|-----|-----|
  1 PLI01  PLI01A      YOUNG.SAMPLE.PLILST(PLI01A)   9    9 100.0
  2                PROC2A                2    0   0.0
  3 PLI01  PLI01B      YOUNG.SAMPLE.PLILST(PLI01B)  11    8  72.7
  4                PROC1                2    2 100.0
  5 PLI01  PLI01C      YOUNG.SAMPLE.PLILST(PLI01C)   7    4  57.1
  6                PROC1                4    3  75.0
  7 PLI01  PLI01D      YOUNG.SAMPLE.PLILST(PLI01D)   2    0   0.0
  8                PROC1                4    0   0.0
-----|-----|-----|-----|-----|
Summary for all PAs:                41    26  63.4    0    0 100.0
```

```
***** DTCU SUMMARY:                UNEXECUTED CODE                *****
      DATE: 07/01/2002
      TIME: 10:08:38
      TEST CASE ID:
<--  PROGRAM IDENTIFICATION  -->
| PA LOAD MOD PROCEDURE | LISTING NAME | start  end  start  end  start  end |
-----|-----|-----|-----|-----|-----|
  2 PLI01  PROC2A      YOUNG.SAMPLE.PLILST(PLI01A)   17   18
  3 PLI01  PLI01B      YOUNG.SAMPLE.PLILST(PLI01B)    9    9    15   16
  5 PLI01  PLI01C      YOUNG.SAMPLE.PLILST(PLI01C)    6    6    10   11
  6                PROC1                16   16
  7 PLI01  PLI01D      YOUNG.SAMPLE.PLILST(PLI01D)    3    11
  8                PROC1                6    10
```

```
***** DTCU SUMMARY:                BRANCHES THAT HAVE NOT GONE BOTH WAYS *****
      DATE: 07/01/2002
      TIME: 10:08:38
      TEST CASE ID:
<--  PROGRAM IDENTIFICATION  -->
| PA LOAD MOD PROCEDURE | LISTING NAME | stmt  stmt  stmt  stmt  stmt |
-----|-----|-----|-----|-----|
```

Example: Summary report for C01

This is the summary report that is produced when you use the C sample program C01.

```

***** DTCU SUMMARY:                PROGRAM AREA DATA                *****
      DATE: 07/01/2002
      TIME: 10:09:17
      TEST CASE ID:
|<--          PROGRAM IDENTIFICATION          |-->
| PA LOAD MOD PROCEDURE | LISTING NAME | STATEMENTS: | BRANCHES: |
|                       |              | TOTAL   EXEC % | CPATH  TAKEN % |
-----
  1 C01   PROCA   YOUNG.SAMPLE.CLST(C01A)   2     0  0.0
  2         main   |              | 10     9  90.0
  3 C01   PROCB   YOUNG.SAMPLE.CLST(C01B)   2     2 100.0
  4         C01B   |              |  9     8  88.9
  5 C01   PROCC   YOUNG.SAMPLE.CLST(C01C)   4     3  75.0
  6         C01C   |              |  8     6  75.0
  7 C01   PROCD   YOUNG.SAMPLE.CLST(C01D)   3     0  0.0
  8         C01D   |              |  3     0  0.0
-----
Summary for all PAs:                41     28 68.3      0     0 100.0

```

```

***** DTCU SUMMARY:                UNEXECUTED CODE                *****
      DATE: 07/01/2002
      TIME: 10:09:17
      TEST CASE ID:
|<--          PROGRAM IDENTIFICATION          |-->
| PA LOAD MOD PROCEDURE | LISTING NAME | start  end  start  end  start  end
-----
  1 C01   PROCA   YOUNG.SAMPLE.CLST(C01A)   29   31
  2         main   |              | 25   25
  4 C01   C01B   YOUNG.SAMPLE.CLST(C01B)   27   27
  5 C01   PROCC   YOUNG.SAMPLE.CLST(C01C)   34   34
  6         C01C   |              | 21   21   26   26
  7 C01   PROCD   YOUNG.SAMPLE.CLST(C01D)   21   25
  8         C01D   |              | 15   19
-----

```

```

***** DTCU SUMMARY:                BRANCHES THAT HAVE NOT GONE BOTH WAYS *****
      DATE: 07/01/2002
      TIME: 10:09:17
      TEST CASE ID:
|<--          PROGRAM IDENTIFICATION          |-->
| PA LOAD MOD PROCEDURE | LISTING NAME | stmt  stmt  stmt  stmt  stmt
-----

```

Example: Summary report for ASM01

This is the summary report that is produced when you use the assembler sample program ASM01.

```
***** DTCU SUMMARY:                PROGRAM AREA DATA                *****
      DATE: 07/01/2002
      TIME: 10:08:12
      TEST CASE ID:
<--  PROGRAM IDENTIFICATION  -->
| PA LOAD MOD PROCEDURE | LISTING NAME | STATEMENTS: | BRANCHES: |
|-----|-----|-----|-----|
| TOTAL   EXEC   % | CPATH  TAKEN  % |
-----|-----|-----|-----|
1 ASM01  TEST2   YOUNG.SAMPLE.ASMLST(ASM01A)  41   31  75.6
2 ASM01  TEST2B  YOUNG.SAMPLE.ASMLST(ASM01B)  45   39  86.7
3 ASM01  TEST2C  YOUNG.SAMPLE.ASMLST(ASM01C)  39   32  82.1
4 ASM01  TEST2D  YOUNG.SAMPLE.ASMLST(ASM01D)  25    0   0.0
-----|-----|-----|-----|
Summary for all PAs:                150   102  68.0    0    0 100.0
```

```
***** DTCU SUMMARY:                UNEXECUTED CODE                *****
      DATE: 07/01/2002
      TIME: 10:08:12
      TEST CASE ID:
<--  PROGRAM IDENTIFICATION  -->
| PA LOAD MOD PROCEDURE | LISTING NAME | start  end   start  end   start  end
|-----|-----|-----|-----|-----|-----|
1 ASM01  TEST2   YOUNG.SAMPLE.ASMLST(ASM01A) 000056 000062 000086 0000A0
2 ASM01  TEST2B  YOUNG.SAMPLE.ASMLST(ASM01B) 000050 00005A 000078 000082
3 ASM01  TEST2C  YOUNG.SAMPLE.ASMLST(ASM01C) 000034 00003A 000050 00005A 000092 000098
4 ASM01  TEST2D  YOUNG.SAMPLE.ASMLST(ASM01D) 000000 000000 000016 00006C
-----|-----|-----|-----|-----|-----|
```

```
***** DTCU SUMMARY:                BRANCHES THAT HAVE NOT GONE BOTH WAYS *****
      DATE: 07/01/2002
      TIME: 10:08:12
      TEST CASE ID:
<--  PROGRAM IDENTIFICATION  -->
| PA LOAD MOD PROCEDURE | LISTING NAME | stmt  stmt  stmt  stmt  stmt
|-----|-----|-----|-----|-----|
-----|-----|-----|-----|-----|
```

Producing a sample annotated listing report

If you want more information on some or all of the modules that you tested, you can create an annotated listing report of the sample listing. This listing contains information about each breakpoint. To the right of each statement number, one of the following symbols is shown to indicate the results of the execution of that statement:

- : Instruction has executed.
- ⌞ Instruction has not executed.

In addition to the above annotation symbols used for COBOL, PL/I and C, the following symbols are used for assembler:

- @ Data area
- % Unconditional branch that has been executed

To produce an annotated listing of a sample, do the following steps:

1. "Creating sample setup JCL" on page 22
2. "Creating sample JCL to start a monitor session" on page 23
3. "Creating JCL for an annotated listing report" on page 30.

4. "Editing sample JCL to link and run" on page 24
 5. "Running the annotated sample JCL."
 - "Example: COBOL annotated listing report" on page 32
 - "Example: PL/I annotated listing report" on page 34
 - "Example: C/C++ annotated listing report" on page 35
 - "Example: Assembler annotated listing report" on page 37
- Related tasks**
- "Producing a sample summary" on page 22
 - "Starting the Coverage Utility ISPF dialog" on page 9
 - "Running the annotated sample JCL"
- Related references**
- "Compiler options required by Coverage Utility" on page 65

Creating JCL for an annotated listing report

To create the annotated listing report JCL, do the following steps:

1. Select option 4 from the Debug Tool Coverage Utility panel.
The Create Reports panel is displayed.
2. Select option 2.
The Create JCL for Summary and Annotation Report panel is displayed. You create the JCL for printing the annotated listing report (along with a summary) for the sample from this panel.
3. If necessary, change the program name to the sample name.
4. Select option 1.
Informational messages are written to your screen as the JCL is created. The created JCL is put into the JCL library that is identified on the panel by the member name *Rsample*, where *sample* is the name of the sample, such as RCOB01.
5. Press the End key (PF3) to exit the panel.

Related tasks

"Creating annotated listing report JCL by using the panels" on page 107

Related references

Chapter 11, "Annotated listing report," on page 123

Running the annotated sample JCL

When you have created all of the sample JCL, you can run the sample by running the following JCL in the order listed in the table below, where *sample* is the sample name, such as COB01 or PLI01. If you have run the summary sample JCL, then you only need to run *Rsample*.

Table 2.

Function name	Purpose	Expected results
Compile ²	Performs the compile step	All JCL steps complete with condition code 0.
Ssample ¹	Performs the setup step	All JCL steps complete with condition code 0.
Lsample ²	Links the object modules	The object modules that were modified with breakpoints in the setup step are linked into the sample load module.
Xsample ¹	Starts the monitor	All JCL steps complete with condition code 0.
Gsample ²	Runs the sample program	The sample runs to completion with condition code 0.

Table 2. (continued)

Function name	Purpose	Expected results
EQACUOSA ³ (optional)	Displays statistics	A nonzero EVNTS count in the TOTALS line.
EQACUOSP ³	Stops the monitor session	Coverage Utility writes the statistics to disk.
Rsample ¹	Creates the annotated listing report (and summary report) of the sample	The report is in data set proj_qual.sample.REPORT.
<p>1. The JCL is created from the panels and put into the JCL library.</p> <p>2. The JCL is supplied with the installation materials in <i>hlq.SEQASAMP</i>.</p> <p>3. From either the Control the Monitor panel or the TSO command processor (ISPF option 6), enter the following command, where <i>dtcucmd</i> is the command such as EQACUOSA or EQACUOSP:</p> <p>EX '<i>hlq.SEQAEEXEC(dtucmd)</i>'</p>		

Related tasks

“Using the supplied samples” on page 17

Example: COBOL annotated listing report

This is the annotated listing report that is produced when you use the COBOL sample program COB01.

```

LineID  PL SL  ----+*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7-|
000001          * COB01A - COBOL EXAMPLE FOR DTCU
000002
000003          IDENTIFICATION DIVISION.
000004          PROGRAM-ID. COB01A.
000005          *****
000006          * Licensed Materials - Property of IBM          *
000007          *          *
000008          * 5655-M18: Debug Tool for z/OS          *
000009          * 5655-M19: Debug Tool Utilities and Advanced Functions for z/OS *
000010          * (C) Copyright IBM Corp. 1997, 2004 All Rights Reserved *
000011          *          *
000012          * US Government Users Restricted Rights - Use, duplication or *
000013          * disclosure restricted by GSA ADP Schedule Contract with IBM *
000014          * Corp.          *
000015          *          *
000016          *****
000017
000018          ENVIRONMENT DIVISION.
000019
000020          DATA DIVISION.
000021
000022          WORKING-STORAGE SECTION.
000023          01 TAPARM1      PIC 99 VALUE 5.
000024          01 TAPARM2      PIC 99 VALUE 2.
000025          01 COB01B       PIC X(6) VALUE 'COB01B'.
000026          01 P1PARM1     PIC 99 VALUE 0.
000027
000028          01 TASTRUCT.
000029             05 LOC-ID.
000030                 10 STATE    PIC X(2).
000031                 10 CITY     PIC X(3).
000032                 05 OP-SYS   PIC X(3).
000033
000034          PROCEDURE DIVISION.
000035
000036          * THE FOLLOWING ALWAYS PERFORMED
000037
000038          PROG.
000039          * ACCESS BY TOP LEVEL QUALIFIER
000040             MOVE 'ILCHIMVS' TO TASTRUCT
000041 :
000042
000043          * ACCESS BY MID LEVEL QUALIFIERS
000044 :             MOVE 'ILSPR' TO LOC-ID
000045 :             MOVE 'AIX' TO OP-SYS
000046
000047          * ACCESS BY LOW LEVEL QUALIFIERS
000048 :             MOVE 'KY' TO STATE
000049 :             MOVE 'LEX' TO CITY
000050 :             MOVE 'VM ' TO OP-SYS
000051             .
000052
000053          PROGA.
000054 :             PERFORM LOOP1 UNTIL TAPARM1 = 0
000055
000056 :             IF TAPARM2 = 0 THEN
000057          *          PROCA NOT EXECUTED
000058 ^ 1             PERFORM PROCA.
000059

```

```
000060
000061 :           PERFORM LOOP2 UNTIL TAPARM2 = 0
000062           .
000063 :           STOP RUN
000064           .
000065
000066           PROCA.
000067 *           PROCA NOT EXECUTED
000068 ^           MOVE 10 TO P1PARAM1
000069           .
000070           LOOP1.
000071 :           IF TAPARM1 > 0 THEN
000072 :   1           SUBTRACT 1 FROM TAPARM1.
000073 :           CALL 'COB01B'
000074           .
000075           LOOP2.
000076 :           IF TAPARM2 > 0 THEN
000077 :   1           SUBTRACT 1 FROM TAPARM2.
000078
```

Example: PL/I annotated listing report

This is the annotated listing report that is produced when you use the PL/I sample program PLI01.

```

STMT
1  PLI01A:PROC OPTIONS(MAIN);          /* PL/I DTCU TESTCASE          */
  /*****
  /* Licensed Materials - Property of IBM          */
  /*
  /* 5655-M18: Debug Tool for z/OS                */
  /* 5655-M19: Debug Tool Utilities and Advanced Functions for z/OS */
  /* (C) Copyright IBM Corp. 1997, 2004 All Rights Reserved      */
  /*
  /* US Government Users Restricted Rights - Use, duplication or  */
  /* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*/
  /*
  /*****

2  DCL EXPARM1 FIXED BIN(31) INIT(5);
3  DCL EXPARM2 FIXED BIN(31) INIT(2);
4  DCL PARM2  FIXED BIN(31) INIT(2);
5  DCL PLI01B EXTERNAL ENTRY;          /*                               */
6: DO WHILE (EXPARM1 > 0);             /* THIS DO LOOP EXECUTED 5 TIMES*/
7:   EXPARM1 = EXPARM1 -1;             /*                               */
8:   CALL PLI01B(PARM2);              /* PLI01B CALLED 5 TIMES        */
9: END;
10: IF (EXPARM2 = 0) THEN              /* THIS BRANCH ALWAYS TAKEN     */
    CALL PROC2A(EXPARM2);             /* PROC2A NEVER CALLED         */
11: DO WHILE (EXPARM2 > 0);           /* DO LOOP EXECUTED TWICE      */
12:   EXPARM2 = EXPARM2 - 1;
13: END;
14: RETURN;

15  PROC2A: PROCEDURE(P1PARM1);        /* THIS PROCEDURE NEVER EXECUTED */
16  DCL P1PARM1 FIXED BIN(31);
17^ P1PARM1 = 10;
18^ END PROC2A;
19  END PLI01A;

```

Example: C/C++ annotated listing report

This is the annotated listing report that is produced when you use the C/C++ sample program C01.

* * * * * S O U R C E * * * * *

LINE	STMT	SEQNBR	INCNO
	...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8...+...9...+...		
1	main()	1	
2	/******	2	
3	/* Licensed Materials - Property of IBM */	3	
4	/* */	4	
5	/* 5655-M18: Debug Tool for z/OS */	5	
6	/* 5655-M19: Debug Tool Utilities and Advanced Functions for z/OS */	6	
7	/* (C) Copyright IBM Corp. 1997, 2004 All Rights Reserved */	7	
8	/* */	8	
9	/* US Government Users Restricted Rights - Use, duplication or */	9	
10	/* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*/	10	
11	/* */	11	
12	/******	12	
13		13	
14	{	14	
15:	1 int EXPARM1 = 5;	15	
16:	2 int EXPARM2 = 2;	16	
17	extern void C01B(void);	17	
18	void PROCA(int); /* function not called */	18	
19:	3 while (EXPARM1 > 0) /* execute loop 5 times */	19	
20	{	20	
21:	4 EXPARM1 = EXPARM1 -1;	21	
22:	5 C01B(); /* call C01B 5 times */	22	
23	}	23	
24:	6 if (EXPARM2 == 0) /* branch taken */	24	
25~	7 PROCA(EXPARM2); /* not executed */	25	
26:	8 while (EXPARM2 > 0) /* loop execute 2 times */	26	
27:	9 EXPARM2 = EXPARM2 - 1; /* executed twice */	27	
28:	}	28	
29~	void PROCA(int P1PARM1) /* function not called */	29	
30	{	30	
31~	10 P1PARM1 = 10; /* not executed */	31	
32	}	32	

Example: Assembler annotated listing report

This is the annotated listing report that is produced when you use the Assembler sample program ASM01.

Active Usings: None						HLASM R4.0 2002/07/01 07.20	
Loc	Object Code	Addr1	Addr2	Stmt	Source Statement		
				1	*****		
				2	* Licensed Materials - Property of IBM		*
				3	*		*
				4	* 5655-M18: Debug Tool for z/OS		*
				5	* 5655-M19: Debug Tool Utilities and Advanced Functions for z/OS		*
				6	* (C) Copyright IBM Corp. 1997, 2004 All Rights Reserved		*
				7	*		*
				8	* US Government Users Restricted Rights - Use, duplication or		*
				9	* disclosure restricted by GSA ADP Schedule Contract with IBM		*
				10	* Corp.		*
				11	*		*
				12	*****		
				13	*		
				14	*****		
				15	*		*
				16	* DTCU ASSEMBLER TESTCASE.		*
				17	*		*
				18	*****		
000000		00000	00110	19	TEST2 CSECT ,		01S0001
000000				20	@MAINT DS 0H		01S0001
		R:F	00000	21	USING *,@15		01S0001
000000	47F0 F016		00016	22%	B @PROLOG		01S0001
000004	10			23@	DC AL1(16)		01S0001
000005	E3C5E2E3F2404040			24@	DC C'TEST2 97.295'		01S0001
				25	DROP @15		
000015	00						
000016	90EC D00C		0000C	26:@PROLOG	STM @14,@12,12(@13)		01S0001
00001A	18CF			27:	LR @12,@15		01S0001
			00000	28 @PSTART	EQU TEST2		01S0001
		R:C	00000	29	USING @PSTART,@12		01S0001
00001C	50D0 C0B0		000B0	30:	ST @13,@SA00001+4		01S0001
000020	41E0 C0AC		000AC	31:	LA @14,@SA00001		01S0001
000024	50E0 D008		00008	32:	ST @14,8(,@13)		01S0001
000028	18DE			33:	LR @13,@14		01S0001
				34 *	DO WHILE(EXPARG1>0); /* THIS DO LOOP EXECUTED 5 TIMES */		*/
00002A	47F0 C042		00042	35%	B @DE00006		01S0006
00002E				36 @DL00006	DS 0H		01S0007
				37 *	EXPARG1 = EXPARG1 - 1; /*		*/
00002E	5810 C100		00100	38:	L @01,EXPARG1		01S0007
000032	0610			39:	BCTR @01,0		01S0007
000034	5010 C100		00100	40:	ST @01,EXPARG1		01S0007
				41 *	CALL TEST2B(PARG2); /* TEST2B CALLED 5 TIMES		*/
000038	58F0 C0F8		000F8	42:	L @15,@CV00063		01S0008
00003C	4110 C0A4		000A4	43:	LA @01,@AL00002		01S0008
000040	05EF			44%	BALR @14,@15		01S0008
				45 *	END;		
000042	5800 C100		00100	46:@DE00006	L @00,EXPARG1		01S0009
000046	1200			47:	LTR @00,@00		01S0009
000048	4720 C02E		0002E	48:	BP @DL00006		01S0009
				49 *	IF (EXPARG2 = 0) THEN /* THIS BRANCH ALWAYS TAKEN		*/
00004C	5810 C104		00104	50:	L @01,EXPARG2		01S0010
000050	1211			51:	LTR @01,@01		01S0010
000052	4770 C06C		0006C	52:	BNZ @RF00010		01S0010
				53 *	CALL PROC1(EXPARG2); /* PROC1 NEVER CALLED		*/
000056	4110 C0A8		000A8	54^	LA @01,@AL00003		01S0011
00005A	45E0 C086		00086	55^	BAL @14,PROC1		01S0011
				56 *	DO WHILE(EXPARG2>0); /* DO LOOP EXECUTED TWICE		*/
00005E	47F0 C06C		0006C	57^	B @DE00012		01S0012

000062		58 @DL00012 DS	0H	01S0013
		59 *	EXPARM2 = EXPARM2 - 1;	01S0013
000062 5820 C104	00104	60:	L @02,EXPARM2	01S0013
000066 0620		61:	BCTR @02,0	01S0013
000068 5020 C104	00104	62:	ST @02,EXPARM2	01S0013
		63 *	END;	01S0014
00006C 5830 C104	00104	64:@DE00012 L	@03,EXPARM2	01S0014
000070 1233		65:	LTR @03,@03	01S0014
000072 4720 C062	00062	66:	BP @DL00012	01S0014
		67 *	RETURN CODE(0);	01S0015
000076 1FFF		68:	SLR @15,@15	01S0015
000078 58D0 D004	00004	69:	L @13,4(,@13)	01S0015
00007C 58E0 D00C	0000C	70:	L @14,12(,@13)	01S0015
000080 980C D014	00014	71:	LM @00,@12,20(@13)	01S0015
000084 07FE		72%	BR @14	01S0015
		73 *	END TEST2;	01S0020
		74 *	PROC1:	01S0016
		75 *	PROCEDURE(P1PARAM1);	/* THIS PROCEDURE NEVER EXECUTED */
000086 90EC D00C	0000C	76^PROC1	STM @14,@12,12(@13)	01S0016
00008A D203 C0F4 1000 000F4 00000		77^	MVC @PC00002(4),0(@01)	01S0016
		78 *	P1PARAM1 = 10;	01S0018
000090 5820 C0F4	000F4	79^	L @02,@PA00064	01S0018
000094 4130 000A	0000A	80^	LA @03,10	01S0018
000098 5030 2000	00000	81^	ST @03,P1PARAM1(,@02)	01S0018
		82 *	END PROC1;	01S0019
00009C		83 @EL00002 DS	0H	01S0019
00009C		84 @EF00002 DS	0H	01S0019
00009C 98EC D00C	0000C	85^@ER00002 LM	@14,@12,12(@13)	01S0019
0000A0 07FE		86^	BR @14	01S0019
0000A2		87 @DATA DS	0H	
0000A4		88 DS	0F	
0000A4		89 @AL00002 DS	0A	
0000A4 00000108		90@	DC A(PARM2)	
0000A8		91 @AL00003 DS	0A	
0000A8 00000104		92@	DC A(EXPARM2)	
0000AC		93 DS	0F	
0000AC		94@@SA00001 DS	18F	
0000F4		95@PC00002 DS	1F	
0000F8		96 DS	0F	
0000F8 00000000		97@@CV00063 DC	V(TEST2B)	
000100		98	LTORG	
000100		99 DS	0D	
000100 00000005		100@EXPARM1 DC	F'5'	
000104 00000002		101@EXPARM2 DC	F'2'	
000108 00000002		102@PARM2 DC	F'2'	
	00000	103 @DYN SIZE EQU	0	
	00000	104 @00 EQU	0	
	00001	105 @01 EQU	1	
	00002	106 @02 EQU	2	
	00003	107 @03 EQU	3	
	00004	108 @04 EQU	4	
	00005	109 @05 EQU	5	
	00006	110 @06 EQU	6	
	00007	111 @07 EQU	7	
	00008	112 @08 EQU	8	
	00009	113 @09 EQU	9	
	0000A	114 @10 EQU	10	
	0000B	115 @11 EQU	11	
	0000C	116 @12 EQU	12	
	0000D	117 @13 EQU	13	
	0000E	118 @14 EQU	14	
	0000F	119 @15 EQU	15	
	00000 00004	120 P1PARAM1 EQU	0,4,C'F'	
	000F4 00004	121 @PA00064 EQU	@PC00002,4,C'F'	
	0006C	122 @RF00010 EQU	@DE00012	
000110		123 DS	0D	
	00110	124 @ENDDATA EQU	*	
	00110	125 @MODLEN EQU	@ENDDATA-TEST2	
		126	END	,

Chapter 4. Samples that are provided with Coverage Utility

This section describes the samples that are shipped with Debug Tool and the data sets that you need to allocate for each sample that you run.

“User data sets that are required to run the samples”

“COBOL samples” on page 43

“PL/I samples” on page 43

“C/C++ samples” on page 44

“Assembler samples” on page 44

User data sets that are required to run the samples

Before you run the samples, allocate your personal sample data sets as shown in this table:

Data set name	LRECL	BLKSIZE	RECFM	DSORG
Common data sets: The following data sets are required to run any of the samples.				
prefix.SAMPLE.DTCU	255	*	VB	PO
prefix.SAMPLE.JCL	80	*	FB	PO
prefix.SAMPLE.OBJ	80	3200	FB	PO
prefix.SAMPLE.RUNLIB	0	*	U	PO
prefix.SAMPLE.ZAPOBJ	80	3200	FB	PO
COBOL data sets: The following data sets are required to run the COBOL samples.				
prefix.SAMPLE.COBOLE	80	*	FB	PO
prefix.SAMPLE.COBOLEST	133	*	FBA	PO
	121 for OS/VS COBOL	12100	FBA	PO
PL/I data sets: The following data sets are required to run the PL/I samples.				
prefix.SAMPLE.PLI	80	*	FB	PO
prefix.SAMPLE.PLILST	137 for Enterprise PL/I for z/OS and OS/390 Version 3 Release 2 and above	*	VBA	PO
	132 for VisualAge PL/I and Enterprise PL/I for z/OS and OS/390 Version 3 Release 1	*	VBA	PO
	125 for OS PL/I or PL/I for MVS & VM	*	VBA	PO
C/C++ data sets: The following data sets are required to run the C/C++ samples.				
prefix.SAMPLE.C	256	*	VB	PO
prefix.SAMPLE.CLST	137	*	VBA	PO
Assembler data sets: The following data sets are required to run the assembler samples.				
prefix.SAMPLE.ASM	80	*	FB	PO

Data set name	LRECL	BLKSIZE	RECFM	DSORG
prefix.SAMPLE.ASMLST	133 for High Level Assembler	*	FBA/M	PO
	137 for High Level Assembler	*	VBA/M	PO
	121 for Assembler H	*	FBA	PO
Note: A value of * in the BLKSIZE column indicates that you can use any valid block size.				

COBOL samples

This table lists the members of the sample library SEQASAMP for the COBOL sample. Before you run the sample, allocate the personal sample data sets that are shown in the table and copy the members that are appropriate for your system.

SEQASAMP member name	Your sample data set	Description of member
EQACU1C	prefix.SAMPLE.DTCU(COB01)	Coverage Utility control cards
EQACU1GM	prefix.SAMPLE.JCL(GCOB01)	Go JCL, COBOL for MVS & VM
EQACU1GO	prefix.SAMPLE.JCL(GCOB01)	Go JCL, OS/VS COBOL
EQACU1GZ	prefix.SAMPLE.JCL(GCOB01)	Go JCL, Enterprise COBOL for z/OS and OS/390
EQACU1G2	prefix.SAMPLE.JCL(GCOB01)	Go JCL, VS COBOL II
EQACU1G9	prefix.SAMPLE.JCL(GCOB01)	Go JCL, COBOL for OS/390 & VM
EQACU1LM	prefix.SAMPLE.JCL(LCOB01)	Link JCL COBOL for MVS & VM
EQACU1LO	prefix.SAMPLE.JCL(LCOB01)	Link JCL, OS/VS COBOL
EQACU1LZ	prefix.SAMPLE.JCL(LCOB01)	Link JCL, Enterprise COBOL for z/OS and OS/390
EQACU1L2	prefix.SAMPLE.JCL(LCOB01)	Link JCL, VS COBOL II
EQACU1L9	prefix.SAMPLE.JCL(LCOB01)	Link JCL, COBOL for OS/390 & VM
EQACU1MM	prefix.SAMPLE.JCL(COMPCOB)	Compile JCL, COBOL for MVS & VM
EQACU1MO	prefix.SAMPLE.JCL(COMPCOB)	Compile JCL, OS/VS COBOL
EQACU1MZ	prefix.SAMPLE.JCL(COMPCOB)	Compile JCL, Enterprise COBOL for z/OS and OS/390
EQACU1M2	prefix.SAMPLE.JCL(COMPCOB)	Compile JCL, VS COBOL II
EQACU1M9	prefix.SAMPLE.JCL(COMPCOB)	Compile JCL, COBOL for OS/390 & VM
EQACU1SA	prefix.SAMPLE.COBOL(COB01A)	COBOL source code
EQACU1SB	prefix.SAMPLE.COBOL(COB01B)	
EQACU1SC	prefix.SAMPLE.COBOL(COB01C)	
EQACU1SD	prefix.SAMPLE.COBOL(COB01D)	

PL/I samples

This table lists the members of the sample library SEQASAMP for the PL/I sample. Before you run the sample, allocate the personal sample data sets that are shown in the table and copy the members that are appropriate for your system.

SEQASAMP member name	Your sample data set	Description of member
EQACU2C	prefix.SAMPLE.DTCU(PLI01)	Coverage Utility control cards
EQACU2GV	prefix.SAMPLE.JCL(GPLI01)	Go JCL, VisualAge PL/I
EQACU2GM	prefix.SAMPLE.JCL(GPLI01)	Go JCL, PL/I for MVS & VM
EQACU2GZ	prefix.SAMPLE.JCL(GPLI01)	Go JCL, Enterprise PL/I for z/OS and OS/390
EQACU2G1	prefix.SAMPLE.JCL(GPLI01)	Go JCL, PL/I 1.5.1
EQACU2G2	prefix.SAMPLE.JCL(GPLI01)	Go JCL, PL/I 2.3.0
EQACU2LV	prefix.SAMPLE.JCL(LPLI01)	Link JCL, VisualAge PL/I

SEQASAMP member name	Your sample data set	Description of member
EQACU2LM	prefix.SAMPLE.JCL(LPLI01)	Link JCL, PL/I for MVS & VM
EQACU2LZ	prefix.SAMPLE.JCL(GPLI01)	Link JCL, Enterprise PL/I for z/OS and OS/390
EQACU2L1	prefix.SAMPLE.JCL(LPLI01)	Link JCL, PL/I 1.5.1
EQACU2L2	prefix.SAMPLE.JCL(LPLI01)	Link JCL, PL/I 2.3.0
EQACU2MV	prefix.SAMPLE.JCL(COMPPLI)	Compile JCL, VisualAge PL/I
EQACU2MM	prefix.SAMPLE.JCL(COMPPLI)	Compile JCL, PL/I for MVS & VM
EQACU2MZ	prefix.SAMPLE.JCL(GPLI01)	Compile JCL, Enterprise PL/I for z/OS and OS/390
EQACU2M1	prefix.SAMPLE.JCL(COMPPLI)	Compile JCL, PL/I 1.5.1
EQACU2M2	prefix.SAMPLE.JCL(COMPPLI)	Compile JCL, PL/I 2.3.0
EQACU2SA	prefix.SAMPLE.PLI(PLI01A)	PL/I source code
EQACU2SB	prefix.SAMPLE.PLI(PLI01B)	
EQACU2SC	prefix.SAMPLE.PLI(PLI01C)	
EQACU2SD	prefix.SAMPLE.PLI(PLI01D)	

C/C++ samples

This table lists the members of the sample library SEQASAMP for the C/C++ sample. Before you run the sample, allocate the personal sample data sets that are shown in the table and copy the members that are appropriate for your system.

SEQASAMP member name	Your sample data set	Description of member
EQACU3C	prefix.SAMPLE.DTCU(C01)	Coverage Utility control cards
EQACU3GA	prefix.SAMPLE.JCL(GC01)	Go JCL, OS/390 C
EQACU3LA	prefix.SAMPLE.JCL(LC01)	Link JCL, OS/390 C
EQACU3MA	prefix.SAMPLE.JCL(COMPC)	Compile JCL, OS/390 C
EQACU3SA	prefix.SAMPLE.C(C01A)	OS/390 C source code
EQACU3SB	prefix.SAMPLE.C(C01B)	
EQACU3SC	prefix.SAMPLE.C(C01C)	
EQACU3SD	prefix.SAMPLE.C(C01D)	

Assembler samples

This table lists the members of the sample library SEQASAMP for the Assembler sample. Before you run the sample, allocate the personal sample data sets that are shown in the table and copy the members that are appropriate for your system.

SEQASAMP member name	Your sample data set	Description of member
EQACU4C	prefix.SAMPLE.DTCU(ASM01)	Coverage Utility control cards
EQACU4GH	prefix.SAMPLE.JCL(GASM01)	Go JCL, Assembler H
EQACU4GL	prefix.SAMPLE.JCL(GASM01)	Go JCL, High Level Assembler

SEQASAMP member name	Your sample data set	Description of member
EQACU4LH	prefix.SAMPLE.JCL(LASM01)	Link JCL, Assembler H
EQACU4LL	prefix.SAMPLE.JCL(LASM01)	Link JCL, High Level Assembler
EQACU4MH	prefix.SAMPLE.JCL(COMPASM)	Compile JCL, Assembler H
EQACU4ML	prefix.SAMPLE.JCL(COMPASM)	Compile JCL, High Level Assembler
EQACU4SA	prefix.SAMPLE.ASM(ASM01A)	Assembler source code
EQACU4SB	prefix.SAMPLE.ASM(ASM01B)	
EQACU4SC	prefix.SAMPLE.ASM(ASM01C)	
EQACU4SD	prefix.SAMPLE.ASM(ASM01D)	

Part 3. Preparing the programs to be monitored

Chapter 5. Describing the compile units to be analyzed	49	Determining when to create or submit setup JCL	65
Editing a control file	49	Compiler options required by Coverage Utility	65
Contents of the control file	50	COBOL compiler options required by Coverage Utility	65
Syntax of control file statements	51	PL/I compiler options required by Coverage Utility	66
INCLUDE statement	52	C/C++ compiler options required by Coverage Utility	67
DEFAULTS statement	52	Assembler options required by Coverage Utility	68
COBOL statement (compilation unit definition)	53	Compiler restrictions imposed by Coverage Utility	68
PL/I statement (compilation unit definition)	55	COBOL compiler restrictions imposed by Coverage Utility	68
C statement (compilation unit definition)	56	Coverage Utility	68
ASM statement (compilation unit definition)	58	PL/I compiler restrictions imposed by Coverage Utility	69
Examples: Control files	60	C/C++ compiler restrictions imposed by Coverage Utility	69
Example: Control file for a single compilation unit	60	Assembler restrictions imposed by Coverage Utility	69
Example: Control file for multiple compilation units	60	Setup JCL for the compile job stream	70
Example: Control file for load module	60	Parameters for the setup programs	70
Chapter 6. Preparing to monitor a program	61	EQACUSET	70
Supplying setup input	61	EQACUZPT	72
Setup processing	61	EQACUZPL	72
Restrictions on setup input	62	EQACUZPP	72
Instrumenting object modules or load modules	62		
Creating the setup JCL by using the panels	63		

This information helps you learn how to use the Coverage Utility control file to describe the compile units that you want to be analyzed. The description and function of each statement in the control file is given, together with examples. In this part, you also learn which compiler options to use when compiling your source programs, how to instrument object or load modules with breakpoints, and how to set up Coverage Utility to monitor your programs.

Chapter 5. Describing the compile units to be analyzed

This section describes the function of the Coverage Utility control file and how you can use it to describe the compile units that you want to be analyzed.

- “Editing a control file”
- “Contents of the control file” on page 50
- “Syntax of control file statements” on page 51
- “Examples: Control files” on page 60

You need to create a separate control file for each group of programs that you want to process.

Editing a control file

To edit the control file, do the following steps:

1. Select option 1 from the Debug Tool Coverage Utility panel to display the Work with the Control File panel, shown here:

```
----- Work with the Control File -----
Option ==>

1 Edit          Edit Control File
2 Reset        Reset Control File from Site Master

Enter END to Terminate

Use Program Name for File Name YES (Yes|No) Program Name COB01

Control File:
Control File Dsn. . . 'YOUNG.SAMPLE.DTCU(COB01)'
Listing Type. . . . . COBOL      (COBOL|PL/I|C|ASM)
```

2. Enter option 1 and change the values for **Control File Dsn** and **Listing Type** as appropriate.

The options and fields for the panel are as follows:

Edit Starts an edit session for the control file that you specify in the **Control File Dsn** field.

Reset Replaces the information in the Control File that you specify in the **Control File Dsn** field with information from the site sample control file.

Use Program Name for File Name

Enter YES if you want to construct the data set names from the default high-level qualifier, the specified program name, and the default low-level qualifier for each data set.

When you press Enter, the file names on the panel are changed automatically. Coverage Utility usually constructs the data set names by using the program name.

Program Name

The name to use for Coverage Utility data sets when you enter YES in

the **Use Program Name for File Name** field. This name can be any valid name; it does not have to be the name of any of your programs.

Names of the following forms are created:

- Sequential data sets:

'proj_qual.program_name.file_type'

For example: 'YOUNG.SAMPLE.COB01.BRKTAB'

- Partitioned data sets:

'proj_qual.file_type(program_name)'

For example: 'YOUNG.SAMPLE.BRKTAB(COB01)'

Control file Dsn

The name of the control file data set to be used.

Listing Type

The type of Coverage Utility control card template that is retrieved from *hi_lev_qual*.SEQASAMP if you select Reset or if you select Edit *and* the data set or member in the **Control File Dsn** field does not exist:

- COBOL
- PL/I or PLI
- C or C++ or CPP (all are equivalent to Coverage Utility)
- ASM

3. Press Enter.

An ISPF edit session for the data set that you identified is displayed. If the data set did not previously exist, it is created with comments to help you enter the appropriate information in the fields.

4. Edit the control file to include all programs that you want to be monitored in this session.
5. Press the End key (PF3) to save your information and exit the edit session.

Contents of the control file

The control file consists of a series of statements that specify information about the desired coverage:

- The **INCLUDE** statement. This statement enables control statements in a separate data set to be processed as if they were a part of the current data set. The operand of the **INCLUDE** statement must specify either:
 - The data set name of the file to be included
 - The DD name of a previously allocated file that is to be included
- The **DEFAULTS** statement. This statement enables defaults to be set for certain keywords on subsequent COBOL, PL/I, C, C++, and ASM statements.
- The compilation unit (COBOL, PL/I, C, C++, or ASM) statement. This statement provides the following information:
 - The type of listing file (COBOL, PL/I, C, C++, and ASM)
 - The name of the data set that contains the compiler listing of the compilation unit of interest.
 - The name of the load module that contains the code from the listing
 - The data set that contains either the object code generated by the compiler or the load module created by the linker/binder
 - The data set that is to contain either the instrumented object code generated by the setup job or the instrumented load module generated by the setup job

If breakpoints are placed in object modules, specify a compilation unit only once in a control file. If the compilation unit is link-edited into more than one load module, list it for just one of the load modules.

Within a control file, each compilation unit should contain a unique external program name

- For COBOL, PROGRAM-ID
- For PL/I except for VisualAge PL/I and Enterprise PL/I for z/OS and OS/390 , external procedure name
- For VisualAge PL/I and Enterprise PL/I for z/OS and OS/390, listing data set name
- For C/C++, listing data set name
- For assembler, CSECT name.

“Examples: Control files” on page 60

Related references

Appendix C, “DBCS support,” on page 191

Syntax of control file statements

The syntax of all control statements that are used in the control file follows the same general rules:

- Statements are free-form (not column dependent).
- An asterisk in column 1 indicates a comment.
- Two consecutive slashes (//) indicate that the rest of the line after the two slashes is a comment.
- Lines that contain nothing but blanks are ignored.
- Keywords and operands can be coded in any combination of uppercase and lowercase characters.
- Operands can appear in any order.
- Operands must be separated by a comma.
- One or more blanks can appear between keywords and the corresponding equal sign, the equal sign and the operand, and the operand and the following comma.
- The order of statements is not significant *except* that:
 - All labels must be defined before they are referenced.
 - The DEFAULTS statement is position-dependent; it applies only to the statements that follow it.
 - The default value for some operands is the previous statement of the proper type.
- Statements can be continued by interrupting the line after a comma and continuing the statement on the next line.
- Labels, if present, are specified before the statement name and must be immediately followed by a colon. Labels cannot contain embedded blanks, commas, parentheses, or equal signs.
- Labels that are specified on COBOL, PL/I, C, C++, and ASM statements cannot be repeated on any of those statements.
- Operands that are shown in the syntax diagrams as being enclosed in parentheses, do not have to be enclosed in parentheses if the operand contains no embedded blanks or commas.

membername

The name of the load module that contains this compilation unit.

fromobjdsn

The data set name of the partitioned data set that contains the object generated by the compiler for this compilation unit.

fromloaddsn

The data set name of the partitioned data set that contains the load module generated by the linker/binder. Not applicable for VisualAge PL/I, Version 2 Release 2, and Enterprise PL/I for z/OS and OS/390.

fromvol

The volume that contains the *fromobjdsn* or *fromloaddsn* data set if it is not cataloged.

fromunit

The unit specification for the *fromobjdsn* or *fromloaddsn* data set if it is not cataloged.

toobjdsn

The data set name of the partitioned data set that will contain the instrumented object created by setup for this compilation unit.

toloaddsn

The data set name of the partitioned data set that will contain the instrumented load module generated by setup. Not applicable for VisualAge PL/I, Version 2 Release 2, and Enterprise PL/I for z/OS and OS/390.

tovol

The volume that contains the *toobjdsn* or *toloaddsn* data set if it is not cataloged.

tounit

The unit specification for the *toobjdsn* or *toloaddsn* data set if it is not cataloged.

COBOL statement (compilation unit definition)

The COBOL statement identifies a COBOL compilation unit.

The syntax of the COBOL statement is:

```
▶ [label:] COBOL LISTDSN=listdsname, [LISTMEMBER=listmember, ]
▶ LOADMOD=membername,
▶ FROMOBJDSN=fromobjdsn, [ FVU [ OBJMEMBER=objmem, ] TOOBJDSN=toobjdsn ] TVU
  [ FROMLOADDSN=fromloaddsn, [ FVU [ TOLOADDSN=toloaddsn ] TVU ] ]
```

FVU:

```
[ FROMVOL=fromvol, FROMUNIT=fromunit, ]
```

TVU:

```
[ , TOVOL=tovol, TOUNIT=tounit ]
```

label

A label that can be used to refer to this statement in subsequent statements.

listdsname

The name of the data set that contains the compiler listing for this compilation unit.

listmember

The member name to be substituted for an asterisk specification in the *listdsname*. This operand is usually specified only when the LISTDSN operand is specified on the DEFAULTS statement.

membername

The name of the load module (member name) that contains this compilation unit.

fromobjdsn

The data set name of the partitioned data set that contains the object modules generated by the compiler.

fromloaddsn

The data set name of the partitioned data set that contains the load module generated by the linker/binder.

fromvol

The volume that contains the *fromobjdsn* or *fromloaddsn* data set if it is not cataloged.

fromunit

The unit specification for the *fromobjdsn* or *fromloaddsn* data set if it is not cataloged.

objmem

The member name in the *fromobjdsn* for the object for this compilation unit. **If *fromobjdsn* is specified and this operand is not specified, LISTMEMBER must be specified.** In this case, the member name that is specified as *listmember* is used for *objmem*.

toobjdsn

The data set name of the partitioned data set that will contain the modified object modules generated by setup.

toloaddsn

The data set name of the partitioned data set that will contain the instrumented load module generated by setup.

tovol

The volume that contains the *toobjdsn* or *toloaddsn* data set if it is not cataloged.

tounit

The unit specification for the *toobjdsn* or *toloaddsn* data set if it is not cataloged.

In a set of control cards you can process either object modules or load modules, but not both. Thus, the FROMOBJDSN and FROMLOADDSN keywords are mutually exclusive, as are the TOOBJDSN and TOLOADDSN keywords.

If TOOBJDSN is coded, then FROMOBJDSN is required.

If TOLOADDSN is coded, FROMLOADDSN is optional. If FROMLOADDSN is omitted, its value defaults to the value that is specified for TOLOADDSN and, hence, the load module will be modified in the data set where it is currently located.

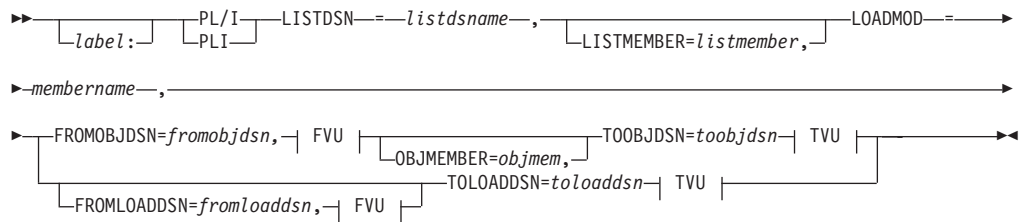
The *fromloaddsn* load module can be coded as an * to nullify the FROMLOADDSN specification on a previous DEFAULTS statement.

EQACUZPP is used to instrument breakpoints in a load module. It is invoked when either the *membername* or *toloaddsn* changes, or when the last control card is processed.

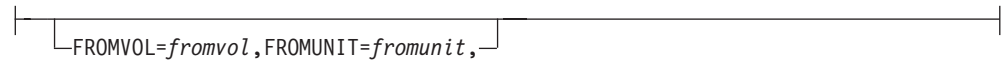
PL/I statement (compilation unit definition)

The PL/I statement identifies a PL/I compilation unit.

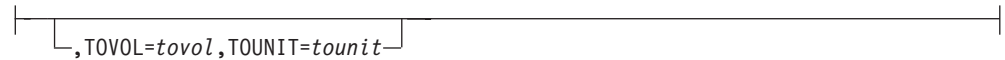
The syntax of the PL/I statement is:



FVU:



TVU:



label

A label that can be used to refer to this statement in subsequent statements.

listdsname

The name of the data set that contains the compiler listing for this compilation unit.

listmember

The member name to be substituted for an asterisk specification in the *listdsname*. This operand would usually be specified only when the LISTDSN operand is specified on the DEFAULTS statement.

membername

The name of the load module (member name) that contains this compilation unit.

fromobjdsn

The data set name of the partitioned data set that contains the object modules generated by the compiler.

fromloaddsn

The data set name of the partitioned data set that contains the load module generated by the linker/binder. Not applicable for VisualAge PL/I, Version 2 Release 2, and Enterprise PL/I for z/OS and OS/390.

fromvol

The volume that contains the *fromobjdsn* or *fromloaddsn* data set if it is not cataloged.

fromunit

The unit specification for the *fromobjdsn* or *fromloaddsn* data set if it is not cataloged.

objmem

The member name in the *fromobjdsn* for the object for this compilation unit. **If *fromobjdsn* is specified and this operand is not specified, LISTMEMBER must be specified.** In this case, the member name specified as *listmember* is used for *objmem*.

toobjdsn

The data set name of the partitioned data set that will contain the modified object modules generated by setup.

toloaddsn

The data set name of the partitioned data set that will contain the instrumented load module generated by setup. Not applicable for VisualAge PL/I, Version 2 Release 2, and Enterprise PL/I for z/OS and OS/390.

tovol

The volume that contains the *toobjdsn* or *toloaddsn* data set if it is not cataloged.

tounit

The unit specification for the *toobjdsn* or *toloaddsn* data set if it is not cataloged.

In a set of control cards you can process either object modules or load modules, but not both. Thus, the FROMOBJDSN and FROMLOADDSN keywords are mutually exclusive, as are the TOOBJDSN and TOLOADDSN keywords.

If TOOBJDSN is coded, then FROMOBJDSN is required.

If TOLOADDSN is coded, FROMLOADDSN is optional. If FROMLOADDSN is omitted, its value defaults to the value specified for TOLOADDSN and, hence, the load module will be modified in the data set in which it is currently located.

The *fromloaddsn* load module can be coded as an * to nullify the FROMLOADDSN specification on a previous DEFAULTS statement.

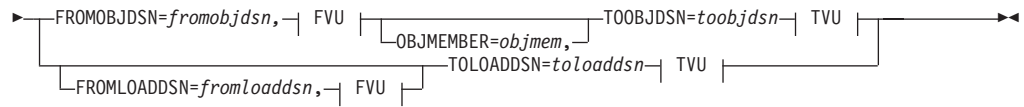
EQACUZPP is used to instrument breakpoints in a load module. It is invoked when either the *membername* or *toloaddsn* changes, or when the last control card is processed.

C statement (compilation unit definition)

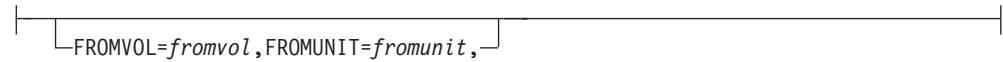
The C statement identifies a C or C++ program compilation unit. Although C, C++, or CPP can be specified, all of these are equivalent. Coverage Utility makes no distinction between processing C and C++ code.

The syntax of the C statement is:

```
▶ [label:] [ C | C++ | CPP ] LISTDSN=--listdsname--, [LISTMEMBER=listmember,] LOADMOD=--▶
▶--membername--▶
```



FVU:



TVU:



label

A label that can be used to refer to this statement in subsequent statements.

listdsname

The name of the data set that contains the compiler listing for this compilation unit.

listmember

The member name to be substituted for an asterisk specification in the *listdsname*. This operand is usually specified only when the LISTDSN operand is specified on the DEFAULTS statement.

membername

The name of the load module (member name) that contains this compilation unit.

fromobjdsn

The data set name of the partitioned data set that contains the object modules generated by the assembler.

fromloadsn

The data set name of the partitioned data set that contains the load module generated by the linker/binder. If *fromloadsn* is specified, the CSECT compiler option or a #pragma CSECT(CODE,...) must have been used to create a CODE CSECT name of no more than eight alphanumeric uppercase characters.

fromvol

The volume that contains the *fromobjdsn* or *fromloadsn* data set if it is not cataloged.

fromunit

The unit specification for the *fromobjdsn* or *fromloadsn* data set if it is not cataloged.

objmem

The member name in the *fromobjdsn* for the object for this compilation unit. **If *fromobjdsn* is specified and this operand is not specified, LISTMEMBER must be specified.** In this case, the member name that is specified as *listmember* is used for *objmem*.

toobjdsn

The data set name of the partitioned data set that will contain the modified object modules generated by setup.

toloaddsn

The data set name of the partitioned data set that will contain the instrumented load module generated by setup.

tovol

The volume that contains the *toobjdsn* or *toloaddsn* data set if it is not cataloged.

tounit

The unit specification for the *toobjdsn* or *toloaddsn* data set if it is not cataloged.

In a set of control cards you can process either object modules or load modules, but not both. Thus, the FROMOBJDSN and FROMLOADDSN keywords are mutually exclusive, as are the TOOBJDSN and TOLOADDSN keywords.

If TOOBJDSN is coded, then FROMOBJDSN is required.

If TOLOADDSN is coded, FROMLOADDSN is optional. If FROMLOADDSN is omitted, its value defaults to the value that is specified for TOLOADDSN and, hence, the load module will be modified in the data set in which it is currently located.

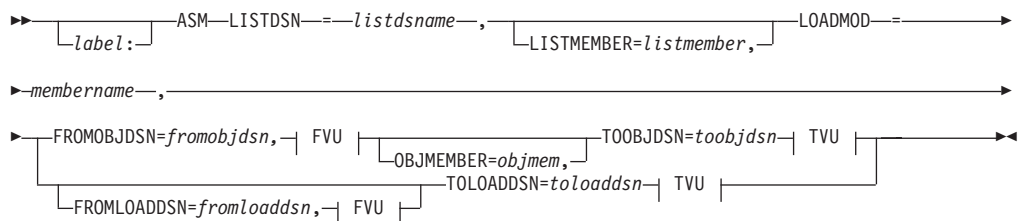
The *fromloaddsn* load module can be coded as an * to nullify the FROMLOADDSN specification on a previous DEFAULTS statement.

EQACUZPP is used to instrument breakpoints in a load module. It is invoked when either the *membername* or *toloaddsn* changes, or when the last control card is processed.

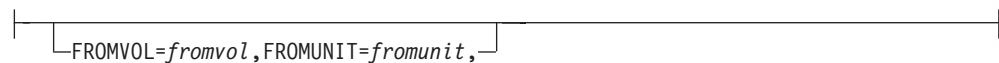
ASM statement (compilation unit definition)

The ASM statement identifies an assembler program compilation unit.

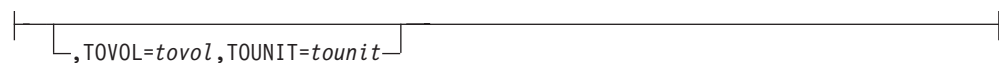
The syntax of the ASM statement is:



FVU:



TVU:



label

A label that can be used to refer to this statement in subsequent statements.

listdsname

The name of the data set that contains the assembler listing for this compilation unit.

listmember

The member name to be substituted for an asterisk specification in the *listdsname*. This operand is usually specified only when the LISTDSN operand is specified on the DEFAULTS statement.

membername

The name of the load module (member name) that contains this compilation unit.

fromobjdsn

The data set name of the partitioned data set that contains the object modules generated by the assembler.

fromloaddsn

The data set name of the partitioned data set that contains the load module generated by the linker/binder.

fromvol

The volume that contains the *fromobjdsn* or *fromloaddsn* data set if it is not cataloged.

fromunit

The unit specification for the *fromobjdsn* or *fromloaddsn* data set if it is not cataloged.

objmem

The member name in the *fromobjdsn* for the object for this compilation unit. **If *fromobjdsn* is specified and this operand is not specified, LISTMEMBER must be specified.** In this case, the member name that is specified as *listmember* is used for *objmem*.

toobjdsn

The data set name of the partitioned data set that will contain the modified object modules generated by setup.

toloaddsn

The data set name of the partitioned data set that will contain the instrumented load module generated by setup.

tovol

The volume that contains the *toobjdsn* or *toloaddsn* data set if it is not cataloged.

tounit

The unit specification for the *toobjdsn* or *toloaddsn* data set if it is not cataloged.

In a set of control cards you can process either object modules or load modules, but not both. Thus, the FROMOBJDSN and FROMLOADDSN keywords are mutually exclusive, as are the TOOBJDSN and TOLOADDSN keywords.

If TOOBJDSN is coded, then FROMOBJDSN is required.

If TOLOADDSN is coded, FROMLOADDSN is optional. If FROMLOADDSN is omitted, its value defaults to the value that is specified for TOLOADDSN and, hence, the load module will be modified in the data set where it is currently located.

The *fromloaddsn* load module can be coded as an * to nullify the FROMLOADDSN specification on a previous DEFAULTS statement.

EQACUZPP is used to instrument breakpoints in a load module. It is invoked when either the *membername* or *toloaddsn* changes, or when the last control card is processed.

Examples: Control files

The following examples show how you might use these statements.

Example: Control file for a single compilation unit

This example shows a typical control file where you request coverage for a single compilation unit.

```
Cobol ListDsn=YOUNG.SAMPLE.Cobolst(Cob01),
      LoadMod=Cob01,
      FromObjDsn=YOUNG.SAMPLE.Obj,
      ToObjDsn=YOUNG.SAMPLE.ZapObj
```

Example: Control file for multiple compilation units

This example shows a typical control file where you request coverage for multiple compilation units.

```
Defaults ListDsn=YOUNG.SAMPLE.Cobolst(*),
          LoadMod=Cob01,
          FromObjDsn=YOUNG.SAMPLE.Obj,
          ToObjDsn=YOUNG.SAMPLE.ZapObj
*
Cobol ListMember=Cob01A
Cobol ListMember=Cob01B
Cobol ListMember=Cob01C
Cobol ListMember=Cob01D
```

Example: Control file for load module

This example shows a typical control file where you request coverage for multiple compilation units and where breakpoints are to be placed directly into a load module (rather than into object modules):

```
Defaults ListDsn=YOUNG.SAMPLE.Cobolst(*),
          LoadMod=Cob01,
          FromLoadDsn=YOUNG.SAMPLE.LOADLIB,
          ToLoadDsn=YOUNG.SAMPLE.RUNLIB
*
Cobol ListMember=Cob01A
Cobol ListMember=Cob01B
Cobol ListMember=Cob01C
Cobol ListMember=Cob01D
```

Chapter 6. Preparing to monitor a program

This section describes the steps that you need to perform to prepare a program to be monitored. This function is called setup.

- “Supplying setup input”
- “Instrumenting object modules or load modules” on page 62
- “Creating the setup JCL by using the panels” on page 63
- “Determining when to create or submit setup JCL” on page 65
- “Compiler options required by Coverage Utility” on page 65
- “Compiler restrictions imposed by Coverage Utility” on page 68
- “Setup JCL for the compile job stream” on page 70
- “Parameters for the setup programs” on page 70

Supplying setup input

The setup process uses assembler statements from the listings produced by the compiler to determine where to place breakpoints.

Prepare the following items for setup:

- A Coverage Utility control file.
- Compiler listings with assembler statements included.
- The original object modules created during the source compile step or the original load modules created during the link-edit step. (Applying breakpoints directly to load modules is not supported for load modules generated by the VisualAge PL/I or Enterprise PL/I for z/OS and OS/390 compilers.)
- An object library (allocated like the original) to receive the new object modules (after they have been modified with breakpoints), or a load library to receive the new load modules. (Applying breakpoints directly to load modules is not supported for load modules generated by the VisualAge PL/I or Enterprise PL/I for z/OS and OS/390 compilers.)

Modify this information as necessary in the control file.

Related concepts

“Setup processing”

Related tasks

Chapter 5, “Describing the compile units to be analyzed,” on page 49

Related references

“Restrictions on setup input” on page 62

“Instrumenting object modules or load modules” on page 62

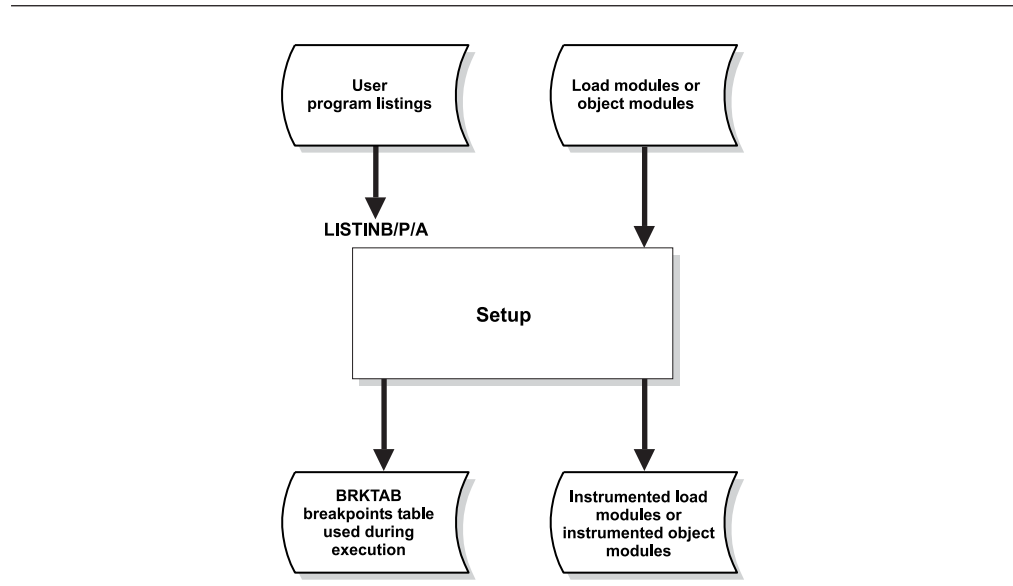
“Compiler options required by Coverage Utility” on page 65

“Compiler restrictions imposed by Coverage Utility” on page 68

Setup processing

Setup creates a breakpoint table (BRKTAB) and uses it together with the object module that is created at compile time to create a new object module that contains the breakpoint data. After the breakpoints have been inserted into the object module or modules, link-edit the object modules into the executable load module. Alternatively, the setup job can insert breakpoints directly into load modules.

The steps involved in the setup procedure for Coverage Utility are shown in the following figure.



DD names: The names outside the box in the figure (for example, LISTINP) correspond to the DD names for the listings.

Restrictions on setup input

The following restrictions apply to the input to setup:

- Each listing can contain one assembly. If you want only a summary report of code coverage, you can discard the compiler listings after the setup step.
- Coverage Utility does not support inserting breakpoints into self-modifying code.
- Within a control file, each compilation unit should contain a unique external program name (for COBOL, PROGRAM-ID; for PL/I except for VisualAge PL/I and Enterprise PL/I for z/OS and OS/390, external procedure name; for VisualAge PL/I and Enterprise PL/I for z/OS and OS/390, listing data set name; for C/C++, listing data set name; and for assembler, CSECT name).

Instrumenting object modules or load modules

You can instrument breakpoints into your object modules before you link them into an executable load module, or you can instrument breakpoints into the executable load module. Instrumentation of load modules is useful when instrumenting object modules is difficult, because the link step is built into your standard location-wide build procedures.

Instrumenting load modules is not supported for VisualAge PL/I, Version 2 Release 2, or Enterprise PL/I for z/OS and OS/390.

The samples that are shipped with Coverage Utility show you how to instrument object modules. To instrument object modules for the COB01 test case, use the following control file:

```
Defaults ListDsn=YOUNG.SAMPLE.COBOLST(*),
        LoadMod=COB01,
        FromObjDsn=YOUNG.SAMPLE.OBJ,
        ToObjDsn=YOUNG.SAMPLE.ZAPOBJ
```



```
COB01A:    COBOL ListMember=COB01A
COB01B:    COBOL ListMember=COB01B
COB01C:    COBOL ListMember=COB01C
COB01D:    COBOL ListMember=COB01D
```

To instrument load modules, you must change the control file. All other steps remain the same, except that you can skip the step that links the instrumented object modules into a program to test.

For example, to instrument the load module for the COB01 test case, use the following COBOL control file:

```
Defaults ListDsn=YOUNG.SAMPLE.COBOLST(*),
LoadMod=COB01,
FromLoadDsn=YOUNG.SAMPLE.LOADLIB,
ToLoadDsn=YOUNG.SAMPLE.RUNLIB

COB01A:    COBOL ListMember=COB01A
COB01B:    COBOL ListMember=COB01B
COB01C:    COBOL ListMember=COB01C
COB01D:    COBOL ListMember=COB01D
```

You do not use the FromObjDsn and ToObjDsn keywords that identify object module libraries. Instead you use the FromLoadDsn and ToLoadDsn keywords. If you supply both FromLoadDsn and ToLoadDsn keywords, the load module is read from the FromLoadDsn data set and written to the ToLoadDsn data set after it has been instrumented. If you supply only the ToLoadDsn keyword, the load module is instrumented in place.

You cannot use both object module instrumentation (use of FromObjDsn and ToObjDsn keywords) and load module instrumentation (FromLoadDsn and ToLoadDsn keywords) in the same set of control cards. If load module instrumentation is used for C/C++, the CSECT compiler option or #pragma CSECT(CODE,...) must be specified during the compilation to create a CODE CSECT name of no more than eight alphanumeric characters.

The instrumentation of the load module is done using the EQACUZPP program.

Related tasks

“Editing the sample control file” on page 21

Related references

“EQACUZPP” on page 72

Creating the setup JCL by using the panels

To create the setup JCL do the following steps:

1. Select option 2 from the Debug Tool Coverage Utility panel.
The Create JCL for Setup panel, shown below, is displayed.
2. Enter any information that you want to change, select option 1, and then press Enter.

```

----- Create JCL for Setup -----
Option ==>

1  Generate      Generate JCL from parameters
2  Edit          Edit JCL
3  Submit        Submit JCL

Enter END to Terminate

Use Program Name for File Name YES (Yes|No) Program Name COB01

Control File:
  Control File Dsn. . . 'YOUNG.SAMPLE.DTCU(COB01)'

JCL Library and Member:
  JCL Dsn . . . . . 'YOUNG.SAMPLE.JCL(SCOB01)'

Output Breakpoint Table:
  Breakpoint Table Dsn. 'YOUNG.SAMPLE.COB01.BRKTAB'

```

The options and fields in the panel are as follows. In most cases, you need to change only the **Program Name** field, and then press Enter. The defaults for the setup step are used.

Generate

Generates JCL using the parameters that you specify on the panel.

Edit Displays an ISPF edit session where you can change existing JCL.

Submit

Submits for execution the JCL that you specify in the **JCL Dsn** field on this panel.

Use Program Name for File Name

Enter YES if you want to construct the data set names from the default high-level qualifier, the specified program name, and the default low-level qualifier for each data set.

When you press Enter, the file names on the panel are changed automatically. Coverage Utility usually constructs the data set names by using the program name.

Program Name

The name to use for Coverage Utility files when you enter YES in the **Use Program Name for File Name** field. This name can be any valid name; it does not need to be the name of any of your programs. Names of the following form are created:

- Sequential data sets:
'proj_qual.program_name.file_type'

For example: 'YOUNG.SAMPLE.COB01.BRKTAB'

- Partitioned data sets:
'proj_qual.file_type(program_name)'

For example: 'YOUNG.SAMPLE.BRKTAB(COB01)'

Control File Dsn

The name of the control file data set that contains the names of the listing

files that you want to be annotated. If you want to edit the control file, you can do so from the Work with the Control File panel.

JCL Dsn

The name of the JCL data set that contains the JCL for this action.

Default: If you set the **Use Program Name for File Name** field to YES, then the member name or program name qualifier of the data set will be *Sxxxxxxx*, where *xxxxxxx* is the last seven characters of the program name.

Breakpoint Table Dsn

The name of the BRKTAB data set that is created during setup and used by the monitor program.

Determining when to create or submit setup JCL

You must run the setup JCL if you change your program (and consequently, the listing).

If the test environment changes, you must recreate the setup JCL only . For example, if you add or delete listings in the Coverage Utility control cards, recreate the setup JCL from the panel, and run the new setup JCL. However, if you change a listing, then submit the old setup JCL without changes.

Compiler options required by Coverage Utility

The following tables show the compiler options required by Coverage Utility for each supported language and compiler. You can also use options in addition to those shown.

COBOL compiler options required by Coverage Utility

The following table shows the COBOL compiler options that Coverage Utility requires:

Compiler	Required options
Enterprise COBOL for z/OS and OS/390 COBOL for OS/390 & VM COBOL for MVS & VM VS COBOL II	SOURCE ¹ LIST ² OBJECT NOOPTIMIZE NONUMBER LIB ³
OS/VS COBOL	SOURCE PMAP OBJECT NOLST NOOPTIMIZE NONUM NOBATCH NOTEST NOFLOW NOSYMDMP NOCOUNT

Compiler	Required options
	<ol style="list-style-type: none"> 1. You specify the *CBL (*CONTROL) NOSOURCE compiler-directing statement to suppress printing of COBOL executable statements; these statements will not be included in the annotated listing report. 2. You specifying the *CBL (*CONTROL) NOLIST compiler-directing statement to suppress printing of the assembler code; Coverage Utility cannot insert breakpoints into the suppressed assembler code. 3. The LIB compiler option is required by Coverage Utility only if you have multiple source programs separated by CBL (PROCESS) compiler-directing statements.

PL/I compiler options required by Coverage Utility

The following table shows PL/I compiler options that Coverage Utility requires:

Compiler	Required options
Enterprise PL/I for z/OS and OS/390	SOURCE LIST OBJECT NOSTMT NOOPTIMIZE NOTEST or TEST(NONE) ² NOBLKOFF ⁵ NUMBER
VisualAge PL/I Version 2 Release 2	SOURCE LIST OBJECT NOSTMT NOINSOURCE ¹ NOOFFSET NOOPTIMIZE NOTEST or TEST(NONE) ²
PL/I for MVS & VM ³	SOURCE LIST OBJECT NOOPTIMIZE NOTEST or TEST(NONE) ²
PL/I 2.3.0 ^{3 4}	SOURCE LIST OBJECT NOOPTIMIZE NOTEST ² NOCOUNT NOFLOW
PL/I 1.5.1 ^{3 4}	SOURCE LIST OBJECT NOOPTIMIZE NOCOUNT NOFLOW

Compiler	Required options
	<ol style="list-style-type: none"> Required only if you are running VisualAge PL/I 2.2.0 without APAR PQ40062. Although NOTEST is recommended, you can use TEST. However, using TEST can cause additional breakpoints to be created for the TEST hooks. These breakpoints can vary depending on the TEST suboptions in effect. The additional breakpoints, if any, will appear in the summary and annotation reports. You specify the %NOPRINT statement to suppress printing of PL/I executable statements; these statements will not be included in the annotated listing report produced by Coverage Utility. Multiple PL/I external source programs that are separated by *PROCESS statements are not supported. Coverage Utility requires that PTF PN49349 be applied to the IBM OS PL/I Optimizing Compiler 2.3.0 to provide support for more than 9999 statements. The IBM PL/I Optimizing Compiler 1.5.1 does not support more than 9999 statements. Coverage Utility requires that PTF UQ71463 be applied to Enterprise PL/I for z/OS and OS/390, Version 3 Release 1, or PTF UQ71704 be applied to Enterprise PL/I for z/OS and OS/390, Version 3 Release 2, to provide support for the NOBLKOFF compiler option.

C/C++ compiler options required by Coverage Utility

The following table shows C/C++ compiler options required by Coverage Utility:

Compiler	Required options
OS/390 C/C++ compiler	CSECT ^{1, 2, 3} NOGOFF ⁴ INLINE or NOINLINE ⁵ NOIPA LIST NOOFFSET OPTIMIZE or NOOPTIMIZE ⁵ SOURCE NOTEST ⁶
	<ol style="list-style-type: none"> You can use #pragma CSECT(CODE,...) instead of the CSECT compiler option. The CSECT option is required only if you are applying breakpoints to the load module. If you are applying breakpoints to the object modules only, the use of CSECT is optional. The CODE CSECT name must be no longer than eight alphanumeric characters. NOGOFF is required only if breakpoints are applied to object modules. It is not required when breakpoints are applied to load modules. Using the OPTIMIZE or INLINE options can change the summary statistics and annotations produced by Coverage Utility. Although NOTEST is recommended, you can use TEST. However, using TEST can cause additional breakpoints to be created for the TEST hooks. These breakpoints can vary depending on the TEST suboptions in effect. The additional breakpoints, if any, will appear in the summary and annotation reports.

Related references

Chapter 12, "Report differences for optimized C/C++ code," on page 135

Assembler options required by Coverage Utility

The following table shows assembler options required by Coverage Utility:

Assembler	Required options
High Level Assembler	NOBATCH ESD NOGOFF or NOXOBJECT ¹ LIST(121) ² OBJECT or DECK PCONTROL(GEN,MSOURCE,ON)
Assembler H ³	NOBATCH ESD LIST OBJECT or DECK
<ol style="list-style-type: none">1. NOGOFF or NOXOBJECT is required only if breakpoints are applied to object modules. It is not required when breakpoints are applied to load modules.2. This option can be implied by LIST without operands (depending on the installation options in effect).3. You specify the PRINT assembler directive to suppress printing of executable instructions or data;setup cannotinsert any breakpoints into the suppressed assembler code.	

If your code contains a branch whose target uses location counter relative addressing, such as B *+20, then you should enable the Frequency Count Mode flag for setup to get accurate summary and annotation information for the target locations.

Related references

“Parameters for the setup programs” on page 70

Compiler restrictions imposed by Coverage Utility

The following sections describe certain compiler restrictions that are imposed by Coverage Utility for programs to be monitored. These include listing attributes and language constructs.

COBOL compiler restrictions imposed by Coverage Utility

The following restrictions are imposed on COBOL compilations by Coverage Utility:

- Each listing can contain multiple COBOL paragraphs. Each paragraph is listed in the summary report as a separate program area (PA). The PA name is the paragraph name.
- COBOL listings must have the following DCB attributes:

DSORG=PSDSORG=PO
For all compilers.

RECFM=FBA
For all compilers.

LRECL=133
For Enterprise COBOL for z/OS and OS/390, COBOL for OS/390 & VM, COBOL for MVS & VM, and VS COBOL II.

LRECL=121
For OS/VS COBOL.

PL/I compiler restrictions imposed by Coverage Utility

The following restrictions are imposed on PL/I compilations by Coverage Utility.

- Each listing can contain one external procedure, and multiple internal procedures, ON-units, and BEGIN blocks. Each of these is listed in the summary report as a separate program area (PA). The PA name is the name of the procedure or labeled BEGIN block, or is a compiler-generated name or Coverage Utility-generated name for ON-units and unlabeled BEGIN blocks.
- The following PL/I condition prefixes should not be enabled:
 - SUBSCRIPTRANGE
 - STRINGRANGE
 - CHECK
- All PL/I listings must have the following DCB attributes:
 - DSORG=PO or DSORG=PS
 - RECFM=VBA
 - LRECL=125
 - For PL/I for MVS & VM, PL/I Version 2 Release 3, PL/I Version 1 Release 5 Mod 1
 - LRECL=132
 - VisualAge PL/I Version 2 Release 2 or Enterprise PL/I for z/OS and OS/390 Version 3 Release 1
 - LRECL=137
 - Enterprise PL/I for z/OS and OS/390 Version 3 Release 2 or later

For VisualAge PL/I and Enterprise PL/I for z/OS and OS/390, breakpoints are set in machine code that is generated from source code in the primary input file only. Machine code that is generated as a result of source code included in the compilation by a preprocessor INCLUDE statement is not part of the by Coverage Utility analyzes.

C/C++ compiler restrictions imposed by Coverage Utility

The following restrictions are imposed on C/C++ compilations by Coverage Utility.

- All C/C++ listings must have the following DCB attributes:
 - DSORG=PO or DSORG=PS
 - RECFM=VBA
 - LRECL=137
- For C/C++, breakpoints are set only in machine code that is generated from source code in the primary input file. Machine code that is generated as a result of source code included in the compilation by a preprocessor #include statement is not part of the code that Coverage Utility analyzes. Therefore code that is generated by instantiation of templates will not have breakpoints placed in it.

Assembler restrictions imposed by Coverage Utility

All ASM listings must have the following DCB attributes:

- DSORG=PO or DSORG=PS
- RECFM=FBM or FBA
- LRECL=133 for the High Level Assembler
- LRECL=121 for Assembler H

Setup JCL for the compile job stream

If you insert the JCL that creates setup output files into your compile job stream, the setup output files are created and saved automatically whenever you change a module and recompile.

You can create this JCL by generating the setup JCL from the Create JCL for Setup panel, and then editing it to suit your needs.

Parameters for the setup programs

The setup programs are used to instrument user programs with breakpoints. The setup programs that you use depend on whether you are instrumenting object modules or load modules.

The following table shows the programs that are used in each of these situations:

Program	Invoked once per	Purpose
Object: If you are instrumenting object modules, you use the following setup programs:		
EQACUSET	Compilation unit	To analyze listings and create the BRKTAB for the compilation unit
EQACUZPT	Compilation unit	To read the BRKTAB and instrument the object module
Load: If you are instrumenting load modules, use the following setup programs:		
EQACUSET	Compilation unit	To analyze listings and create the BRKTAB for the compilation unit
EQACUZPL	Compilation unit	To read the BRKTAB and create or append control statements for EQACUZPP
EQACUZPP	Load module	To read control statements that are created by EQACUZPL and that instrument the load module

The setup program input parameters are built automatically by the ISPF dialog.

Related references

Appendix B, "Resources and requirements," on page 187

EQACUSET

This program analyzes the compiler listings to determine breakpoint placement.

The parameters are as follows:

```
►►—LoadMod—,—LDFP—,—'Listing_data_set_name' —,—2-byte_SVC_#—,——————►
►—4-byte_SVC_#—└──,common_parameters──┘—————►
```

LoadMod

The name of the load module that contains the Program Area (PA) or CSECT name.

LDFP

This is a string of four characters with no embedded blanks. Each character specifies a separate parameter in the following order:

- *List_Type*
- *Debug_mode*
- *Frequency_count_mode*
- *Performance_mode*



List_Type

The type of listing.

- B** COBOL
- P** PL/I
- C** C/C++
- A** Assembler

Debug_mode

Use debug mode?

- Y** Use debug mode: execute breakpoints as many times as encountered.

Attention:: Use debug mode only at the direction of Coverage Utility support personnel, or as directed in this guide.

- N** Do not use debug mode: execute breakpoints once only. Set to N unless you want the execution count information to be displayed in the annotated listing report.

Frequency_count_mode

Use frequency count mode?

- Y** Save frequency and branch-to information (for branches) in the coverage file (BRKOUT). This information adds 16 bytes per breakpoint to the size of the BRKOUT file.
- N** Do not save frequency and branch-to information (for branches) in the coverage file (BRKOUT). Set to N unless you are analyzing assembler code with location counter relative addressing, or you want the execution count information to be displayed in the annotated listing report.

Performance_mode

Use performance mode?

- Y** Conditional branch coverage is disabled.
- N** Conditional branch coverage is enabled.

Listing_data_set_name

The name of the listing data set, delimited by apostrophes. If it is a PDS, you must specify a member name, using the form 'datasetname(member)'.

2-byte_SVC_#

SVC number, in hexadecimal notation, for inserting breakpoints into two-byte instructions.

4-byte_SVC_#

SVC number, in hexadecimal notation, for inserting breakpoints into four- and six-byte instructions.

common_parameters

Any of the parameters that are common to multiple routines, separated by commas.

Related tasks

“Displaying execution counts in an annotated listing report” on page 126

“Using performance mode to reduce monitor overhead” on page 82

Related references

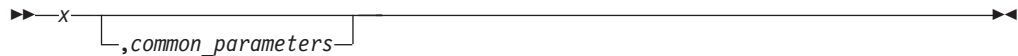
“Assembler options required by Coverage Utility” on page 68

Appendix E, “Parameters that are common to multiple routines,” on page 205

EQACUZPT

The EQACUZPT program is used to insert breakpoints into object modules.

The parameters are as follows:



x A number that indicates which BRKTAB in the BRKTAB data set to use.

common_parameters

Any of the parameters that are common to multiple routines, separated by commas.

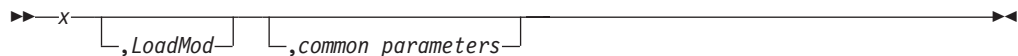
Related references

Appendix E, “Parameters that are common to multiple routines,” on page 205

EQACUZPL

The EQACUZPL program is used to build input for EQACUZPP to insert breakpoints into load modules.

The parameters are as follows:



x A number that indicates which BRKTAB in the BRKTAB data set to use.

LoadMod

An optional parameter that contains the name of the load module to which the breakpoints are applied. If specified, it overrides the load module name that is stored in the BRKTAB (from setup).

common_parameters

Any of the parameters that are common to multiple routines, separated by commas.

Related references

Appendix E, “Parameters that are common to multiple routines,” on page 205

EQACUZPP

The EQACUZPP program inserts breakpoints in user load modules. It takes an input file (SYSIN) of commands that specify the breakpoints to be applied, reads one or more input load modules or program objects from the LIBIN file, applies the requested breakpoints, and writes the load modules or program objects to the LIBOUT file. The input commands that EQACUZPP supports are a subset of those supported by the AMASPZAP program.

This program requires no parameters. It accepts any of the parameters that are common to multiple routines.

```
//LOADZAP EXEC PGM=EQACUZPP,PARM='common_parameters'
```

common_parameters

Any of the parameters that are common to multiple routines, separated by commas.

Related references

Appendix E, "Parameters that are common to multiple routines," on page 205

Appendix B, "Resources and requirements," on page 187

Part 4. Running a Coverage Utility monitor session

Chapter 7. Monitoring a program	77	Restrictions on programs that reside in read-only storage	84
Creating the start monitor JCL by using the panels	77	Restrictions on system modes	84
Parameters for the monitor	79		
Running multiple user sessions.	80	Chapter 8. Monitor commands	85
Changing and using IDs	80	Issuing commands	85
Coverage of common modules with multiple user sessions	80	EQACUOBP (Display breakpoint status).	86
Example: Multiple testers running modules with unique modules per session	80	EQACUOID (Add ID).	89
Example: Multiple testers running with modules monitored in multiple sessions	81	EQACUOPF (Performance mode off).	90
Example: Multiple testers running a module, each with a unique copy	82	EQACUOPN (Performance mode on).	91
Using performance mode to reduce monitor overhead	82	EQACUOQT (Quit).	92
Monitoring a program that is executing under control of the Debug Tool debugger	83	EQACUORE (Reset)	93
Restrictions on monitoring programs	83	EQACUOSA (Display statistics).	94
		EQACUOSE (Display sessions).	96
		EQACUOSL (Display listings)	97
		EQACUOSN (Snapshot)	99
		EQACUOSP (Stop)	100

In this part of the document, learn the procedures for running a Coverage Utility monitor session, including the parameters needed. Learn how to run multiple-user sessions and to monitor a program that is running under the control of the Debug Tool debugger. The first section also provides information about restrictions for monitoring programs.

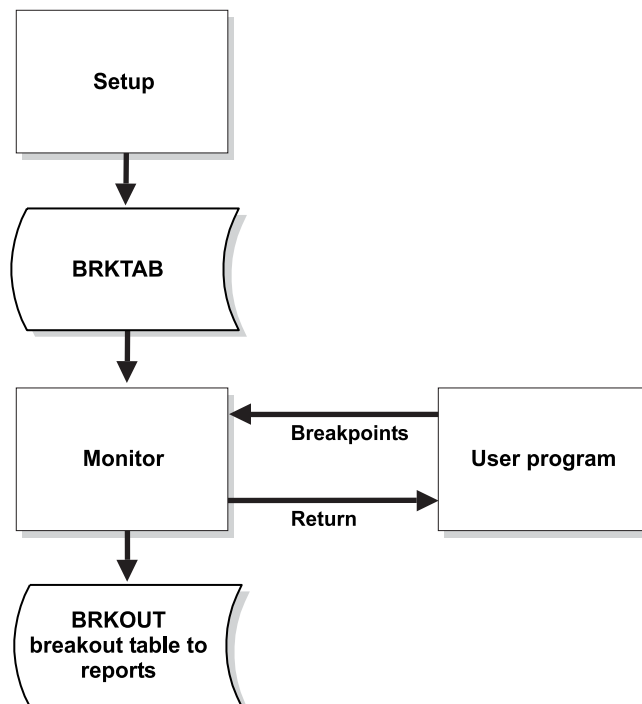
In the second section learn how to use commands to control a Coverage Utility monitor session and to display statistics.

Chapter 7. Monitoring a program

This section describes how to run a Coverage Utility monitor session on MVS. The monitor handles the user supervisor call instructions (SVCs) that are used as breakpoints (BPs).

You create the JCL to start a monitor session from the Create JCL to Start the Monitor panel. You can have multiple user sessions active at the same time.

The monitor program examines the breakpoint table information produced by the setup step and accumulates data as the application program runs. When you run the EQACUOSP command, the monitor writes the results to disk. This diagram shows the Coverage Utility execution flow:



When you start the monitor session, the tables needed for handling the session are created in ECSA. Ensure that you have the amount of storage needed for a user session.

Related tasks

“Creating the start monitor JCL by using the panels”

“Running multiple user sessions” on page 80

Related references

Appendix B, “Resources and requirements,” on page 187

Creating the start monitor JCL by using the panels

To create the start monitor JCL, do the following steps:

1. Select option 3 from the Debug Tool Coverage Utility panel.

The Create JCL to Start the Monitor panel, shown below, is displayed.

2. Enter any information that you want to change, select option 1, and then press Enter.

```
----- Create JCL to Start the Monitor -----
Option ==>

1 Generate      Generate JCL from parameters
2 Edit         Edit JCL
3 Submit       Submit JCL

Enter END to Terminate

Use Program Name for File Name YES (Yes|No) Program Name COB01

Session ID . . . . . YOUNG

Input File:
Breakpoint Table Dsn. 'YOUNG.SAMPLE.COB01.BRKTAB'

JCL Library and Member:
JCL Dsn . . . . . 'YOUNG.SAMPLE.JCL(XCOB01)'

Output File:
Breakout Dsn. . . . . 'YOUNG.SAMPLE.COB01.BRKOUT'
```

The options and fields on the panel are as follows:

Generate

Generates JCL from the parameters that you specify on the panel.

Edit Displays an ISPF edit session where you can change existing JCL.

Submit

Submits for execution the JCL that you specify in the **JCL Dsn** field on this panel.

Use Program Name for File Name

Enter YES if you want to construct the data set name from the default high-level qualifier, the specified program name, and the default low-level qualifier for each data set.

When you press Enter, the file names on the panel are changed automatically. Coverage Utility usually constructs the data set name by using the program name.

Program Name

The name to use for Coverage Utility files when you enter YES in the **Use Program Name for File Name** field. This name can be any valid name; it does not need to be the name of any of your programs. Names of the following form are created:

- Sequential data sets:
'proj_qual.program_name.file_type'

For example: 'YOUNG.SAMPLE.COB01.BRKTAB'

- Partitioned data sets:
'proj_qual.file_type(program_name)'

For example: 'YOUNG.SAMPLE.BRKTAB(COB01)'

Running multiple user sessions

More than one tester can use the Coverage Utility monitor simultaneously on an MVS system. Each separate invocation of the monitor is called a session. The monitor identifies a session by a session ID passed to it as a parameter. The Create JCL to Start the Monitor panel, which creates the JCL, creates a session ID from the tester's TSO user ID or a user-specified session ID. Each tester can start or stop a session independently of any other tester.

Changing and using IDs

You can change the session ID to a user-defined ID either by changing the Session ID option on the Create JCL to Start the Monitor panel or by editing the start monitor JCL. To change the session ID by editing the start monitor JCL, change each of the following to the ID that you choose:

- The *sessid* qualifier in the following data set names:
 - `syspref.sysuid.sessid.EXTEMP.EXEC`
 - `sysuid.sessid.EXTEMP.EXEC`
- The second parameter to the EQACUOCM program

Changing the session ID lets you create custom batch test runs for automation purposes.

Any user can stop or cancel any session if the user knows the session ID (which can be determined by issuing the EQACUOSE command). This action can be necessary, for example, if there are plans to IPL the system, which will cause the loss of test data.

Coverage of common modules with multiple user sessions

When multiple testers simultaneously run shared modules, reports are affected. In some environments (for example, CICS®), only one copy of a module is in storage for all users of that module.

In general when multiple testers run a monitored module, the following rules apply:

- When the module is monitored by only one session (its BRKTAB appears in only one session), all coverage from all testers appears in that session's coverage data.
- When the module is monitored by multiple sessions (its BRKTAB appears in multiple sessions), the first session in which the module is started will usually receive all of the coverage data. Other sessions where the module is monitored will show no coverage data.

The following examples show various scenarios in which you can use the monitor:

“Example: Multiple testers running modules with unique modules per session”

“Example: Multiple testers running with modules monitored in multiple sessions” on page 81

“Example: Multiple testers running a module, each with a unique copy” on page 82

Example: Multiple testers running modules with unique modules per session

This example describes multiple testers running the same modules, with only one session monitoring each module:

Tester	Setup	Start session	Run code
Tester 1 monitors modules A, C, and D, but running test cases that run modules A through F.	<ul style="list-style-type: none"> Listings for module A, module C, and module D Instrumented objects linked into executable modules A through F 	<ul style="list-style-type: none"> Session 1 BRKTABs from module A, module C, and module D 	In modules A through F
Tester 2 monitors modules B and E, but running test cases that run modules A through F.	<ul style="list-style-type: none"> Listings for module B and module E Instrumented objects linked into executable modules A through F 	<ul style="list-style-type: none"> Session 2 BRKTABs from module B and module E 	Same as for Tester 1
Tester 3 monitors module F, but running test cases that run modules A through F.	<ul style="list-style-type: none"> Listings for module F Instrumented objects linked into executable modules A through F 	<ul style="list-style-type: none"> Session 3 BRKTABs from module F 	Same as for Tester 1

The coverage data from session 1 will be the cumulative coverage of all three testers for modules A, C, and D. There is no way for Tester 1 to know what coverage data was caused by each tester. The same is true for the coverage data from session 2 and session 3.

Example: Multiple testers running with modules monitored in multiple sessions

This example describes multiple testers executing the same module, with multiple sessions monitoring identical modules.

Common setup is performed for modules A through F and instrumented objects are linked into executable modules A through F

Tester	Setup	Start session	Run code
Tester 1 monitors the coverage of modules A, C, and D, but running test cases that run Modules A through F.	<ul style="list-style-type: none"> Listings for modules A through F Instrumented objects linked into executable modules A through F 	<ul style="list-style-type: none"> Session 1 BRKTABs from module A, module C, and module D 	In modules A through F
Tester 2 monitors modules B, C and E, but running test cases that run modules A through F.	Same as for Tester 1	<ul style="list-style-type: none"> Session 2 BRKTABs from module B, module C, and module E 	Same as for Tester 1

Tester	Setup	Start session	Run code
Tester 3 monitors modules B and F, but running test cases that run modules A through F.	Same as for Tester 1	<ul style="list-style-type: none"> • Session 3 • BRKTABs from module B and module F 	Same as for Tester 1

Assume that the order that the sessions were started is session 1, session 2, and then session 3. Module C is being monitored in sessions 1 and 2, and module B is being monitored in sessions 2 and 3.

Any coverage in module C from any tester will usually be shown in session 1. Even though session 2 is monitoring C, the coverage data for C from session 2 will probably show no coverage. The same is true for module B, monitored by sessions 2 and 3. Coverage for B will show up in session 2.

However, if session 1 is stopped, any subsequent execution of module C now appears in the session 2 coverage data. The same will be true for module B, if session 2 is stopped but session 3 is still active.

Example: Multiple testers running a module, each with a unique copy

Each tester can ensure unique coverage data for test cases run by that tester, by using the following procedure:

Tester	Setup	Start session	Run code
Tester 1 monitors modules A through F.	<ul style="list-style-type: none"> • Listings for modules A through F • Link into modules A1 through F1 	<ul style="list-style-type: none"> • Session 1 • BRKTABs from modules A through F from Tester 1 setup 	In modules A1 through F1
Tester 2 monitors modules A through F.	<ul style="list-style-type: none"> • Listings for modules A through F • Link into modules A2 through F2 	<ul style="list-style-type: none"> • Session 2 • BRKTABs from modules A through F from Tester 2 setup 	In modules A2 through F2
Tester 3 monitors modules A through F.	<ul style="list-style-type: none"> • Listings for modules A through F • Link into modules A3 through F3 	<ul style="list-style-type: none"> • Session 3 • BRKTABs from modules A through F from Tester 3 setup 	In modules A3 through F3

Each tester performed a unique setup for the code to be monitored in modules A through F. Then the instrumented objects were linked into unique executable modules Ax through Fx. The coverage statistics represent testing done by an individual tester and no one else.

Using performance mode to reduce monitor overhead

Measuring when a conditional branch is taken requires additional overhead. If the increased overhead is unacceptable for your testing, you can turn off conditional branch coverage by setting performance mode on.

When you process C/C++ code that contains inline functions, performance mode must be on. If it is off, a message is issued and the performance mode is assumed to be on.

You can control performance mode in the following ways:

- Change a default setting. Change the performance mode default (in the setup Defaults section of Defaults) to YES. Any setup JCL that you subsequently create by the setup JCL generator will indicate that performance mode is to be used. Hence, any BRKTAB files that are created in the setup step will have a flag set to indicate that conditional branches should **not** be analyzed.
- Change the *performance_mode* flag in the setup JCL for individual modules (EQACUSET). Any BRKTAB files that are created by setup when this flag is set to Y will have a flag set to indicate that conditional branches should **not** be analyzed.

When performance mode is set on by setup, the summary report for the test run will not show conditional branch coverage. The annotation for conditional statements in an annotated listing report is also modified.

When performance mode is set off by setup, and then later enabled by EQACUOPN, summary reports and annotated listing reports will still include conditional breakpoint information, although the data might be incomplete.

Related concepts

“Suppression of conditional branch coverage with performance mode” on page 119

“Changes in annotation symbols with performance mode” on page 125

Related tasks

“Modifying your Coverage Utility defaults” on page 9

Related references

“EQACUSET” on page 70

Monitoring a program that is executing under control of the Debug Tool debugger

You can both monitor a C, C++, COBOL, or PL/I program with the Coverage Utility monitor and step through it with the Debug Tool debugger at the same time, if you do not use the Debug Tool debugger Dynamic Debug facility.

Attention: if you are running a program that contains monitor breakpoints, you must use SET DYNDEBUG OFF and SET DISASSEMBLY OFF while using the debugger.

Restrictions on monitoring programs

Coverage Utility cannot monitor programs running in the UNIX System Services environment.

For programs that run on different MVS images, you must start and stop the monitor session on the same MVS image on which monitored programs are run.

In addition, Coverage Utility imposes restrictions on the following types of programs:

- Programs that reside in read-only storage.
- Programs that affect certain system modes.

Related references

“Restrictions on programs that reside in read-only storage” on page 84

“Restrictions on system modes” on page 84

Restrictions on programs that reside in read-only storage

Coverage Utility can replace breakpoints in user programs only by using the key of the user program. Therefore, user programs that run in read-only storage cannot be run if they are instrumented with the breakpoints that Coverage Utility needs.

A reentrant program runs in read-only storage when it is run as follows:

- Loaded in a CICS region with reentrant program protection (controlled by the CICS RENTPGM=PROTECT system initialization parameter). You can test in a CICS region without reentrant program protection (RENTPGM=NOPROTECT).
- Loaded from an authorized library. Relink the program as non-reentrant, or place the program in a non-authorized library (if possible).
- Loaded from the link pack area (LPA)

Restrictions on system modes

A type 3 user SVC is used for a breakpoint. There are some system modes that cannot be in effect when this type of SVC is issued. Typically these would be used in assembler code only. If a breakpoint is hit when one of these system modes is in effect, an 0F8 system ABEND occurs.

For details about which system modes can lead to this ABEND, see the reason code list for an 0F8 system ABEND code in the *z/OS MVS System Codes* or *OS/390 MVS System Codes* manuals (except for reason code 18, AR mode, which is not a restriction with the Coverage Utility breakpoint SVCs).

Chapter 8. Monitor commands

You can use several commands to control a Coverage Utility monitor session and to display statistics. You can run these commands only while a monitor session is running:

- “EQACUOBP (Display breakpoint status)” on page 86
- “EQACUOID (Add ID)” on page 89
- “EQACUOPF (Performance mode off)” on page 90
- “EQACUOPN (Performance mode on)” on page 91
- “EQACUOQT (Quit)” on page 92
- “EQACUORE (Reset)” on page 93
- “EQACUOSA (Display statistics)” on page 94
- “EQACUOSE (Display sessions)” on page 96
- “EQACUOSL (Display listings)” on page 97
- “EQACUOSN (Snapshot)” on page 99
- “EQACUOSP (Stop)” on page 100

The command execs are shipped in a partitioned data set (PDS) named *hi_lev_qual.SEQAEXEC*. Any status messages that result from the commands are written to the data set prefix.MSGS.FILE. If this data set does not exist, it is created when a command is issued.

Related tasks

“Issuing commands”

Issuing commands

You can issue commands in the following ways:

- From the Control the Monitor panel
The main commands are listed on the panel. When you select a command, a panel is displayed that enables you to enter any command parameters.

- From the TSO command line

Some commands have optional parameters that you can use. This explicit invocation shows the method for passing parameters to the command:

```
EX 'hi_lev_qual.SEQAEXEC(cmdname)' 'parm1 parm2'
```

If the Debug Tool Utilities and Advanced Functions data sets were installed into your normal logon procedure, you can use this form to issue commands:

```
cmdname parm1 parm2
```

- From MVS BATCH using JCL

This method is useful in automating test case runs. You can embed the JCL in your batch stream. When the commands are run in batch mode, the MSGS.FILE is appended with messages from each command that is run in the job. You can view this file for problem determination. The following example shows JCL to issue commands:

1. If PROFILE NOPREFIX is set in your TSO Session, then prefix is set to your user ID. If PROFILE PREFIX is set, then prefix is set to the PROFILE PREFIX value, appended with period and your userid ID if the PREFIX value and your user ID are different.

EQACUOBP - Display breakpoint status

```
//YOUNGC JOB (12345678),
//      YOUNG,NOTIFY=YOUNG,USER=YOUNG,
//      TIME=1,MSGCLASS=H,CLASS=A,REGION=2M
//*
//* FIRST A EQACUOSA COMMAND IS RUN, WITH RESULTS IN
//* YOUNG.MSGS.FILE
//* NEXT A EQACUOSP COMMAND IS RUN
//TSOTMP EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
//SYSEXEC DD DSN=hi_lev_qual.SEQAEXEC,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
        EQACUOSA YOUNG 1 1 9999
        EQACUOSP
/*
```

To see a list of the main Coverage Utility monitor commands, select option 5 from the Debug Tool Coverage Utility panel. The resulting Control the Monitor panel, shown here, contains a list of commands that you can select.

```
----- Control the Monitor -----
Option ==>

1 Start      Create JCL to Start the Monitor
2 Stop      Stop monitor execution normally          (EQACUOSP)
3 SessDisplay Display all active sessions          (EQACUOSE)
4 Listings   Display listings                        (EQACUOSL)
5 Statistics  Display statistics                    (EQACUOSA)
6 BPDDisplay Display Breakpoint status              (EQACUOBP)
7 AddId      Specify a unique testcase id          (EQACUOID)
8 Snapshot   Take snapshot of data                 (EQACUOSN)
9 Reset      Reset all data in monitor              (EQACUORE)

10 Quit      Terminate monitor without saving breakpoint data (EQACUOQT)

Enter END to Terminate
```

Related references

- “EQACUOBP (Display breakpoint status)”
- “EQACUOID (Add ID)” on page 89
- “EQACUOQT (Quit)” on page 92
- “EQACUORE (Reset)” on page 93
- “EQACUOSA (Display statistics)” on page 94
- “EQACUOSE (Display sessions)” on page 96
- “EQACUOSL (Display listings)” on page 97
- “EQACUOSN (Snapshot)” on page 99
- “EQACUOSP (Stop)” on page 100

EQACUOBP (Display breakpoint status)

The EQACUOBP command, issued from the panel shown here, displays the status of breakpoints.


```

----- Monitor: Display Breakpoint Status -----
Command ==>

To display breakpoint status, complete the menu below and press ENTER:

Session id . . . . . YOUNG
List number . . . . . 1
PA number. . . . . 1
First breakpoint number . 1
Last breakpoint number . . 9999

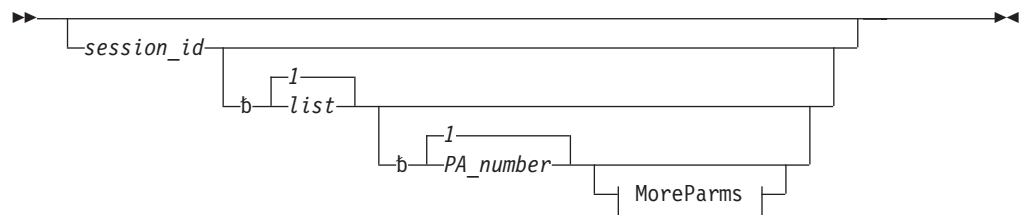
```

Important: This command can produce a large amount of data, so use it with discretion.

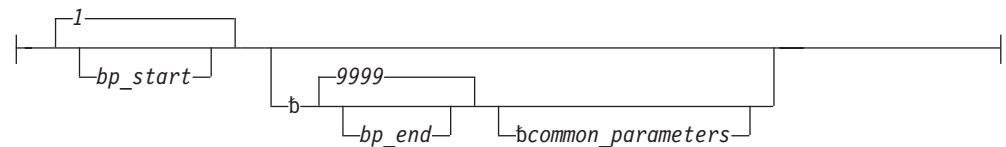
You enter this command on the command line:

```
EX 'hi_lev_qual.SEQAXEC(EQACUOBP)' 'parameters'
```

Specify *parameters* in the following syntax:



MoreParms:



where:

session_id

The ID of the session to be displayed. The default is the TSO user ID.

b One or more blanks.

list

The number of the listing; the default is 1.

PA_number

The number of the program area; the default is 1. The PA number is 1-originated for the specified listing.

bp_start

The number of the first breakpoint in the program area that you want displayed; the default is 1.

bp_end

The number of the last breakpoint in the program area that you want displayed; the default is 9999.

EQACUOBP - Display breakpoint status

common_parameters

Any of the parameters that are common to multiple routines (LINECOUNT, LOCALE, NATLANG), separated by blanks.

Example 1

To display all breakpoints in LIST 1, PA 1 of your session ID, enter this command:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOBP)'
```

Example 2

To display all breakpoints for session YOUNG in LIST 1, PA 2 with a break point number between 10 and 50, enter this command:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOBP)' 'YOUNG 1 2 10 50'
```

A status panel, such as that shown here, is displayed.

```
-----  
BROWSE      YOUNG.MSGS.FILE                               Line 00000000 Col 001 080  
Command ==>                                           Scroll ==> CSR  
*****  
***** Top of Data *****  
-----  
Num Listing                               Date Time      PAs  BPs  
-----  
001 YOUNG.SAMPLE.COBOLST(COB01A)          99.204 07:08.54 00005 000031  
PA   ADR      BPS      EVNTS      ACTIVE  
-----  
00001 00000000 000006 0000000000 000006  
RNUM OFFSET OPCD      BRNCH TO EVENTS      NB AB DE CT CF BV B> AC  
-----  
4F39 00025C D2078020 00000000 00000000 X  
419D 000262 D2048020 00000000 00000000 X  
4ED1 000268 D2028025 00000000 00000000 X  
4824 00026E D2018020 00000000 00000000 X  
F131 000274 D2028022 00000000 00000000 X  
948B 00027A D2028025 00000000 00000000 X  
***** Bottom of Data *****  
-----
```

The following fields are displayed for the selected listing and program area::

Num The sequential number of the listing.

Listing

The name of the listing data set.

Date The date of the compile.

Time The time of the compile.

PAs The number of program areas in the listing.

BPs The number of break points in the listing.

PA The sequential number of the program area.

ADR If the program has run, this is the storage address of the program area.

BPS The number of break points in this program area.

EVNTS

The number of break points that have executed for this program area.

ACTIVE

The number of break points that are still in storage in the program area.

The following fields are displayed for each breakpoint:

RNUM

A random number for this long SVC break point (0 if short SVC break point).

OFFSET

The hexadecimal offset of the breakpoint in the program area.

OPCD (op code)

The op code of the instruction at the breakpoint.

BRNCH TO (branch to)

If this instruction is a branch instruction that has branched, this address is the target address. If it starts with FF, the address is an offset within this program area.

EVENTS

The number of times that this breakpoint was executed before it was removed. For Coverage Utility, breakpoints are removed as soon as possible, so that most breakpoints are executed once only. In some cases, conditional branch breakpoints must stay in memory and are executed more than once.

NB (not a branch)

If this instruction is not a branch instruction, this entry is an X.

AB (always branch)

If this instruction is an unconditional branch instruction, this entry is an X.

DE (dummy entry)

If this is a dummy entry, this entry is an X. A dummy entry is the instruction after a conditional branch instruction that contains a breakpoint to tell when the branch falls through.

CT (condition true)

A conditional branch instruction that has branched.

CF (condition false)

A conditional branch instruction that has fallen through.

BV (conditional branch fall through)

A conditional branch that has always fallen through and never branched.

B> (conditional branch branched)

A conditional branch that has always branched and never fallen through. The breakpoint is not active. When the dummy entry after this instruction is executed, this breakpoint is updated as fallen through.

AC (active)

The breakpoint is active.

Related references

Appendix E, "Parameters that are common to multiple routines," on page 205

EQACUOID (Add ID)

Issue the EQACUOID command from the panel shown here, to add a unique test case ID.

EQACUOID - Add ID

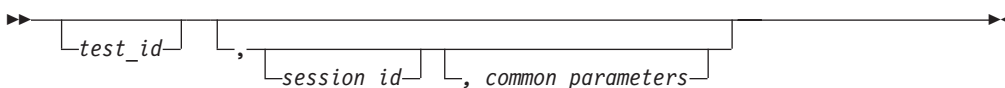
```
----- Monitor: Add ID -----  
Option ==>_  
  
To assign a test case ID, complete the menu below and press ENTER:  
  
Test Case ID . . . . .  
Session ID . . . . . YOUNG
```

The test case ID is printed in the summary report.

Enter this command on the command line:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOID)' 'parameters'
```

Specify *parameters* in the following syntax:



test_id

The assigned test ID. This ID can be from 1 to 16 characters. The default is a time stamp that consists of the date and time when the command was invoked. The test case ID is printed in the summary report.

session_id

The ID of the session for which the test case ID is to be set. The default is the TSO user ID.

A status panel, such as that shown here, is displayed.

```
-----  
BROWSE    YOUNG.MSGS.FILE                               Line 00000000 Col 001 080  
Command ==>_                                           Scroll ==> PAGE  
  
***** Top of Data *****  
The TEST ID has been set to 05/14/9713:38:38  
***** Bottom of Data *****
```

EQACUOPF (Performance mode off)

The EQACUOPF command turns the monitor performance mode off and so enables conditional branch coverage. The break points that are needed for conditional coverage are left in storage. Overhead is higher when these break points are left in storage. You can use EQACUOPF to turn off performance mode *only* if performance mode was off during setup and later turned on by the EQACUOPN command.

If you are not interested in conditional coverage, turn performance mode on, by either of the following means:

- Change the default parameter that controls performance mode before you generate the JCL for the setup step.
- Use the EQACUOPN command.

You can turn performance mode on and off for all program areas or for a selected program area.

Enter this command on the command line:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOPF)' 'parameters'
```

Specify *parameters* in the following syntax:



session_id

The ID of the session for which to turn performance mode off. The default is the TSO user ID.

PA_number

The number of the program area. The default is all program areas in the requested session.

common_parameters

Any of the parameters that are common to multiple routines, separated by commas.

Use the EQACUOSA command to get the program area number for selectively setting performance mode off.

Example 1:

To turn performance mode off for all program areas, enter this command:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOPF)'
```

Example 2:

To turn performance mode off for session YOUNG, PA 2, enter this command:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOPF)' 'YOUNG, 2'
```

Related references

Appendix E, “Parameters that are common to multiple routines,” on page 205

EQACUOPN (Performance mode on)

The EQACUOPN command turns the monitor performance mode on and so disables conditional branch coverage. The break points that are needed for conditional coverage are not left in storage. Overhead is higher when these break points are left in storage.

If you are not interested in conditional coverage, turn performance mode on, by either of the following means:

- Change the default parameter that controls performance mode before you generate the JCL for the setup step.
- Use the EQACUOPN command.

You can turn performance mode on and off for all program areas or for a selected program area.

Enter this command on the command line:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOPN)' 'parameters'
```

EQACUOPN - Perf. mode on

Specify *parameters* in the following syntax:



where:

session_id

The ID of the session for which to turn performance mode on. The default is the TSO user ID.

PA_number

The number of the program area. The default is all program areas in the requested session.

common_parameters

Any of the parameters that are common to multiple routines, separated by commas.

Use the EQACUOSA command to get the program area number for selectively setting performance mode on.

Mode flag N: If the BRKTAB for this session was built with the performance mode flag set to N (off), then the summary and annotated listing reports will still include (possibly incomplete) conditional breakpoint information after this command is issued.

Example 1:

To turn performance mode on for all program areas, enter this command:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOPN)'
```

Example 2:

To turn performance mode on for session YOUNG, PA 2, enter this command:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOPN)' 'YOUNG, 2'
```

Related references

Appendix E, "Parameters that are common to multiple routines," on page 205

EQACUOQT (Quit)

The EQACUOQT command, issued from the panel shown here, is the same as EQACUOSP, except that no output file is written (BRKOUT).

```
----- Monitor: Quit Monitor -----  
Option ==>_  
  
Quit Monitor Without Saving Breakpoint Data:  
Override default session id NO (Yes|No)
```

When you press Enter from this panel, the monitor displays the panel shown here, enabling you to enter additional parameters.

EQACUORE - Reset

Specify *parameters* in the following syntax:



session_id

The ID of the session for which to reset statistics. The default is the TSO user ID.

PA_number

The number of the program area. The default is all program area's in the requested session.

common_parameters

Any of the parameters that are common to multiple routines, separated by commas.

Use the EQACUOSA command to get the program area number for selectively resetting breakpoints.

Related references

Appendix E, "Parameters that are common to multiple routines," on page 205

EQACUOSA (Display statistics)

Issue the EQACUOSA command from the panel shown here, to select a session ID and program areas for which to display statistics.

```
----- Monitor: Display Statistics -----
Command ==>

To display PA statistics, complete the menu below and press ENTER:

  Session ID . . . . . YOUNG

  List number . . . . . 1

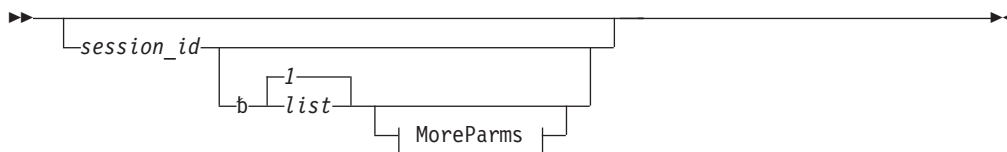
  Starting PA number . . 1

  Ending PA number . . . 9999
```

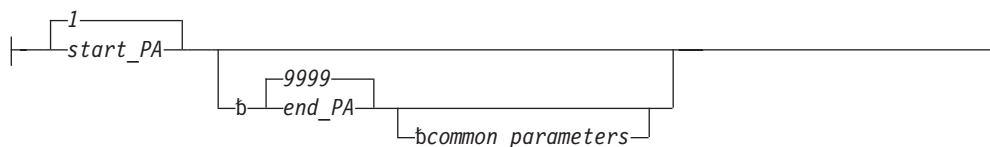
Enter this command on the command line:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOSA) 'parameters'
```

Specify *parameters* in the following syntax:



MoreParms:

*session_id*

The ID of the session for which statistics are to be displayed. The default is the TSO user ID.

b One or more blanks.

list

The listing number to display; the default is 1.

start_PA

The first program area number to be displayed in this listing; the default is 1. The start_PA number is *1-originated* for the specified listing.

end_PA

The last program area number to be displayed; the default is 9999. The end_PA number is *1-originated* for the specified listing.

common_parameters

Any of the parameters that are common to multiple routines, separated by commas.

Example 1:

To display statistics for all program areas, enter this command:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOSA)'
```

Example 2:

To display statistics for session YOUNG for LIST 1 starting with PA 1, enter this command:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOSA) 'YOUNG 1 1'
```

Example 3:

To display statistics for LIST 1 on program areas 2 through 4 of session YOUNG, enter this command:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOSA) 'YOUNG 1 2 4'
```

A statistics panel, such as that shown here, is displayed.

EQACUOSA - Display statistics

```
-----  
BROWSE    YOUNG.MSGS.FILE                               Line 00000000 Col 001 080  
Command ==>                                           Scroll ==> CSR  
***** Top of Data *****  
Num Listing                               Date Time    PAs  BPs  
-----  
001 YOUNG.SAMPLE.COBOLST(COB01A)                99.204 07:08.54 00005 000031  
PA   ADR    BPS    EVNTS    ACTVE  
-----  
00001 00000000 000006 000000000 000006  
00002 00000000 000015 000000000 000015  
00003 00000000 000001 000000000 000001  
00004 00000000 000005 000000000 000005  
00005 00000000 000004 000000000 000004  
-----  
TOTAL 00000000 000031 000000000 000031  
***** Bottom of Data *****
```

For each program area, the following fields are displayed:

Num The sequential number of the listing.

Listing

The name of the listing data set.

Date The date of the compile.

Time The time of the compile.

PAs The number of program areas in the listing.

BPs The number of break points in the listing.

PA The sequential number of the program area in the listing.

ADR When program area has run, this is the storage address of the program area.

BPS The number of break points that have been executed for the program area.

EVNTS (events)

The number of break points that have executed for this program area.

ACTVE (active)

The number of break points that are still in storage in the program area.

Related references

Appendix E, "Parameters that are common to multiple routines," on page 205

EQACUOSE (Display sessions)

The EQACUOSE command displays a list of the current active sessions. Use this command to identify session names or currently active users who need to stop or cancel their sessions before the monitor is terminated.

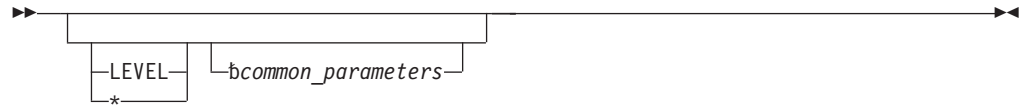
When you select option 3 (Display all active sessions) on the Control the Monitor panel, you get a response such as the following display:

```
-----  
BROWSE    YOUNG.MSGS.FILE                               Line 00000000 Col 001 080  
Command ==>                                           Scroll ==> PAGE  
***** Top of Data *****  
Session name = YOUNG    in use.  
***** Bottom of Data *****
```

Enter this command on the command line:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOSE)' 'parameters'
```

Specify *parameters* in the following syntax:

**LEVEL**

Requests that the current release of the Coverage Utility monitor be displayed. Otherwise, all active sessions will be displayed.

b One or more blanks.

common_parameters

Any of the parameters that are common to multiple routines, separated by commas.

The EQACUOSE command with the **LEVEL** parameter displays the release level and table address data used by Coverage Utility support, as this example shows:

```
-----
BROWSE    YOUNG.MSGS.FILE                               Line 00000000 Col 001 080
Command ==>                                          Scroll ==> CSR
***** Top of Data *****
Monitor Release: V7R1M0 Date: 2002.245
MAST: 00F9B270 PSA: 00FB4000 CPU: 00FB4000 SEST: 00F9BCE8 UNID: 0000001E
SESSION ID: YOUNG   PA: 01AF07D0 BP: 01AB1C38
***** Bottom of Data *****
```

Related references

Appendix E, "Parameters that are common to multiple routines," on page 205

EQACUOSL (Display listings)

Issue the EQACUOSL command from the panel shown here, to select listings for which you want to display statistics.

```
----- Monitor: Display Listings -----
Command ==>

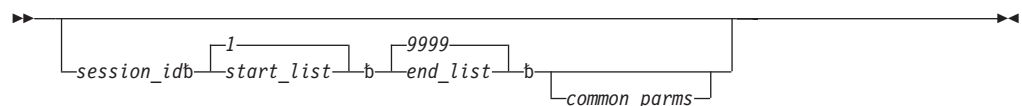
To display the listings for a session, complete the menu below and press
ENTER:

  Session ID . . . . . YOUNG
  Starting List number . 1
  Ending List number . . 9999
```

Enter this command on the command line:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOSL)' 'parameters'
```

Specify *parameters* in the following syntax:



where:

EQACUOSL - Display listings

session_id

The session for which statistics are to be displayed. The default is the TSO user ID.

b One or more blanks.

start_list

The first listing number that is to be displayed; the default is 1.

end_list

The last listing number that is to be displayed; the default is 9999.

common_parms

Any of the parameters that are common to multiple routines, separated by commas.

Example 1:

To display statistics on all listings, enter this command:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOSL)'
```

Example 2:

To display statistics for session YOUNG starting with listing 2, enter this command:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOSL)' 'YOUNG 2'
```

Example 3:

To display statistics for session YOUNG starting with listings 2 to 4, enter this command:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOSL)' 'YOUNG 2 4'
```

A statistics panel, such as that shown here, is displayed.

```
-----  
BROWSE    YOUNG.MSGS.FILE                               Line 00000000 Col 001 080  
Command ==>                                           Scroll ==> CSR  
***** Top of Data *****  
Num Listing      Date Time      PAs  BPs  
-----  
001 YOUNG.SAMPLE.COBOLST(COB01A)      99.204 07:08.54 00005 000031  
002 YOUNG.SAMPLE.COBOLST(COB01B)      99.204 07:08.56 00003 000021  
003 YOUNG.SAMPLE.COBOLST(COB01C)      99.204 07:08.59 00004 000032  
004 YOUNG.SAMPLE.COBOLST(COB01D)      99.204 07:09.01 00003 000013  
***** Bottom of Data *****
```

For each listing, the following fields are displayed:

Num The sequential number of the listing

Listing

The name of the listing data set

Date The date of the compile

Time The time of the compile

PAs The number of program areas in the listing

BPs The number of break points in the listing

Related references

Appendix E, "Parameters that are common to multiple routines," on page 205

EQACUOSN (Snapshot)

The EQACUOSN command, issued from the panel shown here, writes the current statistics (BRKOUT) to disk.

```

----- Monitor: Take Snapshot of Data -----
Command ==>

Override Default File and Session Info
Specify a test case id          NO (Yes|No)

Override default session id     NO (Yes|No)

```

When you press Enter on this panel, the monitor displays the panel shown here, enabling you to enter additional parameters.

```

----- Monitor: Take Snapshot of Data -----
Command ==>

Complete information to be overridden and press ENTER to write stats:
Specify a test case id . . . . NO (Yes|No)
Test case id. . . . .

Override default session id . . NO (Yes|No)
Session id. . . . .

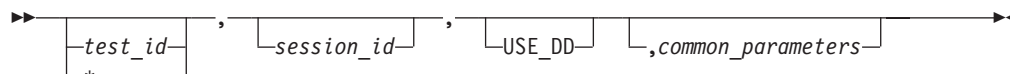
```

Statistics are *not* reset. This command enables you to take a snapshot of the current coverage activity and run a report (for example, for each test case).

Enter this command on the command line:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOSN)' 'parameters'
```

Specify *parameters* in the following syntax:



test_id

The assigned test ID. This ID can be from 1 to 16 characters. The first 8 characters of the test ID are used to build the BRKOUT data set name. If you do not provide the test ID parameter (or it is specified as an asterisk), the default time stamp (date and time) is used and the BRKOUT data set name is created by using the day of the year and the number of seconds that have elapsed since the start of the day.

session_id

The ID of the session for the snapshot. The default is the TSO user ID.

USE_DD

Indicates that a preallocated DD BRKOUT is provided for the output BRKOUT file.

common_parameters

Any of the parameters that are common to multiple routines, separated by commas.

EQACUOSN - Snapshot

Trailing commas can be omitted.

Example 1

To have the test ID default to a time stamp and the BRKOUT name use that same time stamp, enter this command:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOSN)'
```

Example 2

To add the test ID to the BRKOUT data and write the BRKOUT file to prefix.TCASE1.BRKOUT¹, enter this command:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOSN)' 'TCASE1'
```

A status panel, such as that shown here, is displayed.

```
-----  
BROWSE      YOUNG.MSGS.FILE                               Line 00000000 Col 001 080  
Command ==> _                                           Scroll ==> PAGE  
  
***** Top of Data *****  
The TEST ID has been set to 05/14/9713:38:59  
The data has been written to 'YOUNG.M134.M49139.BRKOUT'  
***** Bottom of Data *****
```

Related references

Appendix E, “Parameters that are common to multiple routines,” on page 205

EQACUOSP (Stop)

The EQACUOSP command, issued from the panel shown here, writes current statistics to disk, removes all remaining breakpoints for all program areas, and terminates the monitor session.

```
----- Monitor: Stop Monitor -----  
Command ==>  
  
Stop Monitor Normally:  
  Override default session id      NO (Yes|No)  
  
  Override breakout dsn           NO (Yes|No)
```

When you press Enter for this panel, the monitor displays the panel shown here, enabling you to enter additional parameters.

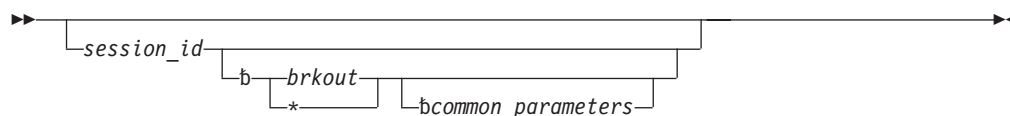
```
----- Monitor: Stop Monitor -----  
Command ==>  
  
Press ENTER to Stop Monitor Normally:  
  Override default session id      NO (Yes|No)  
  Session id . . . . .  
  
  Override breakout dsn           NO (Yes|No)  
  Breakout Dsn . . . . .
```

By default, the statistics are written to the BRKOUT data set name as supplied in the JCL that started the monitor session. You can supply a different data set name with the EQACUOSP command.

You enter this command on the command line:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOSP)' 'parameters'
```

Specify *parameters* in the following syntax:



session_id

The ID of the session to stop. The default is the current TSO user ID.

b One or more blanks.

brkout

The name of the breakout table data set. If this parameter is not specified or is specified as an asterisk, an exec named prefix.sessionid.EXTEMP.EXEC¹, which is built when the monitor session is started, is run. This exec allocates the BRKOUT file that is based on the data set names that are used when the start monitor JCL was created. Otherwise, the statistics are written to the BRKOUT file that you specify.

common_parameters

Any of the parameters common to multiple routines, separated by commas.

Example 1

To write data to the default files and set the BRKOUT session ID to the default value of the TSO user ID, enter this command:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOSP)'
```

Example 2

To write data from MySessId to prefix.TEST1.BRKOUT¹, enter this command:

```
EX 'hi_lev_qual.SEQAEXEC(EQACUOSP)' 'MySessID TEST1.BRKOUT'
```

When the stop command completes, a status panel, such as that shown here, is displayed.

```
-----
BROWSE   YOUNG.MSGS.FILE                               Line 00000000 Col 001 080
Command ==>_                                           Scroll ==> CSR

***** Top of Data *****
Monitor session YOUNG   stopped - session data written to disk
***** Bottom of Data *****
```

Related references

Appendix E, “Parameters that are common to multiple routines,” on page 205

EQACUOSP - Stop

Part 5. Obtaining Coverage Utility reports

Chapter 9. Creating reports	105	The effects of code motion	135
Creating summary report JCL by using the panels	106	The effects of dead code elimination.	135
Creating annotated listing report JCL by using the panels	107	The effects of statement decomposition.	136
Creating export JCL by using the panels	109	The effects of inlining	136
		Summary report with inline code.	137
		Annotated listing report with inline code	138
Chapter 10. Summary report.	113	Chapter 13. Report program parameters	139
Sections of the summary report	113	Parameters for the summary and report programs	139
PROGRAM AREA DATA section	114	Summary program parameters	139
UNEXECUTED CODE section.	114	Report program parameters	140
BRANCHES THAT HAVE NOT GONE BOTH		Parameters for the export data program	140
WAYS section	115	Chapter 14. HTML reports	143
Example: COBOL summary report	115	HTML Annotated Listing Report	143
Example: PL/I summary report	117	Creating an HTML Annotated Listing Report by using the panel interface	143
Example: C summary report	118	Creating an HTML Annotated Listing Report by using the command interface	144
Example: Assembler summary report	119	Restrictions on creating an HTML Annotated Listing Report	145
Suppression of conditional branch coverage with performance mode	119	Format of the HTML Annotated Listing Report	145
Example: Summary report with performance mode enabled during setup	120	Example HTML Annotated Listing Report	146
Chapter 11. Annotated listing report	123	HTML Targeted Coverage report	146
Selecting specific listings to annotate	123	Creating an HTML Targeted Coverage Report by using the panel interface	147
Reducing the size of an annotated listing report	124	Creating an HTML Targeted Coverage Report by using the command interface	148
Changes in annotation symbols with performance mode	125	Format of the HTML Targeted Coverage Report	149
Displaying execution counts in an annotated listing report	126	Restrictions on creating an HTML Targeted Coverage Report	149
Example: COBOL annotated listing report	127		
Example: PL/I annotated listing report	129		
Example: C annotated listing report	130		
Example: Assembler annotated listing report	131		
Chapter 12. Report differences for optimized C/C++ code	135		

With Coverage Utility you can generate two kinds of reports to describe test case coverage:

Summary report

A summary of the test case coverage.

Annotated listing report

Annotated compiler and assembler listings with a character on each statement that contains a breakpoint to describe how each statement was executed.

You can select the type of Coverage Utility report that you want from the Create Reports panel.

Coverage Utility can also export the coverage data to a file in XML format, which other programs can process. You can use coverage data in XML format as input to other programs to manipulate the coverage data.

| You can also use the Create Reports panel to export the coverage data in XML
| format.

Chapter 9. Creating reports

To create a report, select option 4 from the Debug Tool Coverage Utility panel. The Create Reports panel, shown here, is displayed. Use this panel to select the kind of report that you want.

```
----- Create Reports -----  
Option ==>  
  
1 Summary      Create JCL for Summary Report  
2 Annotation   Create JCL for Summary and Annotation Report  
3 Export       Create JCL for Exporting Data  
  
4 Combine      Create JCL for Combining Multiple Runs  
  
5 GAnnotation  Create HTML Annotated Listing Report  
6 GTarget      Create HTML Targeted Coverage Report  
  
Enter END to Terminate
```

The options on the panel are as follows:

Summary

Creates JCL for a summary report.

Annotation

Creates JCL for both summary and annotated listing reports.

Export Exports data from a coverage run to a file in XML format.

Combine

Creates JCL for combining multiple Coverage Utility coverage runs.

GAnnotation

Creates an HTML annotated listing report.

GTarget

Creates an HTML targeted coverage report.

- “Creating summary report JCL by using the panels”
- “Creating annotated listing report JCL by using the panels” on page 107
- “Creating export JCL by using the panels” on page 109

Creating summary report JCL by using the panels

To create a summary report, select option 1 on the Create Reports panel. Use the Create JCL for Summary report panel, shown here, to specify summary report options and parameters.

```

----- Create JCL for Summary Report -----
Option ==>

1 Generate      Generate JCL from parameters
2 Edit          Edit JCL
3 Submit        Submit JCL

Enter END to Terminate

Use Program Name for File Name YES (Yes|No) Program Name COB01

Input Files:
Breakpoint Table Dsn. 'YOUNG.SAMPLE.COB01.BRKTAB'
Breakout Dsn. . . . . 'YOUNG.SAMPLE.COB01.BRKOUT'

JCL Library and Member:
JCL Dsn . . . . . 'YOUNG.SAMPLE.JCL(TCOB01)'

Output Summary Type and File:
Type. . . . . INTERNAL (Internal|External)
Inline . . . . . N (I|N)
Report Dsn . . . . . 'YOUNG.SAMPLE.COB01.SUMMARY'
(* for default sysout class)

```

The options and fields on the panel are as follows:

Generate

Generates JCL from the parameters that you have specified on the panel.

Edit Displays an ISPF edit session where you can change existing JCL.

Submit

Submits for execution the JCL that you specify in the **JCL Dsn** field on this panel.

Use Program Name for File Name

Enter YES if you want to construct the data set names from the default high-level qualifier, the specified program name, and the default low-level qualifier for each data set.

When you press Enter, the file names on the panel are changed automatically. Coverage Utility usually constructs the data sets names by using the program name.

Program Name

The name to use for Coverage Utility files when you enter YES in the **Use Program Name for File Name** field. This name can be any valid name; it does not need to be the name of any of your programs. Names of the following form are created:

- Sequential data sets:
'proj_qual.program_name.file_type'

For example: 'YOUNG.SAMPLE.COB01.BRKTAB'

- Partitioned data sets:
'proj_qual.file_type(program_name)'

For example: 'YOUNG.SAMPLE.BRKTAB(COB01)'

Input Files

The names of the breakpoint table and breakout data sets.

JCL Dsn

The name of the JCL data set that contains the JCL for this action.

Default: If the **Use Program Name for File Name** field is set to YES, then the member name or program name qualifier of the data set will be Txxxxxxx, where xxxxxx is the last seven characters of the program name.

Type The type of summary report to be produced.

Internal

The report contains information about each program area.

External

The report contains information with all program areas combined.

This option is ignored for assembler program areas.

Inline For languages for which Coverage Utility supports optimized code, the summary processor might include or ignore counts and percentages from inline code.

I Include all lines of inline code in the summary counts and percentages.

N Do **not** include inline code in the summary counts and percentages.

Report Dsn

The name of the data set that contains the summary report.

Creating annotated listing report JCL by using the panels

To create a summary report and annotated listing report, select option 2 on the Create Reports panel. Use the Create JCL for Summary and Annotation Report panel, shown here, to specify summary report and annotated listing report options.

```

----- Create JCL for Summary and Annotation Report -----
Option ==>

1 Generate      Generate JCL from parameters
2 Edit          Edit JCL
3 Submit        Submit JCL

Enter END to Terminate

Use Program Name for File Name YES (Yes|No) Program Name COB01

Input Files:
Control File Dsn. . . 'YOUNG.SAMPLE.DTCU(COB01)'
Breakpoint Table Dsn. 'YOUNG.SAMPLE.COB01.BRKTAB'
Breakout Dsn. . . . . 'YOUNG.SAMPLE.COB01.BRKOUT'

JCL Library and Member:
JCL Dsn . . . . . 'YOUNG.SAMPLE.JCL(RCOB01)'

Output Summary Type and Annotation File:
Type. . . . . INTERNAL (Internal|External)
Inline . . . . . N (I|N)
Report Dsn . . . . . 'YOUNG.SAMPLE.COB01.REPORT'
(* for default sysout class)

```

The options and fields on the panel are as follows:

Generate

Generates JCL from the parameters that you specify on the panel.

Edit

Displays an ISPF edit session where you can change existing JCL.

Submit

Submits for execution the JCL that you specify in the **JCL Dsn** field on this panel.

Use Program Name for File Name

Enter YES if you want to construct the data set names from the default high-level qualifier, the specified program name, and the default low-level qualifier for each data set.

When you press Enter, the file names on the panel are changed automatically. Coverage Utility usually constructs the data set names by using the program name.

Program Name

The name to use for Coverage Utility files when you enter YES in the **Use Program Name for File Name** field. This name can be any valid name; it does not need to be the name of any of your programs. Names of the following form are created:

- Sequential data sets:

'proj_qual.program_name.file_type'

For example: 'YOUNG.SAMPLE.COB01.BRKTAB'

- Partitioned data sets:

'proj_qual.file_type(program_name)'

For example: 'YOUNG.SAMPLE.BRKTAB(COB01)'

Input Files

The names of the control file, breakpoint table, and breakout data sets.

JCL Dsn

The name of the JCL data set that contains the JCL for this action.

Default: If you set the **Use Program Name for File Name** field to YES, then the member name or program name qualifier of the data set will be Rxxxxxxx, where xxxxxx is the last seven characters of the program name.

Type The type of summary report to be produced.

Internal

The report contains information about each program area.

External

The report contains information with all program areas combined.

This option is ignored for assembler program areas.

Inline For languages for which Coverage Utility supports optimized code, the summary processor might include or ignore counts and percentages from inline code.

I Include all lines of inline code in the summary counts and percentages.

N Do **not** include inline code in the summary counts and percentages.

Report Dsn

The name of the data set that contains the summary and annotated listing report.

Related concepts

“The effects of inlining” on page 136

Creating export JCL by using the panels

To export coverage data to a file in XML format, select option 3 on the Create Reports panel. Use the Create JCL for Exporting Data panel, here, to specify necessary options and parameters.

```
----- Create JCL for Exporting Data -----
Option ==>

1 Generate      Generate JCL from parameters
2 Edit          Edit JCL
3 Submit        Submit JCL

Enter END to Terminate

Use Program Name for File Name YES (Yes|No) Program Name COB01

Input Files:
Breakpoint Table Dsn. 'YOUNG.SAMPLE.COB01.BRKTAB'
Breakout Dsn. . . . . 'YOUNG.SAMPLE.COB01.BRKOUT'

JCL Library and Member:
JCL Dsn . . . . . 'YOUNG.SAMPLE.JCL(MCOB01)'

Output Options and XML File:
Branch Analysis . . . NO      (Yes|No)
Frequency . . . . . NO      (Yes|No)
BP Details. . . . . NO      (Yes|No)
XML Dsn . . . . . 'YOUNG.SAMPLE.COB01.XML'
```

The options and fields on the panel are as follows:

Generate

Generates JCL from the parameters that you specify on the panel.

Edit Displays an ISPF edit session where you can change existing JCL.

Submit

Submits for execution the JCL that you specify in the **JCL Dsn** field on this panel.

Use Program Name for File Name

Enter YES if you want to construct the data set names from the default high-level qualifier, the specified program name, and the default low-level qualifier for each data set.

When you press Enter, the file names on the panel are changed automatically. Coverage Utility usually constructs the data set names by using the program name.

Program Name

The name to use for Coverage Utility files when you enter YES in the **Use Program Name for File Name** field. This name can be any valid name; it does not need to be the name of any of your programs. Names of the following form are created:

- Sequential data sets:

'proj_qual.program_name.file_type'

For example: 'YOUNG.SAMPLE.COB01.BRKTAB'

- Partitioned data sets:

'proj_qual.file_type(program_name)'

For example: 'YOUNG.SAMPLE.BRKTAB(COB01)'

Input Files

The names of the breakpoint table and breakout data sets.

JCL Dsn

The name of the JCL data set that contains the JCL for this action.

Default: If you set the **Use Program Name for File Name** field to YES, then the member name or program name qualifier of the data set will be Rxxxxxxx, where xxxxxx is the last seven characters of the program name.

Branch Analysis

Specifies whether the exported file is to contain branch analysis information.

Yes The XML file contains branch analysis information (if it was collected during the coverage run).

No Branch analysis information is excluded from the XML file.

Frequency

Specifies whether the exported file is to contain information about the number of times that a statement was executed.

Yes The XML file contains frequency information (if it was collected during the coverage run).

No Frequency information will be excluded from the XML file.

BP Details

Specifies whether the exported file is to contain detailed breakpoint information.

- Yes** The XML file contains detailed information about each breakpoint in the program.
- No** Breakpoint details will be excluded from the XML file.

XML Dsn

The name of the data set to contain the coverage data in XML format.

Chapter 10. Summary report

The summary report gives statistics about the coverage of all program areas (PAs) during the test run. The summary report is divided into these sections:

Program Area Data

Lists summary data for each program area. Lists the total number of code statements and the number that were executed. Also lists the total number of branches and the number of branches executed.

Unexecuted Code

Lists the unexecuted code statements in each program area.

Branches That Have Not Gone Both Ways

Lists the conditional branches that have not executed in both directions for each program area.

Each section of a report contains the date, time, and test case ID (if provided) of the Coverage Utility test run. You can provide the test case ID with the EQACUOID command during a monitor session. A summary of data for all program areas is displayed following the PA-specific data.

“Example: COBOL summary report” on page 115

“Example: PL/I summary report” on page 117

“Example: C summary report” on page 118

“Example: Assembler summary report” on page 119

Related concepts

Chapter 12, “Report differences for optimized C/C++ code,” on page 135

Related references

“EQACUOID (Add ID)” on page 89

Sections of the summary report

Each section of the report includes a program identification area. This information gives the load module name, procedure name, and listing name for the program.

The columns in the PROGRAM IDENTIFICATION area are:

PA The number of the program area (PA)

LOAD MOD

The name of the load module.

PROCEDURE

COBOL: Paragraph name. Section names are listed only if they contain statements outside of paragraphs.

PL/I: Procedure, ON-unit, or Begin-block name (a user-supplied label, the compiler-generated name, or a Coverage Utility-generated name).

C/C++: Function name.

ASM: CSECT name.

In cases where a CSECT is interrupted by another CSECT and then, later, the original CSECT is resumed, a new program area is generated each time when a CSECT is resumed. For example, the following code creates three program areas (C1, C2, and C1):

```

C1      CSECT
        . . .
C2      CSECT
        . . .
C1      CSECT
        . . .

```

In all cases, program areas occur in the order in which they occur in the generated assembler code.

LISTING NAME

The name of the listing . If the listing name is longer than 40 characters, only the right-most 40 characters are shown.

PROGRAM AREA DATA section

The section of the report called PROGRAM AREA DATA contains coverage statistics in addition to the program identification information.

The columns in the coverage statistics area are:

STATEMENTS: TOTAL

The statements of code for this test case run.

Assembler only: The number of executable assembler statements in the program. Data areas that occur anywhere in the listing are excluded from this count.

STATEMENTS: EXEC

The statements of code that executed.

Assembler only: The number of executable assembler statements that executed.

STATEMENTS: %

The percentage of statements that executed

BRANCHES: CPATH

The number of conditional branch paths (the number of conditional branches multiplied by 2).

BRANCHES: TAKEN

The number of conditional branch paths that executed.

BRANCHES: %

The percentage of conditional branch paths that executed.

UNEXECUTED CODE section

The section of the report called UNEXECUTED CODE contains information for unexecuted code segments in addition to the program identification information.

The columns for this area are:

start The line or statement number of the first unexecuted instruction in this unexecuted segment.

Assembler only: The offset within the program area of the first unexecuted instruction in this unexecuted segment.

end The line or statement number of the last unexecuted instruction in this unexecuted segment.

Assembler only: The offset within the program area of the last unexecuted instruction in this unexecuted segment.

The number that appears for **start** and **end** is the number that is used to identify each line or statement in the compiler listing. It is either a compiler-assigned statement number or a composite line number, depending on the compiler and, in some cases, the compiler options in effect.

BRANCHES THAT HAVE NOT GONE BOTH WAYS section

The section of the report called BRANCHES THAT HAVE NOT GONE BOTH WAYS contains the following information in addition to the program identification:

stmt The line or statement number of the conditional branch instruction that did not execute in both directions.

Assembler only: The offset within the program area of the conditional branch instruction that did not execute in both directions.

The number that appears for **stmt** is the number that is used to identify each line or statement in the compiler listing. It is either a compiler-assigned statement number or a composite line number, depending on the compiler and, in some cases, the compiler options in effect.

Example: COBOL summary report

The INTERNAL option was used to create the summary report for a set of COBOL routines shown in this example. Paragraphs and their related statistics appear on separate lines. If the EXTERNAL option had been used, statistics for all paragraphs in the object module would have been combined on one line.

This example was run with the Performance Mode flag set to N, (the default is Y).

```

***** DTCU SUMMARY:                PROGRAM AREA DATA                *****
      DATE: 07/01/2002
      TIME: 08:10:36
      TEST CASE ID:

```

PROGRAM IDENTIFICATION			STATEMENTS:			BRANCHES:			
PA	LOAD MOD	PROCEDURE	LISTING NAME	TOTAL	EXEC	%	CPATH	TAKEN	%
1	COB01	PROG	YOUNG.SAMPLE.COBOLST(COB01A)	6	6	100.0	0	0	100.0
2		PROGA		5	4	80.0	6	5	83.3
3		PROCA		1	0	0.0	0	0	100.0
4		LOOP1		3	3	100.0	2	1	50.0
5		LOOP2		2	2	100.0	2	1	50.0
6	COB01	PROGB	YOUNG.SAMPLE.COBOLST(COB01B)	6	5	83.3	6	4	66.7
7		PROCB		1	1	100.0	0	0	100.0
8		LOOP1		3	3	100.0	2	1	50.0
9	COB01	PROGC	YOUNG.SAMPLE.COBOLST(COB01C)	5	5	100.0	6	6	100.0
10		PROCC		3	2	66.7	2	1	50.0
11		LOOP1		4	3	75.0	4	2	50.0
12		LOOP2		2	2	100.0	2	1	50.0
13	COB01	PROGD	YOUNG.SAMPLE.COBOLST(COB01D)	4	0	0.0	4	0	0.0
14		PROCD		1	0	0.0	0	0	100.0
15		LOOP1		1	0	0.0	0	0	100.0
Summary for all PAs:				47	36	76.6	36	22	61.1

```

***** DTCU SUMMARY:                UNEXECUTED CODE                *****
      DATE: 07/01/2002
      TIME: 08:10:36
      TEST CASE ID:

```

PROGRAM IDENTIFICATION			start	end	start	end	start	end
PA	LOAD MOD	PROCEDURE	LISTING NAME					
2	COB01	PROGA	YOUNG.SAMPLE.COBOLST(COB01A)	58	58			
3		PROCA		68	68			
6	COB01	PROGB	YOUNG.SAMPLE.COBOLST(COB01B)	40	40			
10	COB01	PROCC	YOUNG.SAMPLE.COBOLST(COB01C)	46	46			
11		LOOP1		55	55			
13	COB01	PROGD	YOUNG.SAMPLE.COBOLST(COB01D)	32	37			
14		PROCD		41	41			
15		LOOP1		45	45			

```

***** DTCU SUMMARY:                BRANCHES THAT HAVE NOT GONE BOTH WAYS *****
      DATE: 07/01/2002
      TIME: 08:10:36
      TEST CASE ID:

```

PROGRAM IDENTIFICATION			stmt	stmt	stmt	stmt	
PA	LOAD MOD	PROCEDURE	LISTING NAME				
2	COB01	PROGA	YOUNG.SAMPLE.COBOLST(COB01A)	56			
4		LOOP1		71			
5		LOOP2		76			
6	COB01	PROGB	YOUNG.SAMPLE.COBOLST(COB01B)	34	38		
8		LOOP1		48			
10	COB01	PROCC	YOUNG.SAMPLE.COBOLST(COB01C)	44			
11		LOOP1		51	53		
12		LOOP2		59			
13	COB01	PROGD	YOUNG.SAMPLE.COBOLST(COB01D)	32	34		

Related concepts

“Suppression of conditional branch coverage with performance mode” on page 119

Example: PL/I summary report

The INTERNAL option was used to create the summary report for a set of PL/I routines shown in this example. Procedures, ON-units, Begin-blocks, and their related statistics are shown on separate lines. If the EXTERNAL option had been used, statistics for all procedures, ON-units, and Begin-blocks in the object module would have been combined on one line.

This example was run with the Performance Mode flag set to N, (the default is Y).

```
***** DTCU SUMMARY:                PROGRAM AREA DATA                *****
      DATE: 07/01/2002
      TIME: 08:11:49
      TEST CASE ID:
<--  PROGRAM IDENTIFICATION  -->
| PA LOAD MOD PROCEDURE | LISTING NAME | STATEMENTS: | BRANCHES: |
|-----|-----|-----|-----|
| TOTAL  EXEC  % | CPATH  TAKEN  % |
-----|-----|-----|-----|
1 PLI01  PLI01A  YOUNG.SAMPLE.PLILST(PLI01A)  9  9 100.0  6  5 83.3
2          PROC2A  YOUNG.SAMPLE.PLILST(PLI01A)  2  0  0.0  0  0 100.0
3 PLI01  PLI01B  YOUNG.SAMPLE.PLILST(PLI01B)  11  8  72.7  6  4 66.7
4          PROC1  YOUNG.SAMPLE.PLILST(PLI01B)  2  2 100.0  0  0 100.0
5 PLI01  PLI01C  YOUNG.SAMPLE.PLILST(PLI01C)  7  4  57.1  6  3 50.0
6          PROC1  YOUNG.SAMPLE.PLILST(PLI01C)  4  3  75.0  2  1 50.0
7 PLI01  PLI01D  YOUNG.SAMPLE.PLILST(PLI01D)  2  0  0.0  2  0  0.0
8          PROC1  YOUNG.SAMPLE.PLILST(PLI01D)  4  0  0.0  2  0  0.0
-----|-----|-----|-----|
Summary for all PAs:                41  26 63.4  24  13 54.2
```

```
***** DTCU SUMMARY:                UNEXECUTED CODE                *****
      DATE: 07/01/2002
      TIME: 08:11:49
      TEST CASE ID:
<--  PROGRAM IDENTIFICATION  -->
| PA LOAD MOD PROCEDURE | LISTING NAME | start  end  start  end  start  end |
|-----|-----|-----|-----|-----|-----|
2 PLI01  PROC2A  YOUNG.SAMPLE.PLILST(PLI01A)  17  18
3 PLI01  PLI01B  YOUNG.SAMPLE.PLILST(PLI01B)  9   9   15  16
5 PLI01  PLI01C  YOUNG.SAMPLE.PLILST(PLI01C)  6   6   10  11
6          PROC1  YOUNG.SAMPLE.PLILST(PLI01C)  16  16
7 PLI01  PLI01D  YOUNG.SAMPLE.PLILST(PLI01D)  3  11
8          PROC1  YOUNG.SAMPLE.PLILST(PLI01D)  6  10
```

```
***** DTCU SUMMARY:                BRANCHES THAT HAVE NOT GONE BOTH WAYS *****
      DATE: 07/01/2002
      TIME: 08:11:49
      TEST CASE ID:
<--  PROGRAM IDENTIFICATION  -->
| PA LOAD MOD PROCEDURE | LISTING NAME | stmt  stmt  stmt |
|-----|-----|-----|-----|
1 PLI01  PLI01A  YOUNG.SAMPLE.PLILST(PLI01A)  10
3 PLI01  PLI01B  YOUNG.SAMPLE.PLILST(PLI01B)  8   14
5 PLI01  PLI01C  YOUNG.SAMPLE.PLILST(PLI01C)  5   8   9
6          PROC1  YOUNG.SAMPLE.PLILST(PLI01C)  15
7 PLI01  PLI01D  YOUNG.SAMPLE.PLILST(PLI01D)  3
8          PROC1  YOUNG.SAMPLE.PLILST(PLI01D)  7
```

Related concepts

“Suppression of conditional branch coverage with performance mode” on page 119

Example: C summary report

The INTERNAL option was used to create the summary report for a set of C routines shown in this example. Functions and their related statistics are shown on separate lines. If the EXTERNAL option had been used, statistics for all functions in the object module would have been combined on one line.

This example was run with the Performance Mode flag set to N, (the default is Y).

```
***** DTCU SUMMARY:                PROGRAM AREA DATA                *****
      DATE: 07/01/2002
      TIME: 08:12:22
      TEST CASE ID:

<--  PROGRAM IDENTIFICATION  -->
| PA LOAD MOD PROCEDURE | LISTING NAME | STATEMENTS: | BRANCHES: |
| TOTAL EXEC % | CPATH TAKEN % |
-----
  1 C01  PROCA          YOUNG.SAMPLE.CLST(C01A)    2     0  0.0    0     0 100.0
  2      main          YOUNG.SAMPLE.CLST(C01A)   10     9  90.0   10     7  70.0
  3 C01  PROCB          YOUNG.SAMPLE.CLST(C01B)    2     2 100.0    0     0 100.0
  4      C01B          YOUNG.SAMPLE.CLST(C01B)    9     8  88.9   10     5  50.0
  5 C01  PROCC          YOUNG.SAMPLE.CLST(C01C)    4     3  75.0    2     1  50.0
  6      C01C          YOUNG.SAMPLE.CLST(C01C)    8     6  75.0    8     3  37.5
  7 C01  PROCD          YOUNG.SAMPLE.CLST(C01D)    3     0  0.0     2     0  0.0
  8      C01D          YOUNG.SAMPLE.CLST(C01D)    3     0  0.0     2     0  0.0
-----
Summary for all PAs:                41     28 68.3    34    16 47.1

***** DTCU SUMMARY:                UNEXECUTED CODE                *****
      DATE: 07/01/2002
      TIME: 08:12:22
      TEST CASE ID:

<--  PROGRAM IDENTIFICATION  -->
| PA LOAD MOD PROCEDURE | LISTING NAME | start end | start end | start end |
-----
  1 C01  PROCA          YOUNG.SAMPLE.CLST(C01A)   29  31
  2      main          YOUNG.SAMPLE.CLST(C01A)   25  25
  4 C01  C01B          YOUNG.SAMPLE.CLST(C01B)   26  27
  5 C01  PROCC          YOUNG.SAMPLE.CLST(C01C)   34  34
  6      C01C          YOUNG.SAMPLE.CLST(C01C)   21  21    25  26
  7 C01  PROCD          YOUNG.SAMPLE.CLST(C01D)   21  25
  8      C01D          YOUNG.SAMPLE.CLST(C01D)   15  19
-----

***** DTCU SUMMARY:                BRANCHES THAT HAVE NOT GONE BOTH WAYS *****
      DATE: 07/01/2002
      TIME: 08:12:22
      TEST CASE ID:

<--  PROGRAM IDENTIFICATION  -->
| PA LOAD MOD PROCEDURE | LISTING NAME | stmt  stmt  stmt  stmt  stmt |
-----
  2 C01  main          YOUNG.SAMPLE.CLST(C01A)   19    24    26
  4 C01  C01B          YOUNG.SAMPLE.CLST(C01B)   19    24    26    26
  5 C01  PROCC          YOUNG.SAMPLE.CLST(C01C)   32
  6      C01C          YOUNG.SAMPLE.CLST(C01C)   19    23    25    25
  7 C01  PROCD          YOUNG.SAMPLE.CLST(C01D)   25
  8      C01D          YOUNG.SAMPLE.CLST(C01D)   18
-----
```

Related concepts

“Suppression of conditional branch coverage with performance mode” on page 119

Example: Assembler summary report

This example was run with the Performance Mode flag set to N, (the default is Y).

```
***** DTCU SUMMARY:                PROGRAM AREA DATA                *****
      DATE: 07/01/2002
      TIME: 08:11:13
      TEST CASE ID:
<--  PROGRAM IDENTIFICATION  -->
| PA LOAD MOD PROCEDURE | LISTING NAME | STATEMENTS: | BRANCHES: |
|-----|-----|-----|-----|
| TOTAL  EXEC  % | CPATH  TAKEN  % |
-----|-----|-----|-----|
1 ASM01  TEST2      YOUNG.SAMPLE.ASMLST(ASM01A)  41   31  75.6   6   5  83.3
2 ASM01  TEST2B     YOUNG.SAMPLE.ASMLST(ASM01B)  45   39  86.7   6   4  66.7
3 ASM01  TEST2C     YOUNG.SAMPLE.ASMLST(ASM01C)  39   32  82.1   8   5  62.5
4 ASM01  TEST2D     YOUNG.SAMPLE.ASMLST(ASM01D)  25    0   0.0   4   0   0.0
-----|-----|-----|-----|
Summary for all PAs:                150   102  68.0   24   14  58.3
```

```
***** DTCU SUMMARY:                UNEXECUTED CODE                *****
      DATE: 07/01/2002
      TIME: 08:11:13
      TEST CASE ID:
<--  PROGRAM IDENTIFICATION  -->
| PA LOAD MOD PROCEDURE | LISTING NAME | start  end   start  end   start  end
|-----|-----|-----|-----|-----|-----|
1 ASM01  TEST2      YOUNG.SAMPLE.ASMLST(ASM01A) 000056 000062  000086 0000A0
2 ASM01  TEST2B     YOUNG.SAMPLE.ASMLST(ASM01B) 000050 00005A  000078 000082
3 ASM01  TEST2C     YOUNG.SAMPLE.ASMLST(ASM01C) 000034 00003A  000050 00005A  000092 000098
4 ASM01  TEST2D     YOUNG.SAMPLE.ASMLST(ASM01D) 000000 000000  000016 00006C
-----|-----|-----|-----|-----|-----|
```

```
***** DTCU SUMMARY:                BRANCHES THAT HAVE NOT GONE BOTH WAYS *****
      DATE: 07/01/2002
      TIME: 08:11:13
      TEST CASE ID:
<--  PROGRAM IDENTIFICATION  -->
| PA LOAD MOD PROCEDURE | LISTING NAME | stmt   stmt   stmt   stmt   stmt
|-----|-----|-----|-----|-----|
1 ASM01  TEST2      YOUNG.SAMPLE.ASMLST(ASM01A) 000052
2 ASM01  TEST2B     YOUNG.SAMPLE.ASMLST(ASM01B) 000040  00008C
3 ASM01  TEST2C     YOUNG.SAMPLE.ASMLST(ASM01C) 000030  000064  00008E
4 ASM01  TEST2D     YOUNG.SAMPLE.ASMLST(ASM01D) 000030  00005E
-----|-----|-----|-----|-----|
```

Suppression of conditional branch coverage with performance mode

When you enable performance mode during setup, the breakpoints for conditional branches are not implemented and the conditional branch coverage data is not collected. Therefore, the conditional branch coverage data is suppressed for summary reports: the BRANCHES section in the PROGRAM AREA DATA sections is blank, and the conditional branches are not listed in the BRANCHES THAT HAVE NOT GONE BOTH WAYS section.

If you perform the summary on a test run where some object modules are tested with performance mode enabled and some with it disabled, conditional branch coverage is listed for the object modules that are tested with performance mode disabled.

When you set performance mode off during setup, and then later enable it by the EQACUOPN command, summary reports and annotated listing reports still include conditional breakpoint information, although the data might be incomplete.

“Example: Summary report with performance mode enabled during setup”

Related tasks

“Using performance mode to reduce monitor overhead” on page 82

Example: Summary report with performance mode enabled during setup

This summary report for a set of COBOL routines was generated with the Performance Mode flag set to Y, the default. Hence it contains no statistics for branches.

***** DTCU SUMMARY: PROGRAM AREA DATA *****
 DATE: 07/01/2002
 TIME: 10:07:13

TEST CASE ID:

PROGRAM IDENTIFICATION			STATEMENTS:			BRANCHES:			
PA	LOAD MOD	PROCEDURE	LISTING NAME	TOTAL	EXEC	%	CPATH	TAKEN	%
1	COB01	PROG	YOUNG.SAMPLE.COBOLST(COB01A)	6	6	100.0			
2		PROGA		5	4	80.0			
3		PROCA		1	0	0.0			
4		LOOP1		3	3	100.0			
5		LOOP2		2	2	100.0			
6	COB01	PROGB	YOUNG.SAMPLE.COBOLST(COB01B)	6	5	83.3			
7		PROCB		1	1	100.0			
8		LOOP1		3	3	100.0			
9	COB01	PROGC	YOUNG.SAMPLE.COBOLST(COB01C)	5	5	100.0			
10		PROCC		3	2	66.7			
11		LOOP1		4	3	75.0			
12		LOOP2		2	2	100.0			
13	COB01	PROGD	YOUNG.SAMPLE.COBOLST(COB01D)	4	0	0.0			
14		PROCD		1	0	0.0			
15		LOOP1		1	0	0.0			

Summary for all PAs: 47 36 76.6 0 0 100.0

***** DTCU SUMMARY: UNEXECUTED CODE *****
 DATE: 07/01/2002
 TIME: 10:07:13

TEST CASE ID:

PROGRAM IDENTIFICATION			start		end		start		end	
PA	LOAD MOD	PROCEDURE	LISTING NAME							
2	COB01	PROGA	YOUNG.SAMPLE.COBOLST(COB01A)	58	58					
3		PROCA		68	68					
6	COB01	PROGB	YOUNG.SAMPLE.COBOLST(COB01B)	40	40					
10	COB01	PROCC	YOUNG.SAMPLE.COBOLST(COB01C)	46	46					
11		LOOP1		55	55					
13	COB01	PROGD	YOUNG.SAMPLE.COBOLST(COB01D)	32	37					
14		PROCD		41	41					
15		LOOP1		45	45					

***** DTCU SUMMARY: BRANCHES THAT HAVE NOT GONE BOTH WAYS *****
 DATE: 07/01/2002
 TIME: 10:07:13

TEST CASE ID:

PROGRAM IDENTIFICATION			stmt			
PA	LOAD MOD	PROCEDURE	LISTING NAME			

Chapter 11. Annotated listing report

You can create two kinds of annotated listing reports to show code coverage:

All Every line of the listing is printed.

Unexecuted

Only unexecuted instructions and conditional branch instructions that have not gone both ways are printed.

Each instruction line in the listing has an annotation character placed to the right of the line or statement number to indicate what happened during the test run:

& A conditional branch instruction has executed both ways.
> A conditional branch instruction has branched, but not fallen through.
V A conditional branch instruction has fallen through, but not branched.
: A non-branch instruction has executed.
¬ An instruction has not executed.
@ Data area in the assembler listing.
% Unconditional branch that has been executed in the assembler listing

These characters are the defaults. You can replace them with any others that you prefer by supplying a parameter to the report program.

You might see multiple annotations on a single source statement. Such annotations occur when there are multiple logical segments in the generated machine code for the source statement, such as a complex IF statement.

The line or statement number is the number that is used to identify each line or statement in the compiler listing. It is either a compiler-assigned statement number or a composite line number, depending on the compiler and, in some cases, the compiler options in effect.

The annotation of statements with conditional branches (&, >, V) is affected by performance mode.

“Example: COBOL annotated listing report” on page 127

“Example: PL/I annotated listing report” on page 129

“Example: C annotated listing report” on page 130

“Example: Assembler annotated listing report” on page 131

Related concepts

“Changes in annotation symbols with performance mode” on page 125

Chapter 12, “Report differences for optimized C/C++ code,” on page 135

Selecting specific listings to annotate

You can create a summary report first, and then, based on the summary, decide to produce annotated listing reports on certain program areas.

To select the specific listings that you want to annotate after completing a test run, follow these steps:

1. Edit the Coverage Utility control file, leaving in the names of the listings that you want to annotate:
 - a. Select option 1 from the Debug Tool Coverage Utility panel.

- b. On the Work with the Control File panel, select option 1, which displays an ISPF edit session. Use this edit session to modify the control file. (You can also edit the control file directly by using the ISPF editor.)
 - c. Delete the unwanted listings. The easiest way to do this is to comment out the line by putting an asterisk (*) in column 1.
2. Create the report JCL:
Select option 1 from the Create JCL for Summary and Annotation Report panel.
 3. Submit the report JCL:
Select option 3 from the Create JCL for Summary and Annotation Report panel.
Coverage Utility produces an annotated listing report for each listing file that you specify in the Coverage Utility control cards.

Related tasks

“Creating summary report JCL by using the panels” on page 106

Reducing the size of an annotated listing report

To save paper when printing an annotated listing report, you can reduce the size of the report by printing only the lines with unexecuted code or that had conditional branches that did not go both ways. In the Report Defaults section of the Edit Defaults panel, specify U (display only unexecuted code) rather than A (display all code) as the Report User Options variable, as shown at line **1**. Make this change before generating the Summary and Annotation Report JCL:

```

----- Report Defaults -----
Combined Cntl Dsn. . . . 'YOUNG.SAMPLE.CBCTL(COB01)'
  Type . . . . . CBCTL
  DSORG. . . . . PDS          (SEQ|PDS)
Combined Breakout Dsn. . . 'YOUNG.SAMPLE.COB01.CMBOUT'
  Type . . . . . CMBOUT
  DSORG. . . . . SEQ          (SEQ|PDS)
Jobcard Name . . . . . YOUNG
Jobcard Operands . . . . . (12345678),
                             YOUNG,NOTIFY=YOUNG,USER=YOUNG,
                             MSGCLASS=H,CLASS=A,REGION=32M,TIME=1

JES Control Cards. . . . .

Report File Dsn. . . . . 'YOUNG.SAMPLE.COB01.REPORT'
Summary File Dsn . . . . 'YOUNG.SAMPLE.COB01.SUMMARY'
  Report File Type . . . . REPORT
  Summary File Type. . . . SUMMARY
  DSORG. . . . . SEQ          (SEQ|PDS)
  Alloc Parms. . . . . LRECL(133) RECFM(F B A) BLKSIZE(27930)
                             TRACKS SPACE(10 10)
  DD Parms . . . . . SPACE=(TRK,(10,10)),
                             DCB=(DSORG=PS,RECFM=FBA,LRECL=133,BLKSIZE=27930)
Summary Type . . . . . INTERNAL (Internal|External)
Summary Assembler Stmts. . YES
Summary Inline . . . . . N          (I|N)
Annotation Symbols . . . . :->V%&& (* for default)
Report User Options. . . . U          (A|U) 1
Print Report File Dataset. YES (Yes|No)
Exported XML Dsn . . . . 'YOUNG.SAMPLE.COB01.XML'
  XML File Type. . . . . XML
  DSORG. . . . . SEQ          (SEQ|PDS)
  Alloc Parms. . . . . LRECL(32756) RECFM(V B) BLKSIZE(32760)
                             TRACKS SPACE(10 10)
  DD Parms . . . . . SPACE=(TRK,(10,10)),
                             DCB=(DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760)
XML Branch Analysis. . . . NO (Yes|No)
XML Frequency. . . . . NO (Yes|No)
XML BP Details . . . . . NO (Yes|No)

```

Changes in annotation symbols with performance mode

When you enable performance mode during setup, the breakpoints for conditional branches are not implemented and the conditional branch coverage data is not collected. Therefore, the annotation in the generated report changes for conditional statements such as IF, PERFORM (COBOL), DO WHILE (PL/I). Such statements that were annotated with one conditional annotation symbol (>,V,&) per conditional branch within the statement. are now annotated as follows:

- : (a colon), if the statement is executed
- One ¬ (a not symbol), if the statement is not executed

When you set performance mode off during setup, and then later enable it by the EQACUOPN command, summary reports and annotated listing reports still include conditional breakpoint information, although the data might be incomplete.

Related tasks

“Using performance mode to reduce monitor overhead” on page 82

Displaying execution counts in an annotated listing report

The number of times that each statement was executed can be displayed in an annotated listing report. To do so, change the Frequency Count Mode and Debug Mode flags in your setup defaults before you generate the JCL for the setup step.

To change your setup defaults do the following steps:

1. Select option 0 from the Debug Tool Coverage Utility panel. The Manipulate Defaults panel is displayed.
2. Select option 1. The Edit Defaults panel is displayed.
3. In the Setup Defaults area of the panel, change Frequency Count Mode and Debug Mode flags to Yes.

Every time that setup JCL is created, these flags are set. You can also change these flags in setup JCL that has already been created. This change can be simpler than changing the defaults (and then changing them back), if you want to get execution counts for just one test run. To identify these flags in the parameters passed to the setup program, see the comments in the created setup JCL.

When the Frequency Count Mode and Debug Mode flags are set to Yes, breakpoints are left in storage for the entire test run instead of being removed after their first execution. Each time that the breakpoint is executed, the count field is incremented. The execution counts are then saved in the BRKOUT file of coverage results and displayed on the right hand side of the annotated listing report, as shown here:

LineID	PL	SL	-----*A-1-B-+-----2-----3-----4-----5-----6-----7- -+-----8	Map and Cross	Reference
000058	^	1	PERFORM PROCA.		>0000<
000059					
000060					
000061	&		PERFORM LOOP2 UNTIL TAPARM2 = 0		>0001<
000062			.		
000063	:		STOP RUN		>0001<
000064			.		
000065					
000066			PROCA.		
000067	*		PROCA NOT EXECUTED		
000068	^		MOVE 10 TO P1PARM1		>0000<
000069			.		
000070			LOOP1.		
000071	V		IF TAPARM1 > 0 THEN		>0005<
000072	:	1	SUBTRACT 1 FROM TAPARM1.		>0005<
000073	:		CALL 'COB01B'		>0005<
000074			.		
000075			LOOP2.		
000076	V		IF TAPARM2 > 0 THEN		>0002<
000077	:	1	SUBTRACT 1 FROM TAPARM2.		>0002<
000078					

The execution counts for each statement are on the right between the right arrow (>) and the left arrow (<). For example, statement 72 was executed five times and statement 77 was executed two times.

Performance: Enabling Frequency Count Mode and Debug Mode will significantly degrade the performance of the monitored program.

Example: COBOL annotated listing report

This is an example of a COBOL annotated listing report in which all lines have been printed.

This example was run with the Performance Mode flag set to N.

```
LineID  PL SL  ----+*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7-|
000001          * COB01A - COBOL EXAMPLE FOR DTCU
000002
000003          IDENTIFICATION DIVISION.
000004          PROGRAM-ID. COB01A.
000005          *****
000006          * Licensed Materials - Property of IBM          *
000007          *          *          *
000008          * 5655-M18: Debug Tool for z/OS          *
000009          * 5655-M19: Debug Tool Utilities and Advanced Functions for z/OS *
000010          * (C) Copyright IBM Corp. 1997, 2004 All Rights Reserved          *
000011          *          *          *
000012          * US Government Users Restricted Rights - Use, duplication or          *
000013          * disclosure restricted by GSA ADP Schedule Contract with IBM          *
000014          * Corp.          *
000015          *          *          *
000016          *****
000017
000018          ENVIRONMENT DIVISION.
000019
000020          DATA DIVISION.
000021
000022          WORKING-STORAGE SECTION.
000023          01 TAPARM1      PIC 99 VALUE 5.
000024          01 TAPARM2      PIC 99 VALUE 2.
000025          01 COB01B      PIC X(6) VALUE 'COB01B'.
000026          01 P1PARM1     PIC 99 VALUE 0.
000027
000028          01 TASTRUCT.
000029             05 LOC-ID.
000030                 10 STATE      PIC X(2).
000031                 10 CITY       PIC X(3).
000032                 05 OP-SYS     PIC X(3).
000033
000034          PROCEDURE DIVISION.
000035
000036          * THE FOLLOWING ALWAYS PERFORMED
000037
000038          PROG.
000039          * ACCESS BY TOP LEVEL QUALIFIER
000040          * MOVE 'ILCHIMVS' TO TASTRUCT
000041 :
000042
000043          * ACCESS BY MID LEVEL QUALIFIERS
000044 :          MOVE 'ILSPR' TO LOC-ID
000045 :          MOVE 'AIX' TO OP-SYS
000046
```

```

000047          * ACCESS BY LOW LEVEL QUALIFIERS
000048 :          MOVE 'KY' TO STATE
000049 :          MOVE 'LEX' TO CITY
000050 :          MOVE 'VM ' TO OP-SYS
000051          .
000052
000053          PROGA.
000054 &          PERFORM LOOP1 UNTIL TAPARM1 = 0
000055
000056 >          IF TAPARM2 = 0 THEN
000057 *          PROCA NOT EXECUTED
000058 ↵ 1          PERFORM PROCA.
000059
000060 000061 &          PERFORM LOOP2 UNTIL TAPARM2 = 0
000062          .
000063 :          STOP RUN
000064          .
000065
000066          PROCA.
000067 *          PROCA NOT EXECUTED
000068 ↵          MOVE 10 TO P1PARM1
000069          .
000070          LOOP1.
000071 V          IF TAPARM1 > 0 THEN
000072 : 1          SUBTRACT 1 FROM TAPARM1.
000073 :          CALL 'COB01B'
000074          .
000075          LOOP2.
000076 V          IF TAPARM2 > 0 THEN
000077 : 1          SUBTRACT 1 FROM TAPARM2.
000078

```

Related concepts

“Changes in annotation symbols with performance mode” on page 125

Example: PL/I annotated listing report

This is an example of a PL/I annotated listing report in which all lines have been printed.

This example was run with the Performance Mode flag set to N.

STMT

```
1  PLI01A:PROC OPTIONS(MAIN);          /* PL/I DTCU TESTCASE          */
   /*****
   /* Licensed Materials - Property of IBM          */
   /*
   /* 5655-M18: Debug Tool for z/OS                */
   /* 5655-M19: Debug Tool Utilities and Advanced Functions for z/OS */
   /* (C) Copyright IBM Corp. 1997, 2004 All Rights Reserved          */
   /*
   /* US Government Users Restricted Rights - Use, duplication or
   /* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*/
   /*
   /*****

2  DCL EXPARM1 FIXED BIN(31) INIT(5);
3  DCL EXPARM2 FIXED BIN(31) INIT(2);
4  DCL PARM2   FIXED BIN(31) INIT(2);
5  DCL PLI01B EXTERNAL ENTRY;          /*
6& DO WHILE (EXPARM1 > 0);             /* THIS DO LOOP EXECUTED 5 TIMES*/
7:   EXPARM1 = EXPARM1 -1;             /*
8:   CALL PLI01B(PARM2);               /* PLI01B CALLED 5 TIMES          */
9: END;
10> IF (EXPARM2 = 0) THEN                /* THIS BRANCH ALWAYS TAKEN      */
   CALL PROC2A(EXPARM2);                /* PROC2A NEVER CALLED           */
11& DO WHILE (EXPARM2 > 0);             /* DO LOOP EXECUTED TWICE        */
12:   EXPARM2 = EXPARM2 - 1;
13: END;
14: RETURN;

15 PROC2A: PROCEDURE(P1PARM1);          /* THIS PROCEDURE NEVER EXECUTED */
16 DCL P1PARM1 FIXED BIN(31);
17~ P1PARM1 = 10;
18~ END PROC2A;
19 END PLI01A;
```

Related concepts

“Changes in annotation symbols with performance mode” on page 125

Example: C annotated listing report

This is an example of a C annotated listing report in which all lines have been printed.

This example was run with the Performance Mode flag set to N.

LINE	STMT	SEQNBR	INCNO
	...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...8...+...9...+...		
1	main()	1	
2	/******	2	
3	/* Licensed Materials - Property of IBM */	3	
4	/* */	4	
5	/* 5655-M18: Debug Tool for z/OS */	5	
6	/* 5655-M19: Debug Tool Utilities and Advanced Functions for z/OS */	6	
7	/* (C) Copyright IBM Corp. 1997, 2004 All Rights Reserved */	7	
8	/* */	8	
9	/* US Government Users Restricted Rights - Use, duplication or */	9	
10	/* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*/	10	
11	/* */	11	
12	/******	12	
13		13	
14	{	14	
15: 1	int EXPARM1 = 5;	15	
16: 2	int EXPARM2 = 2;	16	
17	extern void C01B(void);	17	
18	void PROCA(int); /* function not called */	18	
19V& 3	while (EXPARM1 > 0) /* execute loop 5 times */	19	
20	{	20	
21: 4	EXPARM1 = EXPARM1 -1;	21	
22: 5	C01B(); /* call C01B 5 times */	22	
23	}	23	
24> 6	if (EXPARM2 == 0) /* branch taken */	24	
25~ 7	PROCA(EXPARM2); /* not executed */	25	
26V& 8	while (EXPARM2 > 0) /* loop execute 2 times */	26	
27: 9	EXPARM2 = EXPARM2 - 1; /* executed twice */	27	
28:	}	28	
29~	void PROCA(int P1PARM1) /* function not called */	29	
30	{	30	
31~ 10	P1PARM1 = 10; /* not executed */	31	
32	}	32	

Related concepts

“Changes in annotation symbols with performance mode” on page 125

Example: Assembler annotated listing report

This is an example of an Assembler annotated listing report in which all lines have been printed.

This example was run with the Performance Mode flag set to N.

Loc	Object Code	Addr1	Addr2	Stmt	Source Statement	HLASM R4.0 2002/07/01 07.20
				1	*****	
				2	* Licensed Materials - Property of IBM	*
				3	*	*
				4	* 5655-M18: Debug Tool for z/OS	*
				5	* 5655-M19: Debug Tool Utilities and Advanced Functions for z/OS	*
				6	* (C) Copyright IBM Corp. 1997, 2004 All Rights Reserved	*
				7	*	*
				8	* US Government Users Restricted Rights - Use, duplication or	*
				9	* disclosure restricted by GSA ADP Schedule Contract with IBM	*
				10	* Corp.	*
				11	*	*
				12	*****	
				13	*	
				14	*****	
				15	*	*
				16	* DTCU ASSEMBLER TESTCASE.	*
				17	*	*
				18	*****	
000000		00000	00110	19	TEST2 CSECT ,	01S0001
000000				20	@MAINENT DS OH	01S0001
		R:F	00000	21	USING *,@15	01S0001
000000	47F0 F016		00016	22%	B @PROLOG	01S0001
000004	10			23@	DC AL1(16)	01S0001
000005	E3C5E2E3F2404040			24@	DC C'TEST2 97.295'	01S0001
				25	DROP @15	
000015	00					
000016	90EC D00C		0000C	26:@PROLOG	STM @14,@12,12(@13)	01S0001
00001A	18CF			27:	LR @12,@15	01S0001
			00000	28 @PSTART	EQU TEST2	01S0001
		R:C	00000	29	USING @PSTART,@12	01S0001
00001C	50D0 C0B0		000B0	30:	ST @13,@SA00001+4	01S0001
000020	41E0 C0AC		000AC	31:	LA @14,@SA00001	01S0001
000024	50E0 D008		00008	32:	ST @14,8(@13)	01S0001
000028	18DE			33:	LR @13,@14	01S0001
				34 *	DO WHILE(EXPARG1>0); /* THIS DO LOOP EXECUTED 5 TIMES */	
00002A	47F0 C042		00042	35%	B @DE00006	01S0006
00002E				36 @DL00006	DS OH	01S0007
				37 *	EXPARG1 = EXPARG1 - 1; /* */	
00002E	5810 C100		00100	38:	L @01,EXPARG1	01S0007
000032	0610			39:	BCTR @01,0	01S0007
000034	5010 C100		00100	40:	ST @01,EXPARG1	01S0007
				41 *	CALL TEST2B(PARG2); /* TEST2B CALLED 5 TIMES */	
000038	58F0 C0F8		000F8	42:	L @15,@CV00003	01S0008
00003C	4110 C0A4		000A4	43:	LA @01,@AL00002	01S0008
000040	05EF			44%	BALR @14,@15	01S0008
				45 *	END;	
000042	5800 C100		00100	46:@DE00006	L @00,EXPARG1	01S0009
000046	1200			47:	LTR @00,@00	01S0009
000048	4720 C02E		0002E	48&	BP @DL00006	01S0009
				49 *	IF (EXPARG2 = 0) THEN /* THIS BRANCH ALWAYS TAKEN */	
00004C	5810 C104		00104	50:	L @01,EXPARG2	01S0010
000050	1211			51:	LTR @01,@01	01S0010
000052	4770 C06C		0006C	52>	BNZ @RF00010	01S0010
				53 *	CALL PROC1(EXPARG2); /* PROC1 NEVER CALLED */	
000056	4110 C0A8		000A8	54^	LA @01,@AL00003	01S0011

```

00005A 45E0 C086          00086 55^      BAL    @14,PROC1          01S0011
00005E 47F0 C06C          0006C 56 *    DO WHILE(EXPARG2>0);          /* DO LOOP EXECUTED TWICE */
000062                    57^      B      @DE00012          01S0012
000062 5820 C104          00104 58 @DL00012 DS    OH          01S0013
000066 0620                    59 *    EXPARG2 = EXPARG2 - 1;          01S0013
000068 5020 C104          00104 60:     L      @02,EXPARG2          01S0013
000066 0620                    61:     BCTR   @02,0          01S0013
000068 5020 C104          00104 62:     ST     @02,EXPARG2          01S0013
00006C 5830 C104          00104 63 *    END;          01S0014
000070 1233                    64:@DE00012 L    @03,EXPARG2          01S0014
000072 4720 C062          00062 65:     LTR    @03,@03          01S0014
000076 1FFF                    66&     BP     @DL00012          01S0014
000078 58D0 D004          00004 67 *    RETURN CODE(0);          01S0015
00007C 58E0 D00C          0000C 68:     SLR    @15,@15          01S0015
000080 980C D014          00014 69:     L      @13,4(,@13)          01S0015
000084 07FE                    70:     L      @14,12(,@13)          01S0015
000084 07FE                    71:     LM     @00,@12,20(@13)          01S0015
000084 07FE                    72%    BR      @14          01S0015
000086 90EC D00C          0000C 73 *    END TEST2;          01S0020
00008A D203 C0F4 1000 000F4 00000 74 *PROC1:          01S0016
000090 5820 C0F4          000F4 75 *    PROCEDURE(P1PARG1);          /* THIS PROCEDURE NEVER EXECUTED */
000094 4130 000A          0000A 76^PROC1 STM    @14,@12,12(@13)          01S0016
000098 5030 2000          00000 77^     MVC    @PC00002(4),0(@01)          01S0016
00009C                    78 *    P1PARG1 = 10;          01S0018
00009C                    79^     L      @02,@PA00064          01S0018
00009C 98EC D00C          0000C 80^     LA     @03,10          01S0018
0000A0 07FE                    81^     ST     @03,P1PARG1(,@02)          01S0018
0000A2                    82 *    END PROC1;          01S0019
0000A4                    83 @EL00002 DS    OH          01S0019
0000A4                    84 @EF00002 DS    OH          01S0019
0000A4 00000108          0000C 85^@ER00002 LM    @14,@12,12(@13)          01S0019
0000A8 00000104          00000 86^     BR      @14          01S0019
0000AC                    87 @DATA    DS    OH          01S0019
0000AC                    88         DS    OF          01S0019
0000F4                    89 @AL00002 DS    0A          01S0019
0000F8                    90@         DC    A(PARG2)          01S0019
0000F8 00000000          00000 91 @AL00003 DS    0A          01S0019
000100                    92@         DC    A(EXPARG2)          01S0019
000100                    93         DS    OF          01S0019
000100                    94@@SA00001 DS    18F          01S0019
000100                    95@@PC00002 DS    1F          01S0019
000100                    96         DS    OF          01S0019
000100                    97@@CV00063 DC    V(TEST2B)          01S0019
000100                    98         LTORG          01S0019
000100                    99         DS    0D          01S0019
000104 00000002          00000 100@EXPARG1 DC    F'5'          01S0019
000108 00000002          00000 101@EXPARG2 DC    F'2'          01S0019
000110                    102@PARG2  DC    F'2'          01S0019
000110                    103 @DYN SIZE EQU    0          01S0019
000110                    104 @00     EQU    0          01S0019
000110                    105 @01     EQU    1          01S0019
000110                    106 @02     EQU    2          01S0019
000110                    107 @03     EQU    3          01S0019
000110                    108 @04     EQU    4          01S0019
000110                    109 @05     EQU    5          01S0019
000110                    110 @06     EQU    6          01S0019
000110                    111 @07     EQU    7          01S0019
000110                    112 @08     EQU    8          01S0019
000110                    113 @09     EQU    9          01S0019
000110                    114 @10     EQU    10         01S0019
000110                    115 @11     EQU    11         01S0019
000110                    116 @12     EQU    12         01S0019
000110                    117 @13     EQU    13         01S0019
000110                    118 @14     EQU    14         01S0019
000110                    119 @15     EQU    15         01S0019
000110 00000 00004          00004 120 P1PARG1 EQU    0,4,C'F'          01S0019
000110 000F4 00004          00004 121 @PA00064 EQU    @PC00002,4,C'F'          01S0019
000110 0006C                    0006C 122 @RF00010 EQU    @DE00012          01S0019
000110                    123         DS    0D          01S0019
000110                    124 @ENDDATA EQU    *          01S0019
000110                    125 @MODLEN EQU    @ENDDATA-TEST2          01S0019
000110                    126         END      ,          01S0019

```

Related concepts

“Changes in annotation symbols with performance mode” on page 125

Chapter 12. Report differences for optimized C/C++ code

Coverage Utility supports the processing of optimized code in C/C++ only. The optimization techniques that compilers use modify the generated machine code. When Coverage Utility processes this optimized code, the summary and annotated listing reports usually contain differences from the same reports run on unoptimized versions of the same code. This section describes the differences that you should expect when you process optimized code.

- “The effects of code motion”
- “The effects of dead code elimination”
- “The effects of statement decomposition” on page 136
- “The effects of inlining” on page 136

Related references

“Compiler options required by Coverage Utility” on page 65

The effects of code motion

Sometimes the compiler detects that it can generate better machine code by moving part or all of a statement to a different place in the execution flow. For example, for the following high-level statements:

```
IF A=B THEN X=55;
    ELSE Y=55;
```

the compiler might determine that it needs a constant of 55 on both the true and false paths.

The compiler might then generate the following machine code:

```
      L    R1,=F'55'
*IF A=B
      L    R0,A
      C    R0,B
      BNE FALSE
*THEN X=55;
      ST   R1,X
      B    BOTH
*ELSE Y=55;
FALSE ST   R1,Y
BOTH  ...
```

Coverage Utility uses the machine code that is generated by the compiler to associate machine code with high-level statements. When the compiler indicates that a machine instruction is part of a given statement, Coverage Utility associates that statement with the instruction. Thus, instructions that are generated from a different statement and moved due to optimization are reported as if they were generated from the associated statement.

The effects of dead code elimination

Sometimes dead code is eliminated when the compiler detects that certain executable instructions can never be reached. For example, while processing the following high-level statements, the compiler might detect that the test of $X=3$ is unnecessary and that the THEN code could never be reached:

```
X=1;
... (sequential instructions with no flow into this sequence)
G=5;
IF X=3 THEN A=C;
ELSE A=B;
```

In this case it might generate machine code such as:

```
*X=1;
  (machine code for X=1)
*... (sequential instructions with no flow into this sequence)
  (machine code for these instructions)
*G=5;
  (machine code for G=5)
*A=B;
  (machine code for A=B)
```

Note that there is no machine code for IF X=3 or for A=C.

If no machine code is detected for a statement, the statement is not annotated in the annotated listing report and is not counted in the summary report.

The effects of statement decomposition

Sometimes the compiler detects that part of the code for a statement must be located at a place other than the place where the original statement occurred. For example, while processing the following high-level statement, the compiler might generate part of the machine code for the DO statement at the DO statement and part at the END statement:

```
DO I=A TO B;
...
END;
```

The way that this machine code is distributed can be altered by optimization. For example, if the compiler can determine that A is less than B on entry to the loop, it might be able to generate all or most of the code at the bottom of the loop (the END statement). In other cases, it might generate most of the code at the DO statement.

In general, whenever the statement number changes in the machine code, Coverage Utility assumes that a segment of the code of the "new" statement is being processed.

If any segment of a statement is executed, the annotation marks the statement as executed, even when all generated instructions for the statement are not executed due to statement decomposition.

The effects of inlining

A compiler puts a procedure inline when it generates the code of a called procedure at the point of invocation, instead of generating machine instructions to invoke the code for the procedure. This change might be done for some or all of the invocations of the code. For example, when the compiler process the following procedure COMPUTE, it might decide that in places where it can determine that A=0, it will generate the code for A=1:

```
COMPUTE:
  PROCEDURE(A,B);
  IF A=0 THEN A=1;
```

```

ELSE DO;
  (a lot of instructions)
END;
END;

```

In all other instances, it might generate the machine instructions that are needed to invoke COMPUTE as out-of-line code.

When the C INLINE option or the C++ OPTIMIZE option is in effect, the compiler might put inline invocations of some or all functions or procedures, depending on the compiler options in effect and on characteristics of the source code. Whether procedures and functions are inline affects your summary and annotated listing reports.

When functions are put inline, coverage of conditional branches cannot be accurately measured. During Coverage Utility setup, if performance mode is off (that is, conditional branch coverage is requested) and functions are being put inline; the Performance Mode flag is assumed to be on, and a message is issued.

Summary report with inline code

If a C/C++ function is defined as external, an out-of-line copy of the function will always exist. If a C/C++ function is defined as static, an out-of-line copy of the function might or might not exist depending on whether the compiler puts inline all invocations of the function.

When a procedure or function is put inline, several different situations can occur depending on whether an out-of-line copy of the procedure or function is generated and on the setting of the summary INLINE option.

The following table shows the four combinations of these two possibilities:

Case	An out-of-line copy of the procedure or function exists?	Summary INLINE option is:
1	Yes	N
2	Yes	I
3	No	N
4	No	I

The following table shows what happens when a procedure or function is put inline in each case:

Result	Case 1	Case 2	Case 3	Case 4
The summary report has the name of the procedure or function	Yes	Yes	No	No
Associated statistics for the procedure or function reflect accumulated statistics for all copies of the procedure or function.	Yes	Yes	No	No
Summary statistics for procedures or functions that contain inline invocations of the procedure or function reflect additional statistics for these inline statements.	No	Yes	No	Yes

For example, suppose that a function with 20 executable statements contains inline invocations of other functions that contain a total of 40 executable statements. The summary would list a total of 60 statements for that function.

Annotated listing report with inline code

The annotated listing report for inline C/C++ functions accurately reflects the overall coverage of all copies of the function. Each source line of the function has an annotation character that summarizes the execution state of all copies of that source line.

Chapter 13. Report program parameters

You can specify parameters for the summary and report programs in your JCL to determine what information is included in the reports and how the information is presented. You can also specify parameters for the export data program to determine what information is included in the output file.

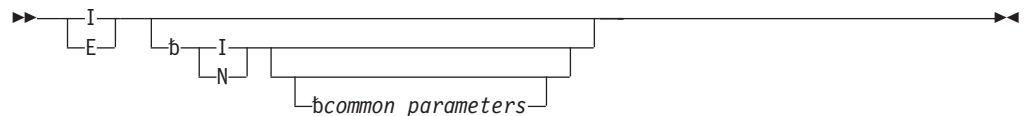
- “Parameters for the summary and report programs”
- “Parameters for the export data program” on page 140

Parameters for the summary and report programs

This section describes the input parameters that you can specify in the **PARM** field on the EXEC JCL statement for the summary and report programs.

Summary program parameters

The summary program (EQACUSUM) produces a summary report from test case results. Its parameters are built automatically by the ISPF dialog. The syntax of the parameter string is:



Internal | External

- I** Summary with each program area listed separately.
- E** Combine all program areas into one entry per object module.

Assembler: This parameter is ignored for assembler.

b Represents one or more blanks.

Inline | NoInline

- I** Include inline code in the summary statistics.
- N** Do not include inline code.

Assembler: This parameter is ignored for assembler.

common_parameters

Any of the parameters that are common to multiple routines, separated by blanks.

Related concepts

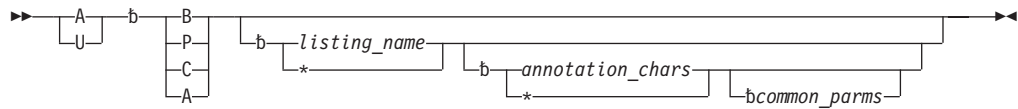
“The effects of inlining” on page 136

Related references

Appendix E, “Parameters that are common to multiple routines,” on page 205

Report program parameters

The report program (EQACURPT) produces annotated source listings from test case results. Its parameters are built automatically by the ISPF dialog. The syntax of this parameter string is:



User code to display — This flag indicates which part of the user code is displayed in the report.

- A All user code is displayed.
- U Only unexecuted user code is displayed.

b Represents one or more blanks.

Listing Type

- B COBOL listing
- P PL/I listing
- C C/C++ listing
- A assembler listing

listing_name

The fully-qualified name of the data set that contains the compiler listing. This parameter is used only for VisualAge PL/I, Version 2 Release 2; Enterprise PL/I for z/OS and OS/390; and C/C++, in which case it is required. In all other cases, it can be omitted if no following parameters are present, or it can be specified as an asterisk if any following parameters are present.

annotation_chars

List of annotation characters. If the default characters are to be used and you specify any of the following parameters, you must specify an asterisk for this parameter. The default list is as follows:

: ~ > V % @ and &

common_parms

Any of the parameters that are common to multiple routines, separated by blanks.

Related references

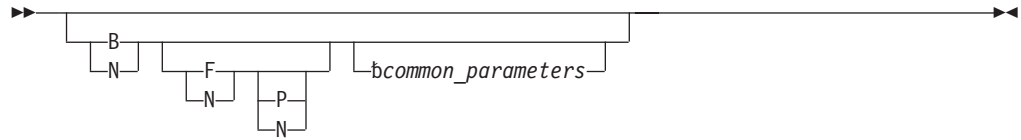
Chapter 11, "Annotated listing report," on page 123

Appendix E, "Parameters that are common to multiple routines," on page 205

Parameters for the export data program

This section describes the input parameters that you can specify in the **PARM** field on the EXEC JCL statement for the export data program.

The export data program (EQACUXML) exports the coverage data in XML format. Its parameters are built automatically by the ISPF dialog. The syntax of the parameter string is:



Branch | NoBranch

B Include branch analysis data in the output file, if the data is available.

N Do not include branch analysis data in the output file.

Frequency | NoFrequency

F Include frequency data in the output file, if the data is available.

N Do not include frequency data in the output file.

BP | NoBP

P Include detailed breakpoint data in the output file.

N Do not include detailed breakpoint data in the output file.

b Represents one or more blanks.

common_parameters

Any of the parameters that are common to multiple routines, separated by blanks.

Related references

Appendix E, "Parameters that are common to multiple routines," on page 205

Chapter 14. HTML reports

The following topics describe how you can generate code coverage reports in an HTML format.

- “HTML Annotated Listing Report”
- “HTML Targeted Coverage report” on page 146

You can display these reports on a workstation by using an internet browser. Unless otherwise noted, these HTML reports can be generated only for COBOL programs. You can generate these reports by entering commands or using the panel interfaces; both methods are described in each topic.

HTML Annotated Listing Report

The HTML Annotated Listing Report takes as input a standard Annotated Listing (described in Chapter 11, “Annotated listing report,” on page 123) and generates an HTML form of the Annotated Listing. During the process it generates a new set of statistics based strictly on the annotation symbols found in the annotated listing report that was used as input. These statistics differ in several ways from those in the original Annotated Listing Report. These differences are described in “Format of the HTML Annotated Listing Report” on page 145.

Creating an HTML Annotated Listing Report by using the panel interface

To create an HTML Annotated Listing Report by using the panel interface, do the following steps:

1. (Optional) Allocate the data set for the HTML Annotated Listing Report file, specifying RECFM=VB and LRECL>=255. If you do not allocate the data set, a data set is allocated for you.
2. Select option 5 of the Coverage Utility Create Reports panel to display the Create HTML Annotated Listing Report panel. The follow example shows the Create HTML Annotated Listing Report panel with sample input data:

```
----- Create HTML Annotated Listing Report -----
Option ==>

Input File:
  Annotated Report Dsn. 'USER1.EQACUANS.TEST.REPORT(COB019) '

Output HTML Annotated Listing File:
  Dsn . . . . . 'USER1.EQACUANS.TEST.HTML(COB019) '

Colors:
  Not Executed . . . . Pink
  Branch Not Taken . . Yellow
  Text . . . . . Black
  Background . . . . . White
  Headers . . . . . Blue
```

3. Specify the Input File as a standard Annotated Listing report as described in Chapter 11, “Annotated listing report,” on page 123.
4. Specify the output HTML annotated listing file.
5. Specify the colors that you want to use to highlight the following items:

Not Executed

The background color of lines that were not run.

Branch Not Taken

The background color of branches that were not taken both ways.

Text

The foreground color of text.

Background

The background color of normal text and headers.

Headers

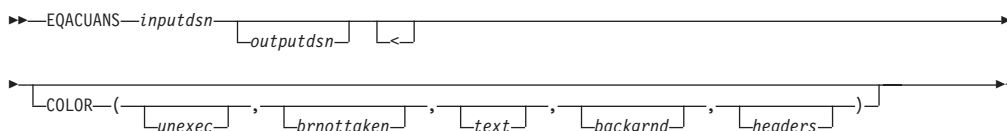
The foreground color of text for headers.

You can specify any color name that is supported by the Web browser that you are using.

Creating an HTML Annotated Listing Report by using the command interface

You can use the EQACUANS command to generate the HTML Annotated Listing Report.

The following diagram illustrates the syntax of the EQACUANS command:



The command parameters are described in the following list:

inputdsn

The DSName of an Annotated Listing Report that is used as input to the command. This must be a sequential data set or a PDS/PDSE with a member name specified.

outputdsn

The data set name of the HTML Annotated Listing Report that is to be generated. If you do not specify this parameter, a data set is generated by replacing the lower-level qualifier of *inputdsn* with HTML. If this data set does not exist, the command creates it. The required DCB attributes are RECFM=VB, LRECL>=255. This must be a sequential data set or a PDS/PDSE with a member name specified.

< A separator that is required if you do not specify an *outputdsn* variable and you specify a COLOR parameter.

unexec

The background color to use for lines of unexecuted code. You can specify any color name that is supported by a browser. The default color is pink.

brnottaken

The background color to use for lines of code containing a branch that was not taken in both directions. You can specify any color name that is supported by a browser. The default color is yellow.

text

The foreground color to use for lines of normal text. You can specify any color name that is supported by a browser. The default color is black.

| *backgrnd*

| The background color to use for normal text. You can specify any color name
| that is supported by a browser. The default color is white.

| *headers*

| The foreground color to use for headers. You can specify any color name that
| is supported by a browser. The default color is blue.

| If you specify fewer than five COLOR parameters, you can omit the commas for
| the last parameters. For example, if you only want to specify a color for the
| unexecuted lines of code, you can enter the following command:

| EQACUANS USER1.EQACUANS.TEST.REPORT(COB01) < COLOR (orange)

| **Restrictions on creating an HTML Annotated Listing Report**

| The following restrictions apply:

- | • You can use this process only on COBOL programs.
- | • This process cannot detect multiple statements on one line.
- | • If any annotations is detected on a line, this counts as one statement.
- | • If the only annotation on a line is "-", this is counted as an unexecuted
| statement.
- | • If any of the following annotations appear on a line, it is counted as an executed
| statement: ":", "&", ">", "V".
- | • Branch analysis statistics are not as correct as statistics generated by the original
| Annotated Listing Report because this process cannot recognize statements
| containing multiple branches when the annotations symbols have been
| consolidated.

| **Format of the HTML Annotated Listing Report**

| The HTML Annotated Listing Report is divided into four sections.

| The first section of the report contains the following information:

- | • Title
- | • Date and time the report was generated
- | • Test case ID, if specified

| Sections two through four contain information similar to the information in a
| Summary report described in Chapter 10, "Summary report," on page 113. The
| following list describes the differences:

- | • The statistics displayed in an HTML Annotated Listing Report are generated
| from the annotation symbols in the Annotated Listing. Therefore, the following
| items cannot be detected and might result in statistics that do not match those in
| the standard Summary report:
 - | – Multiple statements on a source line
 - | – Multiple branches within a single source line or statement
- | • Each line containing one or more annotation symbols is counted as a statement.
- | • The Unexecuted Code and Branches That Have Not Gone Both Ways sections
| contain links to the corresponding statements in the HTML Annotated Listing.

| Following this summary information are the listings from the Annotated Listing
| report. In these listings, lines containing statements that were not executed and
| branches that were not taken both ways are highlighted.

Example HTML Annotated Listing Report

The following example illustrates all four parts of an HTML Annotated Listing Report:

Debug Tool Coverage Utility
HTML Annotated Listing Report
Date: 07/11/2002 Time: 13:17:48
Test Case ID:

Coverage Utility Annotated Summary						
Program	Statements			Branches		
	Total	Exec	%	CPath	Taken	%
COB01A	17	15	88.24%	10	7	70.00%
COB01B	10	9	90.00%	8	5	62.50%
COB01C	14	12	85.71%	14	10	71.43%
COB01D	6	0	0.00%	0	0	0.00%
Total:	47	36	76.60%	32	22	0.00%

Unexecuted Code				
Program	Start-End	Start-End	Start-End	Start-End
COB01A	58	68		
COB01B	40			
COB01C	46	55		
COB01D	32-45			

Branches That Have Not Gone Both Ways									
Program	Stmt	Stmt	Stmt	Stmt	Stmt	Stmt	Stmt	Stmt	Stmt
COB01A	56	71	76						
COB01B	34	38	48						
COB01C	44	51	53	59					

HTML Targeted Coverage report

The HTML Targeted Coverage Report is an HTML file with all lines that were changed between the old and new source files highlighted.

To create an HTML Targeted Coverage Report, you must use the following items as input:

- One of the following items:
 - The DSName of the old and new source files
 - A SuperC compare listing that compares the old and new source files (command interface only)
- One of the following items:
 - A standard Annotated Listing, as described in Chapter 11, “Annotated listing report,” on page 123
 - An HTML Targeted Coverage report previously generated by this program

- a COBOL program ID.

If the standard Annotated Listing report contains multiple COBOL compiler listings and you run this program once, only one listing is updated. You will have to run this program multiple times to update all of the listings. In this situation, do the following steps:

1. Run this program the first time using the standard Annotated Listing as input.
2. Run this program the second (and subsequent) time using the HTML Targeted Coverage report generated by the previous run as input, providing a different set of source files and program ID.

Creating an HTML Targeted Coverage Report by using the panel interface

To create an HTML Targeted Coverage Report by using the panel interface:

1. (Optional) Allocate the data set for the output HTML Targeted Coverage file, specifying RECFM=VB and LRECL>=255. If you do not allocate the data set, a data set is allocated for you.
2. Select Option 6 of the Coverage Utility Create Reports panel to get to the Create HTML Targeted Coverage Report panel, which looks like the following illustration:

```

----- Create HTML Targeted Coverage Report -----
Option ==>

Input:
Old COBOL source Dsn. 'USER1.TCO.COBOL(COB01A)'
New COBOL source Dsn. 'USER1.TCN.COBOL(COB01A)'
Annotated Report Dsn. 'USER1.EQACUANS.TEST.REPORT(COB01)'
COBOL Program-ID. . .

Output HTML Targeted Coverage File:
Dsn . . . . . 'USER1.TARG.HTML(COB01)'

Colors:
Executed . . . . . Yellow
Not Executed . . . . . Pink
Text . . . . . Black
Background . . . . . White

```

3. Specify the old COBOL source. This is the COBOL source file before changes were made.
4. Specify the new COBOL source. This is the COBOL source file after changes were made and from which the Annotated Report was generated.
5. Specify the Annotated Report. This is the Annotated Report file as described in Chapter 11, “Annotated listing report,” on page 123.
6. Specify the COBOL program ID. This is the COBOL Program-ID of the program contained in the old and new source files.
7. Specify the output HTML targeted coverage file.
8. Specify the colors you want to use to highlight the following items:

Executed

The background color of changed lines that were run.

Not Executed

The background color of changed lines that were not run.

Text

The foreground color of text.

Background

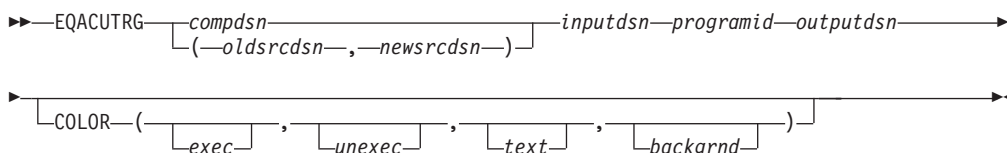
The background color of normal text and headers.

You can specify any color that is supported by the internet browser you are using.

Creating an HTML Targeted Coverage Report by using the command interface

You can use the EQACUTRG command to generate the HTML Targeted Coverage Report.

The following diagram illustrates the syntax of the EQACUTRG command:



The command parameters are described in the following list:

compdsn

DSName of the output data set generated by a SUPERC compare of the old and new source files. This must be a sequential dataset or a PDS / PDSE with a member name specified.

oldsrcdsn

DSName of the old source file to be compared.. This must be a sequential dataset or a PDS / PDSE with a member name specified.

newsrcdsn

DSName of the new source file to be compared.. This must be a sequential dataset or a PDS / PDSE with a member name specified.

inputdsn

DSName of Annotated Listing Report that is used as input to this command. This must be a sequential dataset or a PDS / PDSE with a member name specified.

programid

the COBOL PROGRAM-ID of the listing in the inputdsn file that corresponds to the source files being compared.

outputdsn

DSName of HTML Targeted Coverage Report that is to be generated. If this data set does not exist, it will be created by this command. The required DCB attributes are RECFM=VB, LRECL>=255. This must be a sequential dataset or a PDS / PDSE with a member name specified.

exec

The background color for lines of executed code. You can specify any color name that is supported by a browser. The default color is pink.

unexec

The background color for lines of unexecuted code. You can specify any color name that is supported by a browser. The default color is yellow.

text

The foreground color for lines of normal text. You can specify any color name that is supported by a browser. The default color is black.

| *backgrnd*

| The background color for lines of normal text. You can specify any color name
| that is supported by a browser. The default color is white.

| **Format of the HTML Targeted Coverage Report**

| The output of this process is an HTML file with the same contents as the input file
| except that lines that were changed between the old and new source file are
| highlighted.

| **Restrictions on creating an HTML Targeted Coverage Report**

| The following restrictions apply:

- | • This process supports COBOL programs only.
- | • If *compdsn* is supplied, the SUPERC should be done using the following options:
| LINECMP NOSUMS DLREFM NOPRTCC.
- | • Each invocation of this command processes a single compiler listing in the input
| file. If the input file consists of more than one listing and you would like to
| process more than one of the listings, you can run EQACUTRG multiple times.
| Each time after the first you must use the output annotated listing from the
| previous run as the input annotated listing for the next run.
- | • This process does not mark any lines when a "pure" delete is found.
- | • If a block of code is modified and there is more than one occurrence of these
| lines in the source code, all occurrences of these lines are marked and a message
| is issued indicating that an ambiguous change was detected.
- | • This process cannot detect changes in COPY books. Therefore, such changes are
| not marked.

Part 6. Dealing with special situations

Chapter 15. Using Coverage Utility in a project environment	153
Creating Coverage Utility files during code development	153
For the coder	153
For the tester	154
Combining test case coverage results	155
Creating the combine JCL by using the panels	156
Rules for combining results.	157
Measuring coverage for individual test cases	158

Chapter 16. Diagnosing monitor problems.	159
Solving system 047 abend	159
Solving system 7C1 abend in a user program.	159
Solving protection exception 0C4 (reason code 4) in a user program.	160
Solving system 0F8 abend in a user program.	160
Solving system Fnn abend in a user program.	160
Solving lack of SQA or ECSA space	161
Solving poor performance when measuring conditional branch coverage	161

Use the information in this part to:

- Incorporate into your standard development procedure the creation of the listings and modified object modules used by Coverage Utility.
- Solve system abends and space and performance problems.

These techniques help you use Coverage Utility in a complex testing environment, such as large projects with many programmers and testers.

Chapter 15. Using Coverage Utility in a project environment

Typically, large projects involve the following factors:

- Many code developers and testers.
- The compilation of continually changing application program modules over an extended period of time (a process which produces the listings needed by Coverage Utility).
- A testing interval that can last many days or weeks.

Use this section to learn how to incorporate the creation of files needed by Coverage Utility and the combining of results into your existing procedures. You can generate breakpoint data for an object module every time that you compile the module or for any number of listings at the same time. You can then create a summary and annotated listing report for selected modules.

- “Creating Coverage Utility files during code development”
- “Combining test case coverage results” on page 155
- “Measuring coverage for individual test cases” on page 158

Creating Coverage Utility files during code development

To enable the monitoring of code coverage for your projects, incorporate into your standard development procedure the creation of the listings and modified object modules used by Coverage Utility.

In this section the following terms are used as noted:

Coder A person who develops multiple compilable object modules for a product.

Tester A person who runs test cases on the product to obtain test case coverage data.

Module

A separately compilable object module of the project that has a listing.

To create Coverage Utility files during code development, the coder and tester follow these steps.

For the coder

Each coder generates breakpoint data on object modules when the modules are ready to be tested.

1. Create the breakpoint data automatically by including a Coverage Utility setup step (EQACUSET program) in your compile procedure or JCL, which creates a breakpoint table to match the object module.
2. In the Coverage Utility step, include a EQACUZPT program that uses the breakpoint data and modifies the object module by inserting the breakpoints.

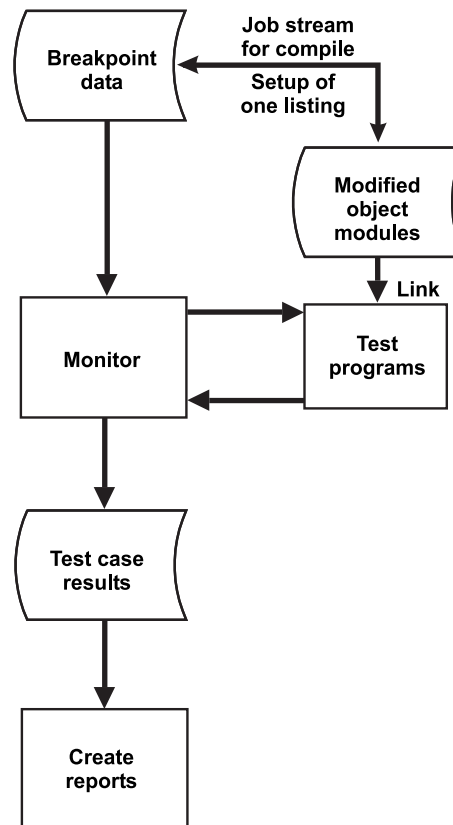
Adding these Coverage Utility steps keeps the breakpoint data current with your development.

For the tester

1. Select the object modules to be measured by Coverage Utility (created by the coder). Link them with the remaining *unmodified* object modules to create the executable load module.
2. In the Coverage Utility control cards, enter the names of the modules that are to be measured by Coverage Utility.
3. Update the start monitor JCL, by concatenating the breakpoint files as DD statements for use by the monitor during the test case coverage run.
4. Start a monitor session and run the test cases.
5. Stop the monitor session. When the monitor is stopped, it writes the test case coverage results for each module to a test case results (BRKOUT) file.
6. Create summary and annotated listing reports for selected modules from the test case coverage results, by putting the names of the modules into the Coverage Utility control cards. These names can be any modules that you have tested in any test coverage run, as long as the results are in a BRKOUT file.
7. Perform steps 1 through 6 for as many coders or test case coverage runs as desired.

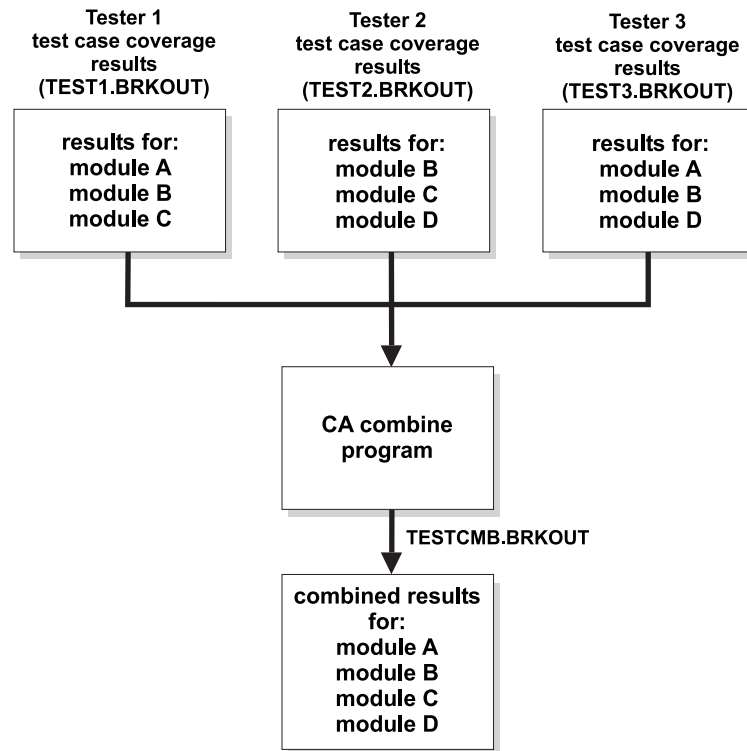
For overall test case coverage, the tester combines results from each coder or test case coverage run into one report.

A flow diagram of test case coverage in a large project environment is shown here:



Combining test case coverage results

To combine individual test case coverage runs into a report that shows overall test case coverage, run the Coverage Utility test case coverage combine program. This diagram is an example of combining test case results:



The diagram shows three test case coverage runs that were made, each with a different set of test cases:

- Tester 1 measured modules A, B, and C.
- Tester 2 measured modules B, C, and D.
- Tester 3 measured modules A, B, and D.

The test case results were put into different BRKOUT files.

To combine the results from multiple test case coverage runs into one file, run the Coverage Utility combine program. The combined results show the overall test case coverage. You can then produce a summary or annotated listing report using the combined BRKOUT file.

The BRKTAB files used for the reports must include all modules that have coverage in BRKOUT (modules A through D in the diagram). If the BRKTAB files for the modules exist in different files, concatenate them on the BRKTAB DD statement in the report JCL.

Creating the combine JCL by using the panels

To combine test case coverage results using the panels, follow these steps:

1. Select option 4 from the Debug Tool Coverage Utility panel to display the Create Reports panel.
2. On that panel select option 4 to display the Create JCL for Combining Multiple Runs panel, shown here:

```
----- Create JCL for Combining Multiple Runs -----
Option ==>

1 EditCtl      Edit Combined Control File
2 ResetCtl     Reset Combined Control File
3 Generate     Generate JCL from parameters
4 Edit         Edit JCL
5 Submit       Submit JCL

Enter END to Terminate

Use Program Name for File Name YES (Yes|No) Program Name COB06M

Combined Control File:
  Combined Cntl Dsn . . 'YOUNG.SAMPLE.CBCTL(COB06M) '

JCL Library and Member:
  JCL Dsn . . . . . 'YOUNG.SAMPLE.JCL(CCOB06M) '

Combined Breakout File:
  Breakout Dsn. . . . . 'YOUNG.SAMPLE.COB06M.CMBOUT'
```

3. Select option 1 (EditCtl) and change the values entered in each field as appropriate.

The options and fields on the panel are as follows:

EditCtl

Edit the combined control file.

ResetCtl

Reset the combined control file.

Generate

Generate JCL using the parameters that you have specified on the panel.

Edit Display an ISPF edit session where you can change existing JCL.

Submit

Submit for execution the JCL that you specify in the **JCL Dsn** field on this panel.

Use Program Name for File Name

Enter YES if you want to construct the data set names from the default high-level qualifier, the specified program name, and the default low-level qualifier for each data set. Using the program name is the normal Coverage Utility procedure.

When you press Enter, the file names on the panel are changed automatically.

Program Name

The name to use for Coverage Utility data sets when you enter YES in the **Use Program Name for File Name** field. This name can be any valid name; it does not need to be the name of any of your programs.

Names of the following forms are created:

- Sequential data sets:
'proj_qual.program_name.file_type'

For example: 'YOUNG.SAMPLE.COB01.BRKTAB'

- Partitioned data sets:
'proj_qual.file_type(program_name)'

For example: 'YOUNG.SAMPLE.BRKTAB(COB01)'

Combined Cntl Dsn

The name of the data set that contains the list of breakout data sets that you want to combine.

JCL Dsn

The name of the JCL data set that contains the JCL for this action.

Default: If you set **Use Program Name for File Name** to YES, then the member name or program name qualifier of the data set will be *Cxxxxxxx*, where *xxxxxxx* is the last seven characters of the program name.

Breakout Dsn

The name of the combined BRKOUT data set that is created by running the combine JCL.

4. The EditCtl option puts you into an ISPF edit session where you list the data sets to be combined. You must enter the complete data set name for each data set. This is an example:

```
EDIT      YOUNG.SAMPLE.CBCTL(COB06M) - 01.00          Columns 00001 00072
Command ==>                                         Scroll ==> CSR
***** ***** Top of Data *****
000001 *
000002 * List the input files (PDS or SEQ) you wish to combine (one per line),
000003 * using JCL naming conventions. All comment lines must start with '*'.
000004 *
000005 YOUNG.SAMPLE.BRKOUT(COB06M1)                  <= Input DSN 1
000006 YOUNG.SAMPLE.BRKOUT(COB06M3)                  <= Input DSN 2
000007 YOUNG.SAMPLE.BRKOUT(COB06M5)                  <= Input DSN 3
000008 YOUNG.SAMPLE.BRKOUT(COB06M7)                  <= Input DSN 4
000009                                               <= Input DSN 5
000010                                               <= Input DSN 6
000011                                               <= Input DSN 7
000012                                               <= Input DSN 8
000013                                               <= Input DSN 9
000014                                               <= Input DSN 10
***** ***** Bottom of Data *****
```

Rules for combining results

The results of two coverage runs can be combined only if there were no changes to the program between the coverage runs. The combine program checks to ensure that each program area contains the same number of breakpoints (statements) in the BRKTAB file. The coverage runs from the following steps are combined:

1. Run setup, which creates BRKTAB file.
2. Run coverage run 1, which uses BRKTAB file.
3. Run coverage run 2, which uses same BRKTAB file.

If you rerun setup without changing the program (it contains the same number of statements), the combine program combines the results.

The following coverage runs from these steps are not be combined, because the program has changed (it contains a different number of statements).

1. Run setup, which creates BRKTAB1 file.
2. Run coverage run 1, which uses BRKTAB1 file.
3. Modify the program.
4. Rerun setup, which creates a new BRKTAB2 file.
5. Run coverage run 2, which uses BRKTAB2 file.

Measuring coverage for individual test cases

As a tester, you might want to keep coverage results on a test case basis. You can then run test cases for regression test fixes that affect only a few modules instead of running all of the test cases.

Test case coverage results are saved in a BRKOUT file if the monitor is running **and** you issue either the EQACUOSN or EQACUOSP command. When you run these commands, you can select the data set name of the BRKOUT file.

For example, if you wanted to measure coverage for each of three test cases, you could do the following procedure:

1. Start the monitor by issuing XTEST2.
2. Run TEST2 for test case 1 by issuing TEST2 parm1
3. Save the coverage results in a file called *prefix*.TC1.BRKOUT¹ by issuing EQACUOSN TC1.
4. Reset the statistics by issuing EQACUORE.
5. Run TEST2 for test case 2 by issuing TEST2 parm2.
6. Save the coverage results in a file called *prefix*.TC2.BRKOUT¹ by issuing EQACUOSN TC2.
7. Reset the statistics by issuing EQACUORE.
8. Run TEST2 for test case 3 by issuing TEST2 parm3.
9. Save the coverage results in a file called *prefix*.TC3.BRKOUT¹ and stop the monitor session by issuing EQACUOSP TC3.

The BRKOUT files (TC1, TC2, and TC3 in the example) contain coverage results for their respective test cases. You can run the Coverage Utility report program on the individual BRKOUT files to obtain coverage results for the specific test cases. To obtain overall coverage, run the combine program on all of the BRKOUT files (TC1, TC2, and TC3 in the example).

Chapter 16. Diagnosing monitor problems

This section describes how you can solve the following problems:

- “Solving system 047 abend” when you try to start a monitor session
- “Solving system 7C1 abend in a user program” in a user program during a test case run
- “Solving protection exception 0C4 (reason code 4) in a user program” on page 160 in a user program during a test case run
- “Solving system 0F8 abend in a user program” on page 160 in a user program during a test case run
- “Solving system Fnn abend in a user program” on page 160 in a user program during a test case run
- “Solving lack of SQA or ECSA space” on page 161
- “Solving poor performance when measuring conditional branch coverage” on page 161 when measuring conditional-branch coverage

Solving system 047 abend

If this abend occurs when you submit the start monitor JCL or when you use the monitor interface commands, the command handler program was not in an authorized data set. Identify the Coverage Utility program that must meet this requirement, and have the data set authorized.

Related references

Debug Tool Customization Guide

Solving system 7C1 abend in a user program

If you run an instrumented user program and have not started a session to handle it, or if you start the session with a BRKTAB file that does not match the executable program, the monitor cannot identify the user supervisor call (SVC) that is installed as the breakpoint. When this problem occurs, the program terminates with a system abend 7C1, reason code 1.

The monitor might not recognize the user SVC as a breakpoint when one of these situations occurs:

- The listing that you used during setup does not match the code in the module being tested.
- You changed the program but did not rerun the setup step.
- You did not relink your program with the modified object modules after you ran the setup step.
- A monitor session that matches the code in the module being tested is not running.
- You have not stopped and restarted the monitor session after the setup step has completed.

Correct all the situations that apply.

Other reason codes that you might receive for the 7C1 abend are:

- 2 Coverage Utility control block error. Reinstall the Coverage Utility monitor SVCs.
- 3 Coverage Utility control block error. Reinstall the Coverage Utility monitor SVCs.
- 4 Cannot get dynamic storage for the user. Rerun the program that is being tested. If this error persists, reinstall the Coverage Utility monitor SVCs.
- 5 Cannot get dynamic storage for the user. Rerun the program that is being tested. If this error persists, reinstall the Coverage Utility monitor SVCs.
- 6 Cannot get a lock to update the coverage data. Rerun the program that is being tested. If the error persists, stop all monitor sessions, and then start them again.

Related references

Debug Tool Customization Guide

Solving protection exception 0C4 (reason code 4) in a user program

When a breakpoint in an instrumented program is hit, it is replaced with the real instruction and the program continues. Because this replacement is done by using the user storage key, the user program must not be in read-only storage. If an update is attempted to read-only storage, the user program terminates with an 0C4 system abend. Under CICS, this abend will be 0C4/AKEA, which is reflected to you as an ASRA transaction abend.

Place the program in a non-authorized library for testing. For CICS programs, test in a CICS region without reentrant program protection (use RENTPGM=NOPROTECT).

Related references

“Restrictions on programs that reside in read-only storage” on page 84

Solving system 0F8 abend in a user program

A type 3 user SVC is used for a breakpoint. There are some system modes that cannot be in effect when this type of SVC is issued. Typically these modes are used in assembler code only. If a breakpoint is hit when one of these system modes is in effect, an 0F8 system abend occurs.

For details on which system modes can lead to this abend, see the reason code list for an 0F8 system abend code.

Exception: Reason code 18, AR mode is not a restriction with the Coverage Utility breakpoint SVCs.

Related references

z/OS MVS System Codes

OS/390 MVS System Codes

Solving system Fnn abend in a user program

If the monitor is not installed and an instrumented program is run, the user program terminates with a system abend code of Fnn, where nn is the SVC number used as the breakpoint.

Install the monitor before attempting to run instrumented programs.

Solving lack of SQA or ECSA space

The monitor is loaded in SQA and ECSA storage and allocates ECSA storage for the session tables. If you encounter ECSA storage limitations, contact your systems programmer to see if ECSA storage can be increased.

Because of ECSA storage fragmentation, there might be enough free storage available, but not in a segment of sufficient size to load the tables. When a request for ECSA storage is made to the operating system, it fails if there is no free segment large enough to honor the request.

Related references

“Monitor CSA, SQA, and ECSA usage” on page 188

Solving poor performance when measuring conditional branch coverage

Measuring coverage when a conditional instruction branches takes more overhead than unconditional instructions (the breakpoint at the conditional branch must be left in storage). If the increased overhead is unacceptable for your testing, you can turn off conditional branch coverage.

Related tasks

“Using performance mode to reduce monitor overhead” on page 82

Part 7. Appendixes

Appendix A. Messages

This appendix describes error messages that you might receive during Coverage Utility installation and operation.

Messages that are noted as not being translated are messages that are issued during the monitor SVC installation or by the monitor SVCs. These messages always appear in English regardless of the NATLANG specification.

EQACU001S Combine: Error opening input file: *dsn*.

Explanation: Combine could not open the specified data set.

Action: Combine terminates.

User response: Ensure that the data set exists and is available.

EQACU002S Combine: Error opening output file: *dsn*.

Explanation: Combine could not open the specified data set.

Action: Combine terminates.

User response: Ensure that the data set is allocated and can be accessed.

EQACU003S Combine: Insufficient storage to satisfy request.

Explanation: The Coverage Utility combine program got an error code from the operating system command used to request storage for the BRKOUT table. Thirty two bytes are needed for each breakpoint (plus 64 bytes per PA and 32 bytes per logical file number).

Action: Combine terminates.

User response: Increase the region size on your job card for the step. Coverage Utility obtains job card information from the user defaults.

EQACU004S Combine: Input file 1 is not a BRKOUT or BRKTAB file.

Explanation: Input file 1 is not a BRKOUT or BRKTAB file.

Action: Combine terminates.

User response: Provide a valid BRKOUT or BRKTAB file for input file 1.

EQACU005S Combine: Input file 2 is not a BRKOUT or BRKTAB file.

Explanation: Input file 2 is not a BRKOUT or BRKTAB file.

Action: Combine terminates.

User response: Provide valid BRKOUT or BRKTAB file as input file 2.

EQACU006S Combine: Input files do not match. One is a BRKTAB and one is a BRKOUT.

Explanation: The input files are of a matching type (either both BRKTAB or both BRKOUT).

Action: Combine terminates.

User response: Provide input files of matching type (BRKOUT or BRKTAB).

EQACU007S Combine: Invalid keyword or value *keyword*

Explanation: An invalid keyword or keyword operand was detected.

Action: Combine terminates.

User response: Correct the specified keyword or operand.

EQACU008S Combine: Unsupported value specified in parameter *parm*

Explanation: An invalid value was specified for the indicated parameter.

Action: Combine terminates.

User response: Correct the specified value.

EQACU009W Combine: Both input files are empty. Empty output file is created.

Explanation: Both input files to combine (INPUT1 and INPUT2) are empty.

Action: Combine generates an empty output data set.

User response: Provide valid BRKTAB or BRKOUT files.

EQACU010W Combine: Input file *inputx* is empty. Other input file is copied to output file.

EQACU011S • EQACU020S

Explanation: The specified combine input file is empty. The other input file appears to contain a valid BRKTAB or BRKOUT file.

Action: The valid input file is copied to the output data set.

User response: Provide valid BRKTAB or BRKOUT files.

EQACU011S Summary: Invalid keyword or value *keyword*.

Explanation: The parameter passed to summary is not valid.

Action: Summary terminates.

User response: Correct keyword or value.

Related references

“Summary program parameters” on page 139

EQACU012S Summary: EOF of BRKTAB before finding the *testid* test case.

Explanation: The *testid* test case was not found in the BRKTAB file provided.

Action: Summary terminates.

User response: Ensure that the BRKTAB file in the BRKTAB DD statement of the JCL is the same one that was used when you ran the monitor session that created the BRKOUT file in the BRKOUT DD statement. If you have run setup to create a new BRKTAB file since creating the BRKOUT file, the BRKOUT and the BRKTAB files will not match.

EQACU013S Summary: Invalid file name *file_name* cannot be opened.

Explanation: The file *file_name* cannot be opened.

Action: Summary terminates.

User response: Ensure that the DDNAME *file_name* is specified in the JCL, that the file it points to exists and is available to the summary job, and that the job that created the file completed successfully.

EQACU014S Summary: Invalid BRKTAB file.

Explanation: The BRKTAB file is not valid.

Action: Summary terminates.

User response: Ensure that you are passing the correct BRKTAB file in the BRKTAB DD statement of the JCL.

EQACU015S Summary: Invalid BRKOUT file.

Explanation: The BRKOUT file is not valid.

Action: Summary terminates.

User response: Ensure that you are passing the correct

BRKOUT file in the BRKOUT DD statement of the JCL.

EQACU016S Summary: No space for BRKTAB file.

Explanation: The Coverage Utility summary program received an error code from the operating system command that was used to request storage for the BRKTAB file.

Action: Summary terminates.

User response: Increase the region size on your job card for this step. Coverage Utility obtains job card information from the user defaults.

EQACU017S Summary: No space for BP table or Bit map.

Explanation: The Coverage Utility summary program received an error code from the operating system command that was used to request storage for the BP table or Bit map.

Action: Summary terminates.

User response: Increase the region size on your job card for this step. Coverage Utility obtains job card information from the user defaults.

EQACU018S Summary: Invalid keyword or value *keyword*.

Explanation: One or more of the parameters that were passed to report are not valid.

Action: Report terminates.

User response: Correct the keyword or value.

Related references

“Parameters for the summary and report programs” on page 139

EQACU019S Report: Invalid file name *file_name* cannot be opened

Explanation: The file *file_name* cannot be opened.

Action: Report terminates.

User response: Ensure that the DDNAME *file-name* is specified in the JCL, that the file it points to exists and is available to the report job, and that the job that created the file completed successfully.

EQACU020S Report: Illegal listing input file *listing_name*

Explanation: Report detected something wrong with the assembler listing. Report expects a standard Assembler H or High Level Assembler listing format.

Action: Report terminates.

User response: Check that you are using a standard Assembler H or High Level Assembler listing. This

error can also be caused by having more than one compile unit in a BRKTAB or BRKOUT file that contains the same external CSECT name, which is not supported.

EQACU021S Report: No procedures or entry name found in listing.

Explanation: The Coverage Utility report program did not find an entry name in the listing being processed. For Enterprise COBOL for z/OS and OS/390, COBOL for MVS & VM and VS COBOL II, report expects to find an entname DS 0H instruction in the assembler listing section of the listing, where entname is the name that was used during setup to identify this PA. For OS/VS COBOL, the name of the first PROGRAM-ID is used (extracted from the title line). For PL/I, report expects to find Procedure entname in the assembler listing section of the listing.

Action: Report terminates.

User response: Verify that you are using the BRKOUT file of test case coverage results with the listings that go with it.

EQACU022S Report: Invalid BRKTAB file.

Explanation: The input BRKTAB file does not contain valid BRKTAB data.

Action: Report terminates.

User response: Ensure that setup has completed creating the BRKTAB file successfully. If it has, contact Coverage Utility support. If it has not, correct any errors that setup issued, and then rerun the job.

EQACU023S Report: Invalid BRKOUT file.

Explanation: The input BRKOUT file does not contain valid BRKOUT data.

Action: Report terminates.

User response: Verify that EQACUOSP or EQACUOSN has been issued in order to create the BRKOUT file. If it has, contact Debug Tool support.

EQACU024S Report: Entry name: *ename* not found in BRKTAB file.

Explanation: The entry name of the listing that was being processed was not found in the BRKTAB table. For COBOL, the entry name is from the PROGRAM-ID paragraph. For PL/I, the entry name is the name of the external procedure.

Action: Report terminates.

User response: Verify that the listing that you are processing is one that has coverage data in the BRKTAB file that is being used. You should be able to

find the first CSECT name of your listing in the BRKOUT file.

EQACU025S Report: Entry name: *ename* not found in BRKOUT file.

Explanation: The entry name of the listing being processed was not found in the BRKOUT table. For COBOL, the entry name is from the PROGRAM-ID paragraph. For PL/I, the entry name is the name of the external procedure.

Action: Report terminates.

User response: Verify that the listing you are processing is one that has coverage data in the BRKOUT file being used. You should be able to find the first CSECT name of your listing in the BRKOUT file.

EQACU026S Report: No space for BRKTAB table.

Explanation: The Coverage Utility report program received an error code from the operating system command that was used to request storage for the BRKOUT table.

Action: Report terminates.

User response: Increase the region size on your job card for the report step. Coverage Utility obtains job card information from the user defaults.

EQACU027S Report: No space for Bit map.

Explanation: The Coverage Utility report program received an error code from the operating system command used to request storage for the bit map.

Action: Report terminates.

User response: Increase the region size on your job card for this step. Coverage Utility obtains job card information from the user defaults.

EQACU028E Report: Mismatches of opcode in storage during execution. *listing_name*

Explanation: An opcode in the listing did not match that in the BRKTAB. The BRKTAB and listing might be out of synch.

Action: The opcode is discarded and processing continues.

User response: Ensure that the listing that is being annotated is the same one that is used to generate the BRKTAB.

EQACU029E Report: Listing name *dsn* not found in BRKTAB file.

Explanation: The specified listing data set name was not found in the BRKTAB file.

Action: Processing terminates.

User response: Correct the *dsn* specification.

EQACU030S Report: Listing name *dsn* not found in BRKOUT file.

Explanation: The specified listing data set name was not found in the BRKOUT file.

Action: Processing terminates.

User response: Correct the *dsn* specification.

EQACU031S Report: Unsupported value specified in parameter *keyword*.

Explanation: The operand of the specified keyword contained a value that is not valid.

Action: Processing terminates.

User response: Correct the operand of the specified keyword.

EQACU032S Summary: Unsupported value specified in parameter *keyword*.

Explanation: The operand of the specified keyword contained a value that is not valid.

Action: Processing terminates.

User response: Correct the operand of the specified keyword.

EQACU033S Export XML: Invalid keyword or value *keyword*.

Explanation: The parameter that was passed to export XML data is not valid.

Action: Export terminates.

User response: Correct the parameter.

Related references

“Parameters for the export data program” on page 140

EQACU034S Export XML: EOF of BRKTAB before finding the *testid* test case.

Explanation: The *testid* test case was not found in the BRKTAB file provided.

Action: Export terminates.

User response: Verify that the BRKTAB file in the BRKTAB DD statement of the JCL is the same one that was used when you ran the monitor session that

created the BRKOUT file in the BRKOUT DD statement. If you have run setup to create a new BRKTAB file since creating the BRKOUT file, the BRKOUT and the BRKTAB files will not match.

EQACU035S Export XML: Invalid file name *file_name* cannot be opened.

Explanation: The file *file_name* cannot be opened.

Action: Export terminates.

User response: Verify that the DDNAME *file-name* is specified in the JCL, that the file it points to exists and is available to the export job, and that the job that created the file completed successfully.

EQACU036S Export XML: Invalid BRKTAB file.

Explanation: The BRKTAB file is not valid.

Action: Export terminates.

User response: Verify that you are passing the correct BRKTAB file in the BRKTAB DD statement of the JCL.

EQACU037S Export XML: Invalid BRKOUT file.

Explanation: The BRKOUT file is not valid.

Action: Export terminates.

User response: Verify that you are passing the correct BRKOUT file in the BRKOUT DD statement of the JCL.

EQACU038S Export XML: No space for BRKTAB file.

Explanation: The export XML data program received an error code from the operating system command that was used to request storage for the BRKTAB file.

Action: Export terminates.

User response: Increase the region size on your job card for this step. Coverage Utility obtains job card information from the user defaults.

EQACU039S Export XML: No space for BP table or Bit map.

Explanation: The export XML data program received an error code from the operating system command that was used to request storage for the BP table or bit map.

Action: Export terminates.

User response: Increase the region size on your job card for this step. Coverage Utility obtains job card information from the user defaults.

EQACU040S Export XML: Unsupported value specified in parameter *keyword*.

Explanation: The operand of the specified keyword contained a value that is not valid.

Action: Processing terminates.

User response: Correct the operand of the specified keyword.

EQACU041S Export XML: No space for Execute tag processing.

Explanation: The export XML data program received an error code from the operating system command used to request storage for Execute tag processing.

Action: Export terminates.

User response: Increase the region size on your job card for this step. Coverage Utility obtains job card information from the user defaults.

EQACU042S Setup: Load and Object data sets can not both be specified within a set of control cards.

Explanation: The object library control cards FROMOBJDSN and TOOBJDSN are mutually exclusive with the load library control cards FROMLOADDSN and TOLOADDSN within a given control data set.

Action: Setup terminates.

User response: Ensure that only object library control cards or only load library control cards are used within a single control data set.

EQACU043S Setup: Illegal listing type in Coverage Utility control file

Explanation: The listing type value (first field) on a Coverage Utility control card was not a valid *list_type*.

Action: Setup terminates.

User response: Correct the value. Rerun setup.

Related references

“EQACUSET” on page 70

EQACU044S Setup: No BRKTAB written. Too few bpoints.

Explanation: No BRKTAB was written, because setup could not find assembler instructions at which to insert breakpoints.

Action: Setup terminates.

User response: Check the input listing that was passed to setup and verify that it is correct, including the assembler statements.

EQACU045S Setup: Illegal assembler listing

Explanation: The assembler listing is not valid.

Action: Setup terminates.

User response: Ensure that the assembler listing that was passed to setup is a valid Assembler H or High Level Assembler listing.

EQACU046S Setup: Undefined error: *errnum*

Explanation: An unexpected error occurred.

Action: Setup terminates.

User response: Contact Debug Tool support with the value of *errnum*.

EQACU047S Setup: Cannot find CSECT: *csect_name*

Explanation: Setup was unable to find the specified CSECT, *csect_name*, in the listing.

Action: Setup terminates.

User response: Ensure that the CSECT name is spelled correctly and is associated with the correct listing in the control file.

EQACU048S Setup: Wrong load module.

Explanation: The specified CSECT name was not found in the load module.

Action: Setup terminates.

User response: Ensure that the correct CSECT and load module names were specified in the control file.

EQACU049S Setup: TXT record not found to match this BRKTAB record. OFFSET (*oooo*) OPCODE (*cccc*)

Explanation: During the scan of an object module, a TXT record was found not to match the breakpoint code (*cccc*) at offset (*oooo*) in the BRKTAB data set. The BRKTAB data set contains all of the breakpoints as identified from the source listings. Setup reads the BRKTAB data set and looks through the object for each breakpoint. If a breakpoint cannot be found, the program terminates.

Action: Program terminates.

User response: Verify that the object module that was passed as input to the program was created by the same source listing as the BRKTAB file. Rerun setup.

EQACU050S Setup: Breakpoint record not found. Errors possible. Check the RPTMSGs data set for additional information.

Explanation: No match was found in the object module for this breakpoint in BRKTAB.

EQACU051S • EQACU059E

Action: Program continues.

User response: Verify that the object module that was passed as input to the program was created by the same source listing as the BRKTAB file. Rerun setup.

EQACU051S Setup: BRKTAB header not found! Cannot continue. BRKTAB header is not record # *nnnnn*.

Explanation: A BRKTAB header number could not be found. The number that was passed is outside the range of possibilities for this BRKTAB.

Action: Program terminates.

User response: Correct the errors that were found in the setup run, and then rerun setup.

EQACU052S Setup: Storage could not be allocated. Program ending. Insufficient storage.

Explanation: Setup could not acquire enough storage in order to process the object module.

Action: Program terminates.

Problem Determination: Not enough storage was available to the job.

User response: Specify a larger REGION size for the setup job.

EQACU053S Setup: Cannot open BRKTAB file created during Setup.

Explanation: The BRKTAB file of breakpoint data created in a previous step cannot be opened.

Action: Program terminates.

User response: The BRKTAB file is created in a previous step. You should not see this error unless you modified the JCL for inclusion into your build process. Check the modified setup JCL.

EQACU054S Setup: Cannot open output file for VER/REP records.

Explanation: The output file that contains update records for the EQACUZPP program that instruments load modules cannot be opened.

Action: Program terminates.

User response: This file is created as a temporary data set in the JCL. You should not see this error unless you modified the JCL for inclusion into your build process. Check the modified setup JCL.

EQACU055S Setup: Illegal BRKTAB file.

Explanation: The BRKTAB file of breakpoint data that was created in a previous step is not a correct BRKTAB.

Action: Program terminates.

User response: The BRKTAB file is created in a previous step. You should not see this error unless you modified the JCL for inclusion into your build process. Check the modified setup JCL.

EQACU056S Setup: Unsupported value specified in parameter *keyword*.

Explanation: The value specified as the operand of the indicated keyword is not valid.

Action: Program terminates.

User response: Correct the indicated operand.

EQACU057S Setup: Invalid keyword or value *keyword*.

Explanation: The indicated keyword or keyword operand is invalid.

Action: Program terminates.

User response: Correct the indicated keyword or operand.

EQACU058W Setup: Listing has no executable statements: no breakpoints inserted

Explanation: A listing that was submitted to setup has no executable statements.

Action: Setup program continues. A BRKTAB with no breakpoints is produced.

User response: You can continue with execution and reports, if this BRKTAB is used with other BRKTABs that do contain breakpoints. This module will not be included in the summary report.

EQACU059E Setup: Input statement out of order at statement *linenum*

Explanation: The input (SYSIN) statement listed immediately before this message (which was found at line *linenum* in the SYSIN file) was not in the expected order. Statements are expected to be in the following order:

- NAME
- BASE (optional)
- VERIFY
- REP (for the same address specified in the preceding VERIFY)
- more VERIFY / REP pairs or another NAME

Action: Execution is aborted. Breakpoints are not applied to the load module that is being processed or

to any subsequent load modules that are to be processed.

User response: Correct the order of input statements, and then rerun the job.

EQACU060E Setup: Invalid hexadecimal number in input statement *linenum*

Explanation: The input (SYSIN) statement that is listed immediately before this message (which was found at line *linenum* in the SYSIN file) contained a hexadecimal number that had either of these errors::

- Contained an invalid hexadecimal digit (not 0-9 and A-F) or
- Did not consist of either 2, 4, 6, or 8 digits

Action: Execution is aborted. Breakpoints are not applied to the load module that is being processed or to any subsequent load modules that is to be processed.

User response: Correct the input statement, and then rerun the job.

EQACU061E Setup: Unexpected input statement at statement *linenum*

Explanation: The input (SYSIN) statement that is listed immediately before this message (which was found at line *linenum* in the SYSIN file) was not a comment or a NAME, BASE, VERIFY, or REP statement.

Action: Execution is aborted. Breakpoints are not applied to the load module that is being processed or any subsequent load modules that are to be processed.

User response: Correct the input statement, and then rerun the job.

EQACU062E Setup: Operand too long in statement *linenum*

Explanation: The input (SYSIN) statement that is listed immediately before this message (which was found at line *linenum* in the SYSIN file) contained an operand that was longer than expected. The maximum length expected for each operand is:

NAME - load module name
8 characters

NAME - External / CSECT name
1024 characters

BASE 8 characters

VERIFY / REP - offset
8 characters

VERIFY / REP - old / new data
8 characters

Action: Execution is aborted. Breakpoints are not applied to the load module that is being processed or

to any subsequent load modules that are to be processed.

User response: Correct the input statement, and then rerun the job.

EQACU063W Setup: Breakpoints already applied.

Explanation: All VERIFY statements failed. However, the current data exactly matched the specified new data in all cases (that is, all breakpoints were already in place).

Action: Execution is terminated. However, the load module that was being processed was copied to the output (LIBOUT) file.

User response: Correct the VERIFY statements, and then rerun the job.

EQACU064W Setup: Some breakpoints already applied.

Explanation: Some VERIFY statements failed. However, the current data exactly matched the specified new data in all cases where the VERIFY failed (that is, some breakpoints were already in place).

Action: Execution is terminated. However, the load module being processed was copied to the output (LIBOUT) file with all desired breakpoints created.

User response: Correct the VERIFY statements, and then rerun the job.

EQACU065E Setup: Load module not found on LIBIN.

Explanation: The load module that is being processed was not present in the data set that was specified by the LIBIN DD statement.

Action: Execution is terminated.

User response: Correct the LIBIN specification, and then rerun the job.

EQACU066E Setup: Error returned from Binder API invoked by EQACUBDZ to apply breakpoints.

Explanation: When the Binder API (IEWBIND) was invoked to apply breakpoints to the specified load module, IEWBIND returned a non-zero return code. This message is followed by message EQACU068E which lists the return and reason codes returned by IEWBIND.

Action: Execution is terminated.

User response: Correct the cause of the error, and then rerun the job.

Related references

DFSMS Program Management

EQACU067E Setup: Error returned from Binder API invoked by EQACUBDM to map external symbols.

Explanation: When the Binder API (IEWBIND) was invoked to read the external symbols in the specified load module, IEWBIND returned a non-zero return code. This message is followed by message EQACU068E, which lists the return and reason codes returned by IEWBIND.

Action: Execution is terminated.

User response: Correct the cause of the error, and then rerun the job.

Related references

DFSMS Program Management

EQACU068E Setup: Return/Reason codes: *returncode/reasoncodeX*

Explanation: The specified return and reason codes were encountered. Refer to the previous message to determine how to interpret these codes.

Action: Refer to message EQACU067E.

User response: Refer to the previous message.

EQACU069E Setup: Unable to load IEWBIND.

Explanation: IEWBIND (the Binder API interface) could not be loaded from the system linklist or the current JOBLIB or STEPLIB.

Action: Execution is terminated.

User response: Ensure that IEWBIND is available in the system linklist or the current JOBLIB or STEPLIB, and rerun the job.

EQACU070E Setup: External (CSECT) name not found.

Explanation: The current external (CSECT) name was not present in the load module that was read from the LIBIN file.

Action: Execution is terminated.

User response: Correct the LIBIN DD specification or the NAME control statement, and then rerun the job.

EQACU071E Setup: Verify failure. Address 'offset'X contains 'olddata'X instead of 'expecteddata'X.

Explanation: The specified *offset* contains *olddata* instead of the data that was specified on the VERIFY statement.

Action: Execution is terminated.

User response: Correct the VERIFY statement, and then rerun the job.

EQACU072E Setup: Return code from EQACUBDZ: *returncode*.

Explanation: The specified return code was returned from EQACUBDZ.

Action: Execution is terminated.

User response: Contact Debug Tool support.

EQACU073E Setup: Cannot open SYSIN, SYSPRINT, or LIBIN.

Explanation: The SYSIN, SYSPRINT or LIBIN file could not be opened. This message is issued using a Write To Operator.

Action: Execution is terminated.

User response: Correct SYSIN, SYSPRINT, or LIBIN DD specification, and then rerun the job.

Translation: This message is not translated.

EQACU074E Setup: Operand missing from input statement.

Explanation: A required operand was not found on the input statement that immediately precedes this message. The NAME, VERIFY, and REP statements require two operands. The BASE statement requires one operand.

Action: Execution is terminated.

User response: Correct the indicated statement, and rerun the job.

EQACU075W Setup: C/C++ compiler was executed with inline optimization in effect - Performance Mode will be turned on.

Explanation: Coverage Utility cannot accurately measure conditional branch coverage when inlining is used in C/C++ code. Setup has been run with the performance mode option set to Y.

Action: Setup uses Y for the performance mode option and processes the C/C++ module for statement coverage only.

User response: You can continue with the steps to measure coverage. Your statement coverage for C/C++ that contains inlined functions will be correct. If you want conditional branch coverage, you must compile without inlining.

Related concepts

“The effects of inlining” on page 136

EQACU076W Setup: Listing has too few executable statements: no breakpoints inserted

Explanation: A listing submitted to setup created only one or two breakpoints. This is too few to allow for accurate identification of breakpoints during execution.

Action: Setup program continues. A BRKTAB with no breakpoints is produced.

User response: You can continue with execution and reports if this BRKTAB is used with other BRKTABS that do contain breakpoints. This module will not be included in the summary report.

EQACU077I Setup: The assembler Program Area *csect_name* containing OFFSET (0000) OPCODE (cccc) has too few executable statements. No breakpoints have been inserted for this PA.

Explanation: An assembler listing that was submitted to setup contains a CSECT (or partial CSECT if there are multiple interleaved CSECTs) that contains code that caused only one or two breakpoints to be created by setup. This is too few to allow for accurate identification of breakpoints during execution.

Action: Setup program continues. The indicated program area and breakpoints are ignored during setup.

User response: None. This message is issued only to indicate that the statements in this CSECT will not have breakpoints placed in them. They will always be marked as 'not executed' in the annotated listing.

EQACU078S Can not find monitor program (EQACUOSV) in authorized library.

Explanation: The system could not find the monitor program (EQACUOSV).

Action: The monitor installer terminates.

User response: Verify that the monitor program is installed in an authorized library in the load module search path for the installation job.

Translation: This message is not translated.

EQACU079S Cannot allocate space in ECSA for monitor.

Explanation: The install program cannot allocate ECSA storage for the monitor. Insufficient ECSA storage was available to install the monitor program.

Action: The monitor installer terminates.

User response: Verify with your system programmer that the amount of ECSA storage needed by the monitor is available.

Related references

Appendix B, "Resources and requirements," on page 187

Translation: This message is not translated.

EQACU080S Error loading monitor in ECSA storage.

Explanation: Loading of the monitor into ECSA (extended common service area) failed.

Action: The monitor installer terminates.

User response: Verify that the monitor program is installed in an authorized library in the load module search path for the installation job.

Translation: This message is not translated.

EQACU081S Error updating SVC table with new SVC numbers.

Explanation: Updating of the SVC table with the new user SVC numbers failed.

Action: The monitor installer terminates.

User response: Verify that the user SVC numbers passed to the monitor install program (EQACUOIN) are available.

Translation: This message is not translated.

EQACU082S Error removing prior SVC token.

Explanation: An operating system error occurred while trying to remove an SVC token.

Action: The function terminates.

User response: This is an operating system error issued when trying to remove an SVC token. Retry. If the problem persists, contact system support or Debug Tool support.

Translation: This message is not translated.

EQACU083S Error setting SVC token.

Explanation: An operating system error occurred while trying to obtain an SVC token.

Action: The function terminates.

User response: This is an operating system error when trying to obtain an SVC token. Retry. If the problem persists, contact system support or Debug Tool support.

Translation: This message is not translated.

EQACU084S Invalid short SVC number.

Explanation: The SVC number that was supplied for short opcodes (the first number passed to EQACUOIN) is not permitted. It must be a hexadecimal number between C8 and FF.

Action: The monitor installer terminates.

User response: Ensure that the first parameter that is passed to the EQACUOIN program in the monitor install JCL is a hexadecimal number between C8 and FF without any surrounding quotation marks.

Translation: This message is not translated.

EQACU085S Invalid long SVC number.

Explanation: The SVC number that was supplied for long opcodes (the second number passed to EQACUOIN) is not permitted. It must be a hexadecimal number between C8 and FF.

Action: The monitor installer terminates.

User response: Ensure that the second parameter that is passed to the EQACUOIN program in the monitor install JCL is a hexadecimal number between C8 and FF without any surrounding quotation marks.

Translation: This message is not translated.

EQACU086S Invalid command

Explanation: A command that is not valid was issued.

Action: The command was ignored.

User response: Because the commands are generated by REXX programs or created JCL, check that the REXX or JCL that issued the command has not been corrupted.

EQACU087S Error reading BRKTAB file

Explanation: You tried to start a session, but there was an error reading the BRKTAB file of breakpoint data.

Action: The command was not run.

User response: Verify that the BRKTAB file (ddname BRKTAB in the start monitor JCL *Xnnnnnnn*) is accessible and is a valid BRKTAB file (starts with characters *BT8*).

EQACU088S Invalid BRKTAB file

Explanation: You tried to start a session, but the BRKTAB file of breakpoint data is not valid.

Action: The command was not run.

User response: Verify that the BRKTAB file (ddname BRKTAB in the start monitor JCL *Xnnnnnnn*) is a valid BRKTAB file and was created by the V3R1M0 or later setup program (starts with characters *BT8*).

EQACU089S Error opening BRKOUT file

Explanation: You tried to stop a session, but the BRKOUT file of breakpoint data could not be opened.

Action: The command was not run.

User response: Verify that the BRKOUT file (ddname BRKOUT in the start monitor JCL *Xnnnnnnn*) exists and is accessible.

EQACU090S Error writing BRKOUT file

Explanation: You tried to stop a session, but the BRKOUT file of breakpoint data could not be written.

Action: The ommand was not run.

User response: Verify that the BRKOUT file (ddname BRKOUT in the start monitor JCL *Xnnnnnnn*) exists and is accessible.

EQACU091S No ECSA space for BP table

Explanation: You tried to start a session, but the monitor could not allocate sufficient space for the breakpoint (BP) table in ECSA.

Action: The command was not run.

User response: Reduce the number of object modules being tested, or contact the system programmer to increase ECSA storage.

Related references

Appendix B, "Resources and requirements," on page 187

EQACU092S No ECSA space for PA table

Explanation: You tried to start a session, but the monitor could not allocate sufficient space for the program area (PA) table in ECSA.

Action: The command was not run.

User response: Reduce the number of object modules being tested, or contact the system programmer to increase ECSA storage.

Related references

Appendix B, "Resources and requirements," on page 187

EQACU093S Downlevel Monitor running - cannot execute command

Explanation: The installed monitor is at a previous level that is now incompatible.

Action: The command was not run.

User response: Install the current monitor.

Translation: This message is not translated.

EQACU094S No storage in SQA for monitor tables

Explanation: During installation of the Coverage Utility monitor, the monitor could not obtain SQA space for its tables.

Action: The installation of the monitor is terminated.

User response: Contact your system programmer to determine the amount of SQA space available on your system.

Related references

Appendix B, "Resources and requirements," on page 187

Translation: This message is not translated.

EQACU095W Dup listing in ID: user_ID listing_name

Explanation: You started a monitor session that contains a BRKTAB for a listing or object module that is being monitored by another session.

Action: The session is started.

User response: If two or more sessions share a BRKTAB for a listing or object module, the first session that was started gets all coverage for the listing or object module. Sessions that are started after the first have incomplete coverage for any shared listings and object modules.

EQACU096S The requested session id *sess_id* was not found.

Explanation: You issued a command that requires a session ID, but the session ID (*sess_id*) was not found.

Action: The command was not run.

User response: Verify that the *sess_id* is valid. The session ID is usually your TSO session ID.

EQACU097S The requested session id is not active. Command rejected.

Explanation: You issued a command that requires a session ID, but the session ID was not active.

Action: The command was rejected.

User response: Verify that the *sess_id* is active by using the EQACUOSE command.

EQACU098S The requested PA# *pa_num* was not found.

Explanation: You issued a command that requires a program area (PA) number, but the PA number *pa_num* was not found.

Action: The command was not run.

User response: Verify that the *pa_num* is valid. The EQACUOSL command lists the number of PAs for each listing in a session.

EQACU099S No free sessions in session table - session not started

Explanation: You attempted to start a new session, but there are new free sessions in the monitor session table.

Action: The command was not run.

User response: Use the EQACUOSE to display active sessions. Stop or quit sessions that are not needed. A

maximum of 32 sessions can be active.

EQACU100S Session already active, cannot be started again

Explanation: You attempted to start a new session, but a session is already started with that session ID.

Action: The command was not run.

User response: Use the EQACUOSE command to display active sessions. Stop or quit the session with the name that you are trying to start, or start a session with a different name.

EQACU101S No previous copy of the monitor exists - cannot do commands to it

Explanation: The monitor is not installed, or has been corrupted.

Action: The command was not run.

User response: Reinstall the monitor.

EQACU102S No session table exists.

Explanation: The session table does not exist or has been corrupted.

Action: The command was not run.

User response: Reinstall the monitor.

EQACU103S The command was not recognized by Coverage Utility

Explanation: The command was not recognized by the command processor.

Action: The command was not run.

User response: Because the commands are generated by REXX execs or created JCL, verify that the REXX or JCL that issued the command has not been corrupted.

EQACU104S Error number *err_num* occurred. Contact Debug Tool support.

Explanation: An undocumented error occurred.

Action: The operation terminates.

User response: Contact Debug Tool support with the error number and explain the circumstances that caused it.

EQACU105S Browse of Messages file was unable to complete.

Explanation: The messages file could not be browsed. The messages file is used by the monitor commands to return messages or data.

Action: The function terminates.

EQACU106S • EQACU115S

User response: Check for other messages or ISPF problems.

EQACU106S Error executing the *cmd* command. Be sure that EQACUOCM is in *authlib*

Explanation: The call to the command processor failed.

Action: The function terminates.

User response: Ensure that EQACUOCM is in the specified authorized library.

EQACU107I No sessions currently active.

Explanation: There are no monitor sessions currently active.

Action: This message is displayed.

User response: None.

EQACU108S Session *sess_id* has not been stopped. Verify that the session exists.

Explanation: An error occurred in the EQACUOSP routine, and the session has not been stopped. The likely cause for the associated errors is that the specified session ID does not exist.

Action: Command not run.

User response: Verify that the session indicated exists. If it does, address other messages and retry, or issue the EQACUOQT command.

EQACU109S Coverage Utility unidentified BP at addr *xxxxxxx* BP *yyyyyyyy*

Explanation: Coverage Utility could not identify an SVC breakpoint inserted into the user program. The breakpoint is at address *xxxxxxx* in the program. The unidentified breakpoint is *yyyyyyyy*.

Action: The program terminates with a 7C1 abend.

User response: Follow the instructions in the following reference.

Related references

“Solving system 7C1 abend in a user program” on page 159

Translation: This message is not translated.

EQACU110S Coverage Utility unidentified BP at addr *xxxxxxx* BP *yyyyyyyy*

Explanation: Coverage Utility could not identify an SVC breakpoint inserted in the user program (that was running in a CICS environment). The breakpoint is at address *xxxxxxx* in the program. The unidentified breakpoint is *yyyyyyyy*.

Action: The program terminates with a 7C1 abend.

User response:

Related references

“Solving system 7C1 abend in a user program” on page 159

Translation: This message is not translated.

EQACU111S Invalid parameter: *xxx*

Explanation: The parameter *xxx* passed to a monitor program is invalid.

Action: Program terminates.

User response: Provide a correct parameter (or none to use the installation default).

EQACU112S Unsupported value specified in parameter *xxx*.

Explanation: The value passed to parameter *xxx* is invalid.

Action: Program terminates.

User response: Provide a correct value for the parameter (or none to use the installation default).

EQACU113S No BPs in BRKTAB. - session not started

Explanation: You tried to start a monitor session with a BRKTAB file (the data on BPs from the setup step) that contained no breakpoints.

Action: Requested monitor session is not started.

User response: Verify that you are providing the correct BRKTAB file in the JCL that starts the session. Verify that the setup step that created this BRKTAB had a listing to process that contained executable code.

EQACU114S INTERNAL SOFTWARE ERROR: Literal *xxx* was not found.

Explanation: An internal software error occurred.

Action: Program terminates.

User response: Contact Debug Tool support.

EQACU115S Error executing *xxxxxxx*. Possible allocation error

Explanation: EQACUOSP ran the 'prefix.sessionid.EXTEMP.EXEC¹¹ REXX EXEC to allocate the output BRKOUT data set. This EXEC could not allocate the BRKOUT data set OLD.

Action: EQACUOSP terminates without stopping the session.

User response: Determine why the BRKOUT data set could not be allocated OLD. Correct the problem, and then rerun EQACUOSP.

EQACU116S Control: Error xxxxxxxx for data set dddddddd.

Explanation: The specified message (xxxxxxx) was returned from the SYSDSN command, indicating that the specified data set (ddddddd) could not be found or could not be processed.

Action: Processing terminates.

User response: Correct the data set name specification or make sure that the data set can be accessed.

EQACU117S Control: 00000000 is not a valid option.

Explanation: The specified option was found on the command invocation but is not a valid option that is recognized by this command.

Action: Processing terminates.

User response: Remove or correct the specified option.

EQACU118S Control: Return Code rr from ALLOCATE allocating dddddddd.

Explanation: Return code *rr* was returned from the ALLOCATE command while attempting to allocate the specified input data set.

Action: Processing terminates.

User response: Correct the data set name specification or ensure that the data set can be allocated.

EQACU119S Control: Return Code rr from EXECIO reading dddddddd.

Explanation: Return code *rr* was returned from the EXECIO command while attempting to read the specified data set.

Action: Processing terminates.

User response: Correct the indicated error or correct the data set name specification.

EQACU120S Control: Invalid DBCS or mixed string. Check DBCS name(s) on line num of the control file.

Explanation: The statement shown in the second line of the message contains a DBCS or mixed string that is not valid. This statement was found on line number *num* in the control file.

Action: Processing terminates.

User response: Correct the DBCS usage in the statement indicated.

EQACU121S Control: 00000000 is not a valid option for the ssss statement.

Explanation: A keyword (0000000) was specified for the *ssss* statement, which is not valid on that statement.

Action: Processing terminates. The statement in error is shown in the second line of the error message.

User response: Correct the specified option.

EQACU122S Control: Volume and Unit cannot be specified without the corresponding DSName.

Explanation: A keyword FromVol or FromUnit was specified without FromObjDsn or ToVol or ToUnit was specified without ToObjDsn statement, which is not valid on that statement. These operands can be specified only when the corresponding data set name is specified.

Action: Processing terminates. The statement in error is shown in the the error message.

User response: Correct the specified option.

EQACU123S Control: 00000000 is not a valid option for the ssss statement.

Explanation: A keyword (0000000) was specified for the *ssss* statement which is not valid on that statement.

Action: Processing terminates. The statement in error is shown in the the error message.

User response: Correct the specified option.

EQACU124S Control: Required operand omitted from:

Explanation: You omitted a required operand from the statement shown in the second line of the message.

Action: Processing terminates. The statement in error is shown in the error message.

User response: Correct the statement by adding all required operands.

EQACU125S Control: LISTDSN=ddddddd contains an asterisk specification. However LISTMEMBER= was not specified. stmt

Explanation: When the data set name *ddddddd* contains an asterisk, LISTMEMBER= must be specified to replace the asterisk.

Action: Processing terminates. The statement in error is shown in the second line of the error message.

User response: Correct the statement by adding the LISTMEMBER= operand or removing the asterisk from the LISTDSN= operand.

EQACU126S Control: *llll* is a duplicate label on the following statement. It is ignored. *stmt*

Explanation: The specified label was previously found on a statement of the same type.

Action: Processing terminates. The statement in error is shown in the second line of the error message.

User response: Change the label to be unique.

EQACU127S Control: *llll* is not a previously defined label. The following statement is ignored. *stmt*

Explanation: The statement shown in the second line of the message referenced a label that was not previously defined.

Action: Processing terminates. The statement in error is shown in the second line of the error message.

User response: Correct the label reference.

EQACU128S Control: Return Code *rr* from EXECIO writing *ddddddd*.

Explanation: Return Code *rr* was returned from the EXECIO command attempting to write the specified data set.

Action: Processing terminates.

User response: Correct the indicated error or correct the data set name specification. *

EQACU129S Control: *ffffff* file is not allocated.

Explanation: The specified DDNAME was not previously allocated.

Action: Processing terminates.

User response: Correct the DDNAME specification or ensure that the file is allocated.

EQACU130S Control: Last line is continued. It is ignored.

Explanation: The last non-comment line in the control file ended in a comma indicating that it was continued. Since no more lines were found in the file, the partial line is ignored.

Action: Processing terminates.

User response: Correct the control statement.

EQACU131S Control: Statement *sssssss* not recognized.

Explanation: The indicated keyword is not a valid control statement type.

Action: Processing terminates. The statement in error is shown in the second line of the error message.

User response: Correct the control statement.

EQACU132S Control: *oooooooo* is not a valid operand for the *kkkkkkkk* keyword. *stmt*

Explanation: The indicated operand is not valid for the indicated keyword.

Action: Processing terminates. The statement in error is shown in the second line of the error message.

User response: Correct the control statement.

EQACU133S Control: Load and Object data sets can not both be specified within a set of control cards. *stmt*

Explanation: The object library control cards FROMOBJDSN and TOOBJDSN are mutually exclusive with the load library control cards FROMLOADDSN and TOLOADDSN within a given control data set.

Action: Processing terminates. The statement at which the error was detected is shown in the second line of the error message.

User response: Ensure that only object library control cards or only load library control cards are used within a single control data set.

EQACU134S Control: Empty control file data set.

Explanation: The control file data set (Coverage Utility) exists, but is empty.

Action: Processing terminates.

User response: Ensure that the correct Coverage Utility control card data set is specified.

EQACU135S Control: No compilation unit statement found (COBOL, PL/I, ...). No output will be generated.

Explanation: The specified control file data set does not contain any compilation unit statements (COBOL, PL/I, etc.).

Action: Processing terminates.

User response: Ensure that the Coverage Utility control card data set contains at least one compilation unit statement.

EQACU136S Control: Error at line *lll* in the control file.
control_file_statement
Return Code=*rc*

Explanation: The indicated control file statement at the indicated line in the control file is not valid.

Action: Processing terminates.

User response: Correct the indicated statement.

EQACU137I *function is starting*

Explanation: The specified function is starting.

Action: The function begins execution

User response: None.

EQACU138I *function is verifying your parameters*

Explanation: The specified function is verifying that the parameters are correct.

Action: The function continues.

User response: None.

EQACU139I *function is done.*

Explanation: The specified function has completed.

Action: The function terminates.

User response: None.

EQACU140I *Data saved data*

Explanation: The specified data has been saved.

Action: The function continues.

User response: None.

EQACU141S *Data set data set name not found*

Explanation: The specified data set was not found.

Action: The function terminates.

User response: Specify the name of an existing data set.

EQACU142S *Unknown option*

Explanation: An unknown option was encountered when Coverage Utility tried to run a function.

Action: The function terminates.

User response: Contact Debug Tool support.

EQACU143I *JCL submitted*

Explanation: The JCL to run the requested function has been submitted.

Action: A batch job is submitted and the function terminates.

User response: None.

EQACU144I *Data set dsn reset*

Explanation: The specified control data set has been reset to the default values.

Action: The function terminates.

User response: None.

EQACU145S *Copy error occurred trying to reset data set dsn*

Explanation: When Coverage Utility tried to reset a control data set from the default template, an error occurred either in reading the default template file SEQASAMP or writing to the user control file.

Action: The function terminates.

User response: Verify that the default template file SEQASAMP is available on the system and that the user control file that is to be reset can be written to.

EQACU146S *Form frm not found.*

Explanation: The control file template *frm* was not found.

Action: The function terminates.

User response: Ensure that the **Sample Dsn** field under “General Defaults” in your user defaults specifies the data set in which the control file templates have been installed.

EQACU147S *option is not a valid option*

Explanation: The specified option is not valid for this function.

Action: The function terminates.

User response: Correct the option, and then retry the function.

EQACU148S *Error rc allocating dsn*

Explanation: An error occurred when Coverage Utility tried to allocate the specified data set. The allocation return code is indicated by *rc*.

Action: The function terminates.

User response: Determine the allocation error, and then retry the function. Ensure that data sets required for this function exist.

EQACU149I *data*

Explanation: This message displays information in support of message EQACU148S.

Action: See the corresponding message.

User response: See the corresponding message.

EQACU150S *No output data set specified*

Explanation: The function requires the name of an output data set and none was specified.

Action: The function terminates.

EQACU151S • EQACU163S

User response: Specify the name of a data set to contain output from the function.

EQACU151S Error processing input/output data set *dsn* : *message*

Explanation: An error occurred while processing the data set specified by *dsn*. Further information about the error is contained in *message*. For example, *message* can indicate a data set or member not found.

Action: The function terminates.

User response: Correct the specified error, and then retry the function.

EQACU152S Members specified but input not PDS

Explanation: A member has been specified for a data set that is not a partitioned data set.

Action: The function terminates.

User response: Do not specify a member name, or specify the correct data set.

EQACU153S Invalid operand *operand*

Explanation: An operand that is not valid has been specified.

Action: The function terminates.

User response: Correct the error, and then retry the function.

EQACU154S Input is PDS but output is not

Explanation: You cannot place the members of a partitioned data set into a sequential data set. Both input and output files must be the same organization.

Action: The function terminates.

User response: Specify a partitioned data set as the output data set.

EQACU155S Input is sequential but output is PDS

Explanation: The input is a sequential data set, but the output is a partitioned data set and no member name has been specified.

Action: The function terminates.

User response: Specify a sequential data set as the output data set or specify a member name.

EQACU156S Error *rc* from *function*

Explanation: The specified function returned the indicated return code.

Action: The function terminates.

User response: Check for other error messages.

EQACU157S ISPF not active

Explanation: This function requires ISPF to run.

Action: The function terminates.

User response: Start ISPF, and then retry the function.

EQACU158S Operand *operand* of *function* is *message*

Explanation: The indicated operand for this function is incorrect. Further information is contained in *message*.

Action: The function terminates.

User response: Correct the operand, and then retry the function.

EQACU159S *operation* is not currently supported

Explanation: The indicated operation for this function is not supported.

Action: The function terminates.

User response: Specify a supported operation.

EQACU160S Error *rc* deleting *dsn*

Explanation: An error occurred when trying to delete the specified data set. *rc* contains the return code from delete.

Action: The function terminates.

User response: Correct the error indicated by the return code, and then retry.

EQACU161S JCL submit error *rc*

Explanation: The TSO submit failed with *rc=rc*.

Action: The function terminates.

User response: None.

EQACU162I Performing File Tailoring *step jcldsn*

Explanation: File tailoring is being performed for *step* on data set *jcldsn*.

Action: The function continues.

User response: None.

EQACU163S FTOpen error *rc=rc*

Explanation: The FTOpen failed with *rc=rc*.

Action: The function terminates.

User response: None.

EQACU164S FTIncl error rc=rc

Explanation: The FTIncl failed with rc=rc.

Action: The function terminates.

User response: None.

EQACU165S FTClose error rc=rc

Explanation: The FTClose failed with rc=rc.

Action: The function terminates.

User response: None.

EQACU166S Unable to alloc ddname data set dsname

Explanation: The TSO alloc command for *ddname* and *dsname* failed.

Action: The function terminates.

User response: None.

EQACU167S Unable to read ddname data set dsname

Explanation: Unable to read *dsname* using *ddname*.

Action: The function terminates.

User response: None.

EQACU168S Unable to write ddname data set dsname

Explanation: Unable to write *dsname* using *ddname*.

Action: The function terminates.

User response: None.

EQACU169S DD not found ddname

Explanation: DD *ddname* not found allocated.

Action: The function terminates.

User response: None.

EQACU170S ISPEXec Error command rc=rc

Explanation: ISPEXec Command *command* failed with rc=rc.

Action: The function terminates.

User response: None.

EQACU171S Member in use member dsname

Explanation: Member *member* of data set *dsname* is already in use.

Action: The function terminates.

User response: None.

EQACU172S Member not found member dsname

Explanation: Member *member* of data set *dsname* could not be found.

Action: The function terminates.

User response: None.

EQACU173S Data set data set name has a DCB characteristic of invalid value, but it should be valid value

Explanation: The specified data set name has a DCB characteristic (such as LRECL) of *invalid value*. The JCL created expects this data set to be preallocated and therefore cannot change its DCB characteristics.

Action: The function terminates.

User response: Reallocate *data set name* with a DCB characteristic of *valid value*.

EQACU174I Reset of data set cntl canceled

Explanation: The reset of the control file was canceled by the user.

Action: The function terminates.

User response: None.

EQACU175S Invalid invocation: Missing or invalid operand

Explanation: A required operand was missing, or an invalid operand was specified in a program invocation.

Action: The function terminates.

User response: Correct the operand, and then restart.

EQACU176S Data set dsName not found

Explanation: The data set *dsName* was not found.

Action: The function terminates.

User response: Correct the data set name, and then restart.

EQACU177S Unable to create data set dsName

Explanation: The program was unable to create the data set *dsName*.

Action: The function terminates.

User response: Check for other messages or data set allocation or space problems, or contact your system programmer.

EQACU178S Data set *dsName* has a bad DSORG

Explanation: The specified data set has the wrong DSORG.

Action: The function terminates.

User response: Specify a data set that has the correct organization for the function.

EQACU179S Unable to alloc import data set *dsn*

Explanation: The import defaults data set could not be allocated.

Action: The Manipulate Defaults function terminates.

User response: Verify that the import defaults data set is accessible and the name is correct.

EQACU180S Unable to read import data set *dsn*

Explanation: The import defaults data set could not be read.

Action: The Manipulate Defaults function terminates.

User response: Ensure that the import defaults data set is accessible and is the proper format.

EQACU181S Unable to alloc export data set *dsn*

Explanation: The export data set to contain the defaults could not be allocated.

Action: The Manipulate Defaults function terminates.

User response: Verify that the export defaults data set is accessible and that the name is correct.

EQACU182S Unable to write export data set *dsn*

Explanation: The export defaults data set could not be written.

Action: The Manipulate Defaults function terminates.

User response: Verify that the export defaults data set is accessible and that it is the proper format.

EQACU183I Coverage Utility ISPF variables saved

Explanation: The ISPF variables have been saved.

Action: The Manipulate Defaults function continues.

User response: None.

EQACU184I Coverage Utility ISPF variable edit canceled

Explanation: The ISPF variables edit has been canceled

Action: The Manipulate Defaults function terminates.

User response: None.

EQACU185I Coverage Utility ISPF variables reset from system defaults

Explanation: The ISPF variables have been reset to the system defaults.

Action: The Manipulate Defaults function completes.

User response: None.

EQACU186I Coverage Utility ISPF variables imported from user data set

Explanation: The ISPF variables have been imported from your import data set.

Action: The Manipulate Defaults function completes.

User response: None.

EQACU187I Coverage Utility ISPF variables exported to user data set

Explanation: The ISPF variables have been exported to your export data set.

Action: The Manipulate Defaults function completes.

User response: None.

EQACU188S Unknown command *cmd*

Explanation: An unknown command was encountered when Coverage Utility tried to run the defaults function.

Action: The Manipulate Defaults function terminates.

User response: Contact Debug Tool support.

EQACU193I JCL generation canceled

Explanation: Errors were found before JCL generation that precluded it from being generated.

Action: Generation of the JCL was not done.

User response: Correct the errors indicated by previous messages, and then rerun the JCL.

EQACU194I You must link the instrumented objects into executables after running Setup JCL

Explanation: The instrumented object modules must be link edited before the program is run.

Action: Processing continues.

User response: Perform the required link edit prior to execution.

EQACU195S INTERNAL SOFTWARE ERROR: Literal *lll* was not found.

Explanation: An internal software error has occurred.

Action: Processing terminates.

User response: Contact your installer or system programmer.

EQACU196S Return code *rc* from IEANTCR creating Name/Token pair for NATLANG.

Explanation: The indicated return code was received from the IEANTCR MVS service.

Action: Processing terminates.

User response: Contact your installer or system programmer.

EQACU197S Return code *rc* from IEANTCR creating Name/Token pair for LOCALE.

Explanation: The indicated return code was received from the IEANTCR MVS service.

Action: Processing terminates.

User response: Contact your installer or system programmer.

EQACU198W The value specified for the NATLANG option is invalid or disabled.

Explanation: Either the value specified for the NATLANG option is not a supported value or this specification has been disabled when Coverage Utility was installed at your installation.

Action: Processing continues using the default NATLANG specification.

User response: Correct the indicated value.

EQACU199W The value specified for the Numeric Format suboption of the LOCALE option is invalid or disabled.

Explanation: Either the value specified for the third operand of the LOCALE option is not a supported value or this specification has been disabled when Coverage Utility was installed at your installation.

Action: Processing continues using the default LOCALE specification.

User response: Correct the indicated value.

EQACU200W The value specified for the Date Format suboption of the LOCALE option is invalid or disabled.

Explanation: Either the value specified for the first operand of the LOCALE option is not a supported

value or this specification has been disabled when Coverage Utility was installed at your installation.

Action: Processing continues using the default LOCALE specification.

User response: Correct the indicated value.

EQACU201W The value specified for the Time Format suboption of the LOCALE option is invalid or disabled.

Explanation: The value specified for the second operand of the LOCALE option is either not a supported value or this specification has been disabled when Coverage Utility was installed at your installation.

Action: Processing continues using the default LOCALE specification.

User response: Correct the indicated value.

EQACU202W The value specified for the LINECOUNT option is not a positive decimal number less than 99999.

Explanation: The value specified for the LINECOUNT option is either not a positive decimal number or it is too large.

Action: Processing continues using the default LINECOUNT specification.

User response: Correct the indicated value.

EQACU203W NATLANG, LOCALE, or LINECOUNT was specified as a parameter to an EXEC when already running under Coverage Utility ISPE.

Explanation: One of the indicated options was specified on a Coverage Utility command while running under the EQASTART command.

Action: Processing continues. The values specified or defaulted on the EQASTART command are used.

User response: Invoke the Coverage Utility command without the specified option, or invoke the command while not running under EQASTART.

EQACU204E Invalid parameter: *pppp*

Explanation: The indicated parameter is not valid.

Action: Processing terminates.

User response: Correct the indicated parameter.

EQACU205E Text not found from *iiii* invocation of EQACU6M with parms *pppp qqqq*

Explanation: The *iiii* REXX EXEC could not access its literal text from the Coverage Utility REXX message module (EQACU6Mn).

Action: Processing terminates.

User response: Contact your installer or system programmer.

EQACU206E Error RC= *rc* from IRXEXCOM by *iiii* invocation of EQACU6M with parms *pppp qqqq*

Explanation: The indicated return code was returned by IRXEXCOM while trying to set REXX variables for the *iiii* EXEC.

Action: Processing terminates.

User response: Contact your installer or system programmer.

EQACU207S No parameters were specified on the invocation of *iiii*.

Explanation: The *iiii* REXX EXEC was invoked with no parameters. One or more parameters are required.

Action: Processing terminates.

User response: Correct the invocation of the EXEC.

EQACU208S Unable to emulate ADDRESS LINKMVS for *program*

Explanation: Coverage Utility could not find the indicated program in the search path for run programs.

Action: Processing terminates.

User response: If you are running in the TSO foreground, ensure that the Coverage Utility load module library is specified correctly in defaults. In TSO batch, ensure that the Coverage Utility load library is available from either the JOBLIB or STEPLIB.

EQACU209S Return code *rc* from IDENTIFY macro loading NLS message table EQACU6M.

Explanation: The specified return code was issued by the MVS IDENTIFY macro while loading the Coverage Utility message table.

Action: Processing terminates.

User response: Contact your installer or system programmer.

EQACU210S Return code *rc* from ListDSI for *dsname*.

Explanation: The specified return code was issued by the LISTDSI command while processing the indicated data set.

Action: Processing terminates.

User response: Correct the data set specification, and then rerun the command.

EQACU214S Combine: Combine requires at least 2 input files.

Explanation: Fewer than two input files were specified for combine.

Action: The combine job is terminated. No output is produced.

User response: Provide two input files, and then rerun combine.

EQACU215E Input listing *membername* was specified in more than one control statement.

Explanation: The same input listing name appeared on more than one control statement. This duplication is not permitted when placing breakpoints in object modules.

Action: The job is terminated.

User response: Correct the control statement, and then retry.

EQACU216S Debug Tool Utilities and Advanced Functions is required to use this function.

Explanation: The Debug Tool Utilities and Advanced Functions product is either not installed or is not registered on this CPU.

Action: The function or job is terminated.

User response: Contact your system programmer or installer. Debug Tool Utilities and Advanced Functions must be ordered, installed, and registered in addition to the base Debug Tool product.

EQACU217S Debug Tool Utilities and Advanced Functions is required to use this function.

Explanation: The Debug Tool Utilities and Advanced Functions product is either not installed or is not registered on this CPU.

Action: The function or job is terminated.

User response: Contact your system programmer or installer. Debug Tool Utilities and Advanced Functions must be ordered, installed, and registered in addition to the base Debug Tool product.

Translation: This message is not translated.

| **EQACU218S Short SVC number already in use by**
| **another SVC.**

| **Explanation:** The SVC number that was supplied for
| the short operation code (opcode parameter), which is
| the first number passed to the EQACUOIN program, is
| used by another product.

| **Action:** The monitor installer (EQACUOIN)
| terminates.

| **User response:** Ensure that the first parameter that is
| passed to the monitor installer (EQACUOIN program)
| in the JCL is a hexadecimal number between C8 and FF
| without any surrounding quotation marks, and that the
| SVC number is not used by another product.

| **Translation:** This message is not translated.

| **EQACU219S Long SVC number already in use by**
| **another SVC.**

| **Explanation:** The SVC number that was supplied for
| the long operation code (opcode parameter), which is
| the second number passed to the EQACUOIN program,
| is used by another product.

| **Action:** The monitor installer (EQACUOIN)
| terminates.

| **User response:** Ensure that the second parameter that
| passed to the monitor installer (EQACUOIN program)
| in the JCL is a hexadecimal number between C8 and FF
| without any surrounding quotation marks, and that the
| SVC number is not in use by another product.

| **Translation:** This message is not translated.

**EQACU999S Internal Error: REXX NLS messages not
loaded**

Explanation: EQACU6M*n* could not be found in the
standard search path. This problem is probably caused
by an installation error. These modules are normally
found in the SEQABMOD load library.

Action: The job is terminated.

User response: Contact your installer or system
programmer.

Appendix B. Resources and requirements

This section lists the resources that you need to set up and run Coverage Utility and to produce reports. Required data set attributes and data set definition (DD) names are also provided.

- “Coverage Utility resources”
- “Coverage Utility requirements” on page 188

Coverage Utility resources

The system resources given in the following topics are required by Coverage Utility:

- “Setup resources”
- “Monitor CSA, SQA, and ECSA usage” on page 188
- “Report programs” on page 188

Setup resources

Coverage Utility provides the following programs used during the setup process:

- EQACUSET creates the breakpoint data file (BRKTAB). It requires the following resources:

Description	Requirements
Disk space needed for the breakpoint table (BRKTAB) upon completion of EQACUSET	To determine the size of the BRKTAB data set in bytes, use the following formula: $128 + (64 + \text{PA name length}) \times \text{number of PAs} + 32 \times \text{number of breakpoints}^1$
Disk space needed for the breakout table (BRKOUT) upon completion of EQACUSET	$128 + (\text{number of breakpoints} \div 8)$
1. For COBOL, PL/I and C, there is approximately one breakpoint per high-level executable statement. For assembler, there are approximately two breakpoints per assembler instruction that changes program flow (that is, branch instructions).	

- EQACUZPT modifies user object modules to insert breakpoints.
- EQACUZPL builds input to enable EQACUZPP to insert breakpoints in user load modules.
- EQACUZPP modifies user load modules and program objects to insert breakpoints.

Monitor CSA, SQA, and ECSA usage

Because the monitor must be able to trace programs in any address space, it is loaded in CSA and SQA space and uses ECSA storage for tables. The monitor uses the following system storage when it is installed:

- Fixed amount when monitor is installed and enabled:
 - CSA 13248 bytes
 - SQA 25496 bytes

Values: Each of these values is approximate.

- Each session started uses the following storage in ECSA: $32932 + (\text{number of breakpoints} \times 52) + (\text{number of compile units} \times 128) + (\text{number of program areas} \times 96)$

Breakpoints are placed at the start of each high-level instruction for COBOL, PL/I, and C/C++ and at conditional branches in high-level statements (except C/C++) that can cause a change of program flow (IF, DO WHILE, PERFORM, and others).

If you want an exact count of break points (BPs) for any given session, count the number of VERIFY or REPLACE statements in the RPTMSGs DDs in the setup job, or issue a EQACUOSA command for a running monitor session and look at the BPS TOTALS column.

If you want an exact count of program areas (PAs) for any given session, look at the last entry in the summary report Program Area Data section for the highest PA number. The number of PAs in a session is also in the annotated listing report.

Report programs

Coverage Utility provides three report programs:

- EQACUSUM produces a summary report.
- EQACURPT produces an annotated listing report.
- EQACUXML exports an XML file.

The reports programs require the following resources:

Description	Requirements
Disk space needed for the summary report on completion of EQACUSUM	Negligible
Disk space needed for the annotated listing report on completion of EQACURPT program	Size of listing
Disk space needed for the export XML program on completion of EQACUXML	Dependant on the number of breakpoints and options used

Coverage Utility requirements

Coverage Utility requires several data sets for input and output. This section identifies each data set by function and ddname and provides the attributes for each data set.

DDNAME requirements

Then data sets required by Coverage Utility are as follows:

DDNAME	Description
BRKOUT	Contains Coverage Utility test case results
BRKTAB	Contains all breakpoint data used by the monitor program
LIBIN	Load modules or program objects input to EQACUZPP. A concatenation of data sets can be specified.
LIBOUT	Load modules or program objects that contain breakpoints output by EQACUZPP
LISTINB	Input listing from COBOL
LISTINP	Input listing from PL/I or C/C++
LISTINA	Input listing from ASM
LISTOUT	Output from EQACURPT program (annotated listing report)
SUM	Output from EQACUSUM program
SYSIN	Input commands for EQACUZPP. These are a subset of the control statements processed by AMASPZAP.
SYSPRINT	Output listing generated by EQACUZPP

Data set attributes

Coverage Utility uses several data sets, whose requirements are as follows:

DDNAME	DSORG	LRECL	BLKSIZE	RECFM
BRKOUT	PS or PO	256	27904 ¹	FB
BRKTAB	PS or PO	256	27904 ¹	FB
Coverage Utility	PS or PO	255	27998 ¹	VB
LIBIN	PO	n/a	any	U
LIBOUT	PO	n/a	any	U
LISTINB ²	PS or PO	133	27930 ¹	FBA
LISTINP	PS or PO	125 for OS PL/I or PL/I for MVS & VM	27998 ¹	VBA
		132 for VisualAge PL/I, Version 2 Release 2, and Enterprise PL/I for z/OS and OS/390 Version 3 Release 1	27998 ¹	
		137 for OS/390 C/C++ compiler and Enterprise PL/I for z/OS and OS/390 Version 3 Release 2 or later	27998 ¹	
LISTINA	PS or PO	133 for High Level Assembler	27930 ¹	FBM
		121 for Assembler H	27951 ¹	
LISTOUT	PS or PO	133	27930 ¹	FBA

DDNAME	DSORG	LRECL	BLKSIZE	RECFM
SUM	PS or PO	133	27930 ¹	FBA
SYSIN	PS or PO	80	any	FB
SYSPRINT	PS or PO	137	any	VBA
Notes:				
1. Any valid BLKSIZE can be used.				
2. LRECL=121, BLKSIZE=12100 (or smaller) for OS/VS COBOL listings.				

Appendix C. DBCS support

This section describes Coverage Utility *DBCS (double-byte character set)* support. Support for DBCS varies among tools. The following topics give the support for the compilers and assemblers and for Coverage Utility itself:

- “DBCS requirements for Coverage Utility compilers and assemblers”
- “DBCS support with Coverage Utility”

DBCS requirements for Coverage Utility compilers and assemblers

The compilers and assemblers supported by Coverage Utility implement DBCS support that is consistent with the following rules:

1. DBCS characters are delimited by a leading Shift Out (0x0E) byte and a trailing Shift In (0x0F) byte.
2. There must be an even number of bytes between Shift Out and Shift, which have values between 0x41 and 0xFE (except the DBCS space 0x4040).
3. Each compiler and assembler has an option that tells the tool whether DBCS is to be recognized as such (also referred to as *enabled*).

The following rules apply to identifiers in COBOL and PL/I only:

1. In identifiers, all lowercase DBCS EBCDIC (0x42 in the first byte) are converted to uppercase. However, PL/I permits DBCS EBCDIC characters in keywords.
2. If an identifier is all DBCS EBCDIC, it is converted to its SBCS (single-byte character set) equivalent.
3. If an identifier contains one or more non-EBCDIC DBCS characters, the entire identifier is converted to its DBCS representation.
4. Any DBCS, SBCS, or mixed DBCS and SBCS identifiers that convert to the same identifier by the previous rules are considered equivalent.
5. The DBCS EBCDIC form of a character is permitted in an identifier name only if the SBCS version is permitted.

DBCS support with Coverage Utility

DBCS support is implemented in the following ways by Coverage Utility:

- Control files accept DBCS identifiers and DBCS strings within comments.
- Coverage Utility provides DBCS support that is consistent with support provided by the compilers (except that the DBCS space is not supported within the control files) except in comments.
- DBCS in control cards is always enabled.
- For COBOL and PL/I, DBCS identifiers are normalized according to the rules in “DBCS requirements for Coverage Utility compilers and assemblers” before any comparisons are performed.
- DBCS identifiers in all outputs are in the normalized form.
- All control file keywords, delimiters, and MVS data set names must be entered in SBCS.
- The COBOL LANGUAGE(JAPANESE) compiler option is permitted.

Appendix D. FastPath

Coverage Utility FastPath provides a streamlined way to gather coverage information for your programs. FastPath condenses the process into three steps:

1. Quick start

Quick start combines the setup and start monitor steps into one step. For programs that consist of one source file, you can enter all the required information on the panels rather than in a control file.

2. Snapshot summary

Use snapshot summary to check your progress without disrupting your testing.

3. Quick stop

Quick stop stops the monitor session and automatically generates a summary report. The report indicates the overall effectiveness of your test suite and can help you decide what annotated listing reports to generate.

FastPath minimizes the steps for average users to run Coverage Utility. Advanced users, or those with atypical situations, might prefer to use the standard panels in some cases.

To display the FastPath panel, select option 6 from the Debug Tool Coverage Utility panel. The FastPath panel is displayed:

```
----- FastPath -----  
Option ==>  
1 Quick Start   Create JCL for Quick Start  
2 Snapshot     Create JCL for Snapshot Summary  
3 Quick Stop   Create JCL for Stop & Summary  
  
Enter END to Terminate
```

The options on this panel are as follows:

Quick Start

Generates JCL to set up and start a monitor session.

Snapshot

Generates JCL to create a summary for a monitor session without stopping the session.

Quick Stop

Generates JCL to stop a monitor session and generate a summary report.

Related tasks

“Creating quick start JCL from the panels”

“Creating snapshot summary JCL from the panels” on page 198

“Creating quick stop JCL from the panels” on page 200

Creating quick start JCL from the panels

Quick start combines the setup and start monitor session steps. Besides reducing the number of steps involved in running Coverage Utility, quick start also ensures that you start the monitor session with the most recent version of your input data sets. If your programs consist of a single object per load module, and you use the

same member name for the listings, object modules, and load modules, you can provide all the required information directly on the panels rather than in a control file. For more complex situations, you can use a Coverage Utility control file to drive quick start.

For the following cases, you can be more efficient by starting and stopping the monitor session by using the normal Coverage Utility setup, start session, and stop session commands:

- Testing a large program that does not change over a period of time
- Collecting coverage per test case

Quick start always analyzes your listings or load modules to create a new instrumented program and breakpoint data. You do not need to do this, if your program does not change. For large programs that do not change, you can save time by not using quick start and doing this setup analysis.

If you want to collect coverage per test case (for example, to determine the most efficient test case bucket), you can use the EQACUOSN and EQACUORE commands after each test case. The quick stop command always stops the session and produces a summary report. The Snapshot command produces a cumulative summary of testing up to that point, but does not reset coverage statistics.

If you are instrumenting object modules, you must link them into executable load modules after you run quick start and before you run your tests. Also, if you are instrumenting load modules in place (which is not recommended), you must ensure that quick start did not previously instrument your load module.

To set up and start a monitor session to measure code coverage, select option 1 from the FastPath panel. The Create JCL for Quick Start panel is displayed:

```
----- Create JCL for Quick Start -----
Option ==>

1 Generate      Generate JCL from parameters
2 Edit          Edit JCL
3 Submit        Submit JCL
4 Parameters    Quick Start Parameters

Enter END to Terminate

Session ID . . . . . YOUNG

Programming Language. . COBOL   (COBOL|PL/I|C|ASM)

CA Control File (optional):
Control File Dsn. . .
  (If the Control File is specified, any program names below are ignored.)

Program Names:
Primary Program Name. COB01

Additional Programs . COB02          COB03
                    COB04          COB05
                    COB06          COB07
                    COB08          COB09
                    COB10         COB11
                    COB12         COB13
                    COB14         COB15
```

The options and fields on this panel are as follows:

Generate

Generates JCL from the parameters that you specify on the panel.

Edit Displays an ISPF edit session where you can make changes to existing JCL.

Submit

Submits for execution the JCL that you specify in the **JCL Dsn** field on this panel.

Parameters

Displays a panel that enables you to set additional parameters used in the JCL generation.

Session ID

An ID session for your session. This ID defaults to your TSO user ID. Multiple testers can use the monitor simultaneously.

Programming Language

The programming language of your source code. The following values are accepted:

- COBOL
- PL/I or PLI
- C or C++ or CPP (all of these are equivalent)
- ASM

Control File Dsn (optional)

The name of the control file data set that contains the locations of the listing files, the object or load module libraries, and the libraries to copy the instrumented versions into, and specifies the members to be processed.

If you specify the **Control File Dsn**, the **Primary Program Name** is used for generated program names only if **Use Program Name for File Name** (available on the Quick Start Parameters panel) is Yes. In this case, quick start ignores the **Additional Programs** field.

Primary Program Name

The name of a program that is monitored by quick start. Generated data set names also use this name when **Use Program Name for File Name** (available on the Quick Start Parameters panel) is Yes.

Restriction: The listing, object modules, and load modules must all have the same member name, and you must have only one object per load module in order to run FastPath using program names. Otherwise, you must use a control file.

Additional Programs

The names of other programs that are monitored by quick start.

Related tasks

“Running multiple user sessions” on page 80

Quick start parameters

Use the Quick Start Parameters panel to provide information about the data sets to be used for setup and the monitor session. If you do not specify a control file, some of the fields on this panel are required. If the fields are not filled in, this panel is displayed the first time that you request JCL generation.

```

----- Quick Start Parameters -----
Command ==>

Enter END (to Exit and Save changes) or CANCEL (to Exit without saving)

Session ID . . . . . YOUNG

Programming Language. . COBOL   (COBOL|PL/I|C|ASM)

Object or Load. . . . . LOAD    (OBJ|LOAD)

Compiler Listing. . . . 'YOUNG.SAMPLE.COBOLST'

Original Library. . . . 'YOUNG.SAMPLE.LOADLIB'

Instrumented Library. . 'YOUNG.SAMPLE.RUNLIB'

Control File:
  Control File Dsn. . .

Use Program Name for File Name YES (Yes|No) Program Name COB01

JCL Library and Member:
  JCL Dsn . . . . . 'YOUNG.SAMPLE.JCL(SCOB01)'

Breakpoint Table:
  Breakpoint Table Dsn. 'YOUNG.SAMPLE.COB01.BRKTAB'

Monitor Output File:
  Breakout Dsn. . . . . 'YOUNG.SAMPLE.COB01.BRKOUT'

```

The options and fields on this panel are as follows:

Session ID

An ID for your session. This ID defaults to your TSO user ID. Multiple testers can use the monitor simultaneously.

Programming Language

The programming language of your source code. The following values are accepted:

- COBOL
- PL/I or PLI
- C or C++ or CPP (all of these are equivalent)
- ASM

Object or Load

The type of setup to be done. You can instrument object modules or load modules. If you instrument object modules, you must link the instrumented objects into executable load modules after you run quick start, and before you run your tests.

Instrumenting of load modules is not supported for VisualAge PL/I, Version 2 Release 2, and Enterprise PL/I for z/OS and OS/390.

When required:

This name is required if you do not specify the **Control File Dsn**.

Compiler Listing

The name of the data set that contains the compiler listings for the programs to be monitored.

When required: This name is required if you do not specify the **Control File Dsn**.

Original Library

The name of the data set that contains the object or load modules to be instrumented. This field is required only if you instrument object modules. If you instrument load modules in place (this is not recommended), you can leave this field blank and enter the name of data set that contains the load modules in the **Instrumented Library** field.

When required: This field is required if you do not specify the **Control File Dsn**.

Instrumented Library

The name of the data set where Coverage Utility stores the instrumented copies of the object or load modules.

When required: This name is required if you do not specify the **Control File Dsn**.

Control File Dsn (optional)

The name of the control file data set that contains the locations of the listing files, the object or load module libraries, and the libraries to copy the instrumented versions into, and specifies the members to be processed.

If you specify the **Control File Dsn**, the following fields are ignored: **Object or Load**, **Compiler Listing**, **Original Library**, and **Instrumented Library**.

Use Program Name for File Name

Enter Yes if you want to construct data set names from the default high-level qualifier, the specified program name, and the default low-level qualifier for the following data sets:

- JCL Dsn
- Breakpoint Table Dsn
- Breakout Dsn

When you press Enter, the affected data set names on the panel change automatically. Coverage Utility normally uses the program name to construct the data names.

Program Name

The name to use for Coverage Utility files when you enter Yes in the **Use Program Name for File Name** field. This name can be any valid name; it does not have to be the name of any of your programs. Names of the following form are created:

- Sequential data sets:

`'proj_qual.program_name.file_type'`

For example: `'YOUNG.SAMPLE.COB01.BRKTAB'`

- Partitioned data sets:

`'proj_qual.file_type(program_name)'`

For example: `'YOUNG.SAMPLE.BRKTAB(COB01)'`

JCL Library and Member

The name of the JCL library data set and member to hold the generated JCL.

Default: If you set the **Use Program Name for File Name** field to Yes, then the member name or program name qualifier of the data set will be Txxxxxxx, where xxxxxx is the last seven characters of the program name.

Breakpoint Table Dsn

The name of the BRKTAB data set that is created during setup and used by the monitor program.

Breakout Dsn

The name of the BRKOUT data set that is created during execution and used by the summary and report programs.

Related tasks

“Running multiple user sessions” on page 80

Creating snapshot summary JCL from the panels

A snapshot summary is a summary report that is generated against the data collected by a running monitor session, without stopping the session. You can use snapshot summaries to check the status of your testing efforts. By issuing the EQACUORE command after each snapshot summary, you can generate an independent summary report for each test in your test suite without stopping and restarting the monitor. Otherwise, each summary is the cumulative result of all testing since the monitor session was started.

Saving data: Snapshot summary does not save the coverage data for a snapshot. If you want to save the coverage data, particularly if you are issuing EQACUORE, use the snapshot command from the Control the Monitor panel to generate a BRKOUT data set that contains the raw coverage data.

To create JCL to generate a snapshot summary, select option 2 from the FastPath panel. The Create JCL for Snapshot Summary panel is displayed:

```
----- Create JCL for Snapshot Summary -----
Option ==>

1 Generate      Generate JCL from parameters
2 Edit          Edit JCL
3 Submit        Submit JCL

Enter END to Terminate

Session ID. . . . . YOUNG

Use Program Name for File Name YES (Yes|No) Program Name COB01

Input Files:
Breakpoint Table Dsn. 'YOUNG.SAMPLE.COB01.BRKTAB'

JCL Library and Member:
JCL Dsn . . . . . 'YOUNG.SAMPLE.JCL(TCOB01)'

Output Summary Type and File:
Type. . . . . INTERNAL (Internal|External)
Inline . . . . . N (I|N)
Report Dsn . . . . . 'YOUNG.SAMPLE.COB01.SUMMARY'
(* for default sysout class)
```

The options and fields on the panel are as follows:

Generate

Generates JCL from the parameters that you specify on the panel.

Edit Displays an ISPF edit session where you can change existing JCL.

Submit

Submits for execution the JCL that you specify in the **JCL Dsn** field on this panel.

Session ID

An ID for your session. This ID defaults to your TSO user ID. Multiple testers can use the monitor simultaneously.

Use Program Name for File Name

Enter Yes if you want to construct the data set names from the default high-level qualifier, the specified program name, and the default low-level qualifier for each data set.

When you press Enter, the file names on the panel change automatically. Coverage Utility normally uses the program name to construct the data set names.

Program Name

The name to use for Coverage Utility files when you enter Yes in the **Use Program Name for File Name** field. This name can be any valid name; it does not have to be the name of any of your programs. Names of the following form are created:

- Sequential data sets:

'proj_qual.program_name.file_type'

For example: 'YOUNG.SAMPLE.COB01.BRKTAB'

- Partitioned data sets:

'proj_qual.file_type(program_name)'

For example: 'YOUNG.SAMPLE.BRKTAB(COB01)'

Breakpoint Table Dsn

The name of the BRKTAB data set that is created during setup and used by the monitor program.

JCL Library and Member

The name of the JCL library data set and member to hold the generated JCL.

Default: If you set the **Use Program Name for File Name** field to Yes, then the member name or program name qualifier of the data set will be Txxxxxxx, where xxxxxx is the last seven characters of the program name.

Type The type of summary report to be produced.

Internal

The report contains information about each program area.

External

Report contains information with all program areas combined.

This option is ignored for assembler program areas.

Inline For languages for which Coverage Utility supports optimized code, the summary processor might include or ignore counts and percentages from inline code.

- I Include all lines of inline code in the summary counts and percentages.
- N Do **not** include inline code in the summary counts and percentages.

Report Dsn

The name of the data set to contain the summary report.

Related concepts

“The effects of inlining” on page 136

Related tasks

“Running multiple user sessions” on page 80

Creating quick stop JCL from the panels

Use quick stop to generate JCL to stop a monitor session and generate a summary report in one step. This is convenient, because you will normally want to run a summary report on the overall coverage. You do not need to use quick start to use quick stop.

To create JCL to stop a monitor session and generate a summary report, select option 3 from the FastPath panel. The Create JCL for Stop & Summary panel is displayed:

```

----- Create JCL for Stop & Summary -----
Option ==>

1 Generate      Generate JCL from parameters
2 Edit         Edit JCL
3 Submit       Submit JCL
4 Parameters   Quick Stop Parameters

Enter END to Terminate

Session ID. . . . . YOUNG

Use Program Name for File Name YES (Yes|No) Program Name COB01

Output Summary Dsn:
  Report Dsn . . . . . 'YOUNG.SAMPLE.COB01.SUMMARY'
  (* for default sysout class)

```

The options and fields on the panel are as follows:

Generate

Generates JCL from the parameters that you specify on the panel.

Edit

Displays an ISPF edit session where you can change existing JCL.

Submit

Submits for execution the JCL that you specify in the **JCL Dsn** field on this panel.

Parameters

Displays a panel that enables you to set additional parameters used in the JCL generation.

Session ID

An ID for your session. This ID defaults to your TSO user ID. Multiple testers can use the monitor simultaneously.

Use Program Name for File Name

Enter Yes if you want to construct the data set names from the default high-level qualifier, the specified program name, and the default low-level qualifier for each data set.

When you press Enter, the file names on the panel are changed automatically. Coverage Utility normally uses the program name to construct the data set names.

Program Name

The name to use for Coverage Utility files when you enter Yes in the **Use Program Name for File Name** field. This name can be any valid name; it does not have to be the name of any of your programs. Names of the following form are created:

- Sequential data sets:

```
'proj_qual.program_name.file_type'  
For example: 'YOUNG.SAMPLE.COB01.BRKTAB'
```

- Partitioned data sets:

```
'proj_qual.file_type(program_name)'  
For example: 'YOUNG.SAMPLE.BRKTAB(COB01)'
```

Report Dsn

The name of the data set to contain the summary report.

Related tasks

“Running multiple user sessions” on page 80

Quick stop parameters

Use the Quick Stop Parameters panel to specify additional information for the data sets to be used for the output data and report, and the type of summary report produced.

```
----- Quick Stop Parameters -----  
Command ==>  
  
Enter END to Exit and Save changes  
  
Session ID. . . . . YOUNG  
  
Use Program Name for File Name YES (Yes|No) Program Name COB01  
  
Input Files:  
Breakpoint Table Dsn. 'YOUNG.SAMPLE.COB01.BRKTAB'  
Breakout Dsn. . . . . 'YOUNG.SAMPLE.COB01.BRKOUT'  
  
JCL Library and Member:  
JCL Dsn . . . . . 'YOUNG.SAMPLE.JCL(TCOB01)'  
  
Output Summary Type and File:  
Type. . . . . INTERNAL (Internal|External)  
Inline . . . . . N (I|N)  
Report Dsn . . . . . 'YOUNG.SAMPLE.COB01.SUMMARY'  
(* for default sysout class)
```

The options and fields on this panel are as follows:

Session ID

An ID for your session. This ID defaults to your TSO user ID. Multiple testers can use the monitor simultaneously.

Use Program Name for File Name

Enter Yes if you want to construct the data set names from the default high-level qualifier, the specified program name, and the default low-level qualifier for each data set.

When you press Enter, the file names on the panel change automatically. Coverage Utility normally uses the program name to construct the data set names.

Program Name

The name to use for Coverage Utility files when you enter Yes in the **Use Program Name for File Name** field. This name can be any valid name; it does not have to be the name of any of your programs. Names of the following form are created:

- Sequential data sets:

'proj_qual.program_name.file_type'
For example: 'YOUNG.SAMPLE.COB01.BRKTAB'

- Partitioned data sets:

'proj_qual.file_type(program_name)'
For example: 'YOUNG.SAMPLE.BRKTAB(COB01)'

Breakpoint Table Dsn

The name of the BRKTAB data set that is created during setup and used by the monitor program.

Breakout Dsn

The name of the BRKOUT data set that is created during execution and used by the summary and report programs.

JCL Library and Member

The name of the JCL library data set and member to hold the generated JCL.

Default: If you set the **Use Program Name for File Name** field to Yes, then the member name or program name qualifier of the data set will be Txxxxxxx, where xxxxxx is the last seven characters of the program name.

Type The type of summary report to be produced.

Internal

The report contains information about each program area.

External

The report contains information with all program areas combined.

This option is ignored for assembler program areas.

Inline For languages for which Coverage Utility supports optimized code, the summary processor might include or ignore counts and percentages from inline code.

I Include all lines of inlined code in the summary counts and percentages.

N Do **not** include inline code in the summary counts and percentages.

Report Dsn

The name of the data set to contain the summary report.

Related concepts

"The effects of inlining" on page 136

Related tasks

“Running multiple user sessions” on page 80

Appendix E. Parameters that are common to multiple routines

There are three parameters, described in this section, that are common to multiple Coverage Utility routines:

- "LINECOUNT" on page 206
- "LOCALE" on page 206
- "NATLANG" on page 208

You can specify these parameters in several different contexts:

- Part of a blank-delimited or comma-delimited PARM string for various Coverage Utility programs
- Blank-delimited operands on various TSO commands, such as EQASTART

When these parameters are passed to Coverage Utility programs in the PARM field, they must come after any parameters that are specific to a particular routine. Generally, the specific parameters must appear in the order that is described in the documentation for the particular routine.

The following general syntax rules apply to LINECOUNT, LOCALE, and NATLANG (but do not necessarily apply to parameters specific to a routine):

1. Parameters can appear in any order.
2. When passed as operands to the following commands or as parameters to the following commands and programs, each parameter must be separated from preceding and following parameters by one or more blanks:
 - EQACUOBP
 - EQACUOSA
 - EQACUOSE
 - EQACUOSL
 - EQACUOSP
 - EQACURPT
 - EQACUSUM
 - EQASTART

When passed as parameters to the following commands and programs, each parameter must be separated from preceding and following parameters by a comma:

- EQACUOCM
 - EQACUOID
 - EQACUOPF
 - EQACUOPN
 - EQACUOQT
 - EQACUORE
 - EQACUOSN
 - EQACUSET
 - EQACUZPL
 - EQACUZPP
 - EQACUZPT
3. Each parameter is identified by a keyword. The keyword can be abbreviated by shortening it to the minimum number of characters required to make it unique from other valid keywords.
 4. If a value is expected for the keyword, the value must be enclosed in parentheses following the keyword.

5. If multiple values for a keyword are specified, each of the values must be separated by a comma.
6. Blanks can appear before or after commas or parentheses.

You can specify LINECOUNT, LOCALE, and NATLANG and their associated values wherever the *common_parameters* are permitted. If you do not specify one or more of these keywords, a default value is used.

Changed defaults: Although the standard default values are indicated below, these defaults can be changed when Coverage Utility is installed on your machine.

If LINECOUNT, LOCALE, or NATLANG is specified on an invocation of EQASTART, these specifications are remembered by EQASTART and become the default if the parameters are not specified on subsequent invocations of EQASTART. In addition, when any of the other TSO commands are invoked under an ISPF session initiated by EQASTART, these parameters *cannot be specified on the TSO commands*. In this case, the commands use the values specified or defaulted on the EQASTART invocation. Similarly, when any of the monitor commands are run under ISPF but not under EQASTART, they remember these specifications and use them as the default the next time one of the monitor commands is run under ISPF.

Whenever LINECOUNT, LOCALE, or NATLANG is permitted, all are accepted and used in any appropriate situation. However, the corresponding specification is used only when applicable. For example, all operands of LOCALE are accepted even if the program never displays a date or time value.

LINECOUNT

The LINECOUNT parameter controls the number of lines (including headings) printed on a page. Not all output files contain carriage control characters, and some files that do contain control characters are not created with pages of a predetermined size. Hence, not all output files will be governed by this parameter.

▶▶—LINECOUNT—(—⁶⁶—*line_ct*—)————▶▶

where

line_ct

A decimal number less than 99999.

LOCALE

The LOCALE parameter specifies the way dates, times, and numbers are to be formatted. The syntax of this parameter is:

▶▶—LOCALE—(—¹—*date_fmt*—, —¹—*time_fmt*—, —¹—*numeric_fmt*—)————▶▶



where

date_fmt

The format for the parts of a date, including the sequence and separators. In addition to the following predefined formats, installation-specific formats might be available. See your local installer for information about installation-specific date formats.

- 1 MM/DD/YYYY
- 2 MM/DD/YY
- 3 DD/MM/YYYY
- 4 DD/MM/YY
- 5 YY.DDD
- 6 YYYY.DDD
- 7 YYYY/MM/DD
- 8 YYYYMMDD
- 9 June DD, YYYY
- 10 Jun DD, YYYY
- 11 DD-Jun-YYYY
- 12 DD June YYYY
- 13 JUNE DD, YYYY
- 14 JUN DD, YYYY
- 15 DD-JUN-YYYY
- 16 DD JUNE, YYYY

time_fmt

The format for hours, seconds, and minutes. In addition to the following predefined formats, installation-specific formats might be available. See your local installer for information about installation-specific time formats.

- 1 HH:MM:SS
- 2 HH:MM:SSam
- 3 HH:MM:SSAM

numeric_fmt

The format for numbers with regard to commas and periods. In addition to the following predefined formats, installation-specific formats might be available. See your local installer for information about installation-specific numeric formats.

- 1 1,234,567.89
- 2 1.234.567,89

NATLANG

The NATLANG parameter specifies that national language to be used to display program messages. The syntax of this parameter is:



language_id

One of the following IDs:

ENU English

UEN Uppercase English

JPN Japanese

Feature needed: JPN is not a valid choice unless the JPN feature of Debug Tool Utilities and Advanced Functions has been installed.

KOR Korean

Feature needed: KOR is not a valid choice unless the KOR feature of Debug Tool Utilities and Advanced Functions has been installed.

Example: Common parameters

The following examples show how the parameters LINECOUNT, LOCALE, and NATLANG might be passed to the Coverage Utility routine EQACUSUM:

1. This example requests LOCALE date format 2, the default LOCALE time format, LOCALE numeric format 2, and a LINECOUNT of 72.

```
//STEP1 EXEC PGM=EQACUSUM,PARM='I N N LOC(2,,2) LINEC(72)'
```

2. This example requests national language ENU and LOCALE time format 2.

```
//STEP1 EXEC PGM=EQACUSUM,PARM='I N N NATL(ENU) LOCALE(,2)'
```

These examples show how these parameters might be passed to EQASTART:

1. This example requests LOCALE date format 2, the default LOCALE time format, LOCALE numeric format 2, and a LINECOUNT of 72.

```
EQASTART LO(2,,2) LINE(72)
```

2. This example requests national language ENU and LOCALE time format 2.

```
EQASTART NATL(ENU) LOCALE(,2)
```

Appendix F. Exported XML data

Although Coverage Utility produces reports that explain the coverage data collected, you might want to process this data using other programs. You might even want to write your own programs to do specialized types of postprocessing of this data.

To do this processing or postprocessing, you must export the coverage data collected by the monitor in a format that can be easily parsed and understood by other programs. Coverage Utility supports the exporting of this data in XML format. This format is widely used, self-defining, and portable among applications.

If you want to write your own programs to process coverage data, use the export data function to output the coverage data in an XML format that can be used as input to other programs. The format and content of this XML file and the supplied DTD are given in the following sections.

Related references

“XML file description”

“XML DTD” on page 215

XML file description

This section describes the contents of the XML file. The following table shows the XML tags, attributes, and data that can appear in the exported file together with the associated data.

XML tag and attributes	Notes	Description
<?xml version="1.0" standalone="no"?>		Standard header
<!DOCTYPE CoverageFile SYSTEM "EQACUDTD.dtd">		Reference DTD under standard name.
<CoverageFile Version="1.0">		One of these per file.
<RunDate>		Date that the coverage data was saved.
<Year>2001</Year>		
<Month>04</Month>		
<Day>24</Day>		
</RunDate>		
<RunTime>		Time that the coverage data was saved
<Hours>10</Hours>		
<Minutes>14</Minutes>		
<Seconds>22</Seconds>		
</RunTime>		
<LoadModule>		One or more load modules per CoverageFile. The same load module can appear more than once.
<MemberName> <i>loadmod</i> </MemberName>		The name of the load module (member). One to eight characters.
<Listing		One or more listings per LoadModule.

XML tag and attributes	Notes	Description
Language="COBOL PLI C Assembler "		This keyword value indicates the language used in the source that generated the listing.
>		
<DSName>dsname</DSName>		The name of the data set that contains the listing.
<CompilationUnit		One CompilationUnit per Listing. (Coverage Utility does not currently support processing of more than one compilation unit per listing.)
Source="Listing"		Listing indicates that the compiler listing was used to breakpoint this CompilationUnit.
Mode="Performance Branch"		Performance indicates that only basic branch data is available (whether or not a breakpoint or statement was executed). If Branch is in effect, information about whether branches were taken or fallen-through is also available.
Counts="Executed Frequency"		Executed indicates that only information as to whether a breakpoint or statement is or is not executed is kept. If Frequency is in effect, the number of times that each statement or breakpoint was executed is also available.
NumberPAs="22"		The number of program areas in this compilation unit.
NumberBPs="55"	P	The number of breakpoints in this compilation unit.
ExecutedBPs="33"	P	The number of breakpoints in this compilation unit that were executed.
>		
<CreateDate>		The date that the program was instrumented for coverage. If a listing data set is modified after this date, it might no longer match this run.
<Year>2001</Year>		
<Month>04</Month>		
<Day>23</Day>		
</CreateDate>		
<CreateTime>		The time that the program was instrumented for coverage. If a listing data set is modified after this time, it might no longer match this run.
<Hours>10</Hours>		
<Minutes>14</Minutes>		
<Seconds>22</Seconds>		
</CreateTime>		
<TestId>idstring</TestId>		An identification string supplied by user. (optional)
<CSECT>		The name of the CSECT that contains the following program areas. There is one of these names per compilation unit.

XML tag and attributes	Notes	Description
<ExtName>csect_name</ExtName>		The name of the CSECT that contains the following program areas. The meaning of this field varies slightly depending on the source language, as follows: <ul style="list-style-type: none"> • COBOL - External PROGRAM ID • Assembler - first CSECT name • PLI - External Procedure name • C - CSECT name Exceptions: <ul style="list-style-type: none"> • For C code with NOCSECT and no <i>#pragma csect</i> in effect and for Assembler code where the first CSECT name is blank, this tag is not present. • For Assembler, only the first CSECT in the CompilationUnit is shown. Subsequent CSECTs are included in the form of ProgramAreas.
<ProgramArea		The name of the program area. There is one or more of these names per CSECT.
NumberBPs="22"	P	The number of breakpoints in this PA.
BPsExecuted="1440"	P	The total number of breakpoints executed in this PA.
BPExecutions="2222"	P	The total of the number of times the breakpoint was executed for all breakpoints in this PA.
>		
<PAName>PA_name</PAName>		The name or label assigned to this PA. There is one of these names per PA. The value of <i>PA_name</i> can be null. More than one PA can have the same name. In this case, these might or might not correspond to distinct areas in the source.
<Executed>		A list of the statement or lline numbers that were executed. There are zero or more of these per PA.
1 2 5 11 12 13 ...		Zero or more decimal numbers separated by blanks.
</Executed>		
<Unexecuted>		A list of the statement or line numbers that were not executed. There are zero or more of these per PA.
3 4 6 7 8 9 10 ...		Zero or more decimal numbers separated by blanks.
</Unexecuted>		
<BranchBoth>	B	A list of the statement or line numbers that contained conditional branches that were taken both ways (that is, the branch was taken and the branch was not taken). There are zero or more of these per PA.
5 8 11 ...	B	Zero or more decimal numbers separated by blanks.
</BranchBoth>	B	
<BranchFallThru>	B	A list of the statement or line numbers that contained conditional branches that were not taken but were executed. There are zero or more of these per PA.
6 9 12 ...	B	Zero or more decimal numbers separated by blanks.
</BranchFallThru>	B	

XML tag and attributes	Notes	Description
<BranchOnly>	B	A list of the statement or line numbers that contained conditional branches that were taken but never fell through.. There are zero or more of these per PA.
7 13 15 ...	B	Zero or more decimal numbers separated by blanks.
</BranchOnly>	B	
<BranchNeither>	B	A list of the statement or line numbers that contained conditional branches that were not executed.. There are zero or more of these per PA.
29 37 ...	B	Zero or more decimal numbers separated by blanks.
</BranchNeither>	B	
<BranchUncond>	B	A list of the statement or line numbers that contained unconditional branches (that is, branches that cannot fall through). This tag is generated from assembler source code only. (This does not necessarily imply the presence of a BRANCH instruction; it can result from an SVC, SELECT, and so on.). There are zero or more of these per PA.
2 6 ...	B	Zero or more decimal numbers separated by blanks.
</BranchUncond>	B	
<Frequency	F	An indication of how many times the indicated statement or line was executed. Present only if Counts="Frequency" is specified as a ProgramArea attribute. . There are zero or more of these per PA.
Stmt="1"	F	Statement or line number
Count="11"	F	Number of times executed
/>	F	
<BP	P	Detailed information about each breakpoint in the PA. There can be multiple BP records for each statement or line. A BP is generated for all executable statements and is not generated for any non-executable statements. There are zero or more of these per PA.
Branch="No CondTrue CondFalse Always"	P	Indicates whether this breakpoint represents a non-branch statement (No), the "True" path of a conditional branch (CondTrue), the "False" path of a conditional branch (CondFalse), or a branch that always branches (assembler code only).
Offset="1C8X"	P	Offset of the breakpoint in the CSECT.
Optimization="NotInLine InLine"	P	If InLine, this breakpoint is part of in-lined code.
NumberStmts="0 1 nnn"	P	"1" indicates that this is the first breakpoint for this statement or line. "0" indicates that the breakpoint is in the same statement as the previous breakpoint. For assembler source modules only, the value can be any positive integer and indicates the number of executable statements represented by this breakpoint.
Stmt="123"	P	The statement or line number that contains the breakpoint. Omitted if NumberStmts=0 or if Language="Assembler".

XML tag and attributes	Notes	Description
ExecutionCount="123"	P	If Counts="Executed" is in effect, this is 0 if the breakpoint was not executed and 1 if it was executed. If Counts="Frequency" is in effect, this is the number of times the breakpoint was executed.
AsmTargetOffset= "1C4X"	P	When the Frequency option was specified during setup, Language="Assembler", and this breakpoint represents a branch (Branch\!="No"), this is the offset in the target of the branch.
/>	P	
</ProgramArea>		
<ProgramArea> ... </ProgramArea>		
</CSECT>		
<CSECT> ... </CSECT>		
</CompilationUnit>		
<CompilationUnit> ... </CompilationUnit>		
</Listing>		
<Listing> ... </Listing>		
</LoadModule>		
<LoadModule> ... </LoadModule>		
</CoverageFile>		
<p>Notes: When the Notes column contains a letter, the letter corresponds to one of the following notes:</p> <p>B This tag or attribute is present only if Mode="Branch" is specified on the CompilationUnit tag and Branch Analysis data is requested when the XML file is exported.</p> <p>F This tag or attribute is present only if Counts="Frequency" is specified on the CompilationUnit tag and Frequency data is requested when the XML file is exported.</p> <p>P This tag or attribute is present only if Breakpoint Details is requested when the XML file is exported.</p>		

The XML tags and attributes have several types of values:

- Decimal values (such as, "22") represent a single decimal number.
- Hexadecimal values that are followed by a character "X" (such as, "124CX") represent a hexadecimal number. (The "X" will be present as part of the value.)
- Text items that are separated by "|" (such as "COBOL | PLI") represent a choice of keywords (one of the items in the list will be specified as the value).
- Values in italics represent a character string that is appropriate for the item (such as *dsname* indicates that *dsname* is a character string representing a data set name).

All text values are in non-DBCS EBCDIC. DBCS values are not currently supported.

Related references

"XML DTD" on page 215

"Sections of the summary report" on page 113

Statement or line numbers

Within a particular program area and tag group (such as, Executed), statement or line numbers always appears in ascending order. However, statement or line numbers in different tag groups within a ProgramArea might not be. For example, the following is valid:

```
<Executed>60 61 62 66 67 71 72</Executed>  
<Unexecuted>67</Unexecuted>  
<Executed>74</Executed>
```

In most cases, when a tag contains one or more statement or line numbers, each of these numbers represents the statement or line number as shown in the compiler listing. However, assembler language source programs are different. When Language="Assembler" is specified for a Listing, all statement or line numbers are specified in the form *nnnx(nnn)* where the *nnnx* outside the parenthesis is the offset in the containing CSECT and the *nnn* inside the parenthesis indicates the number of executable statements represented by this offset.

Within a ProgramArea tag, the same statement or line number does not appear more than once in the Executed and Unexecuted tags.

The same statement or line number can appear more than once in the Branch Analysis tags. This repetition indicates that the statement contained more than one conditional branch.

Execution of statements with breakpoints

When a statement contains more than one breakpoint, the first breakpoint in the statement is used to determine whether the statement was executed and to compute frequency counts.

When more than one disjoint section of code exists for a statement, the statement is considered to be executed if the first breakpoint in any disjoint section is executed.

XML output for in-lined routines

Processing of code that contains in-lined routines (currently present in C/C++ only) creates the following changes to the normal XML output:

- The statement or line numbers for these in-lined routines can appear in lists such as the Executed, Unexecuted, or BranchBoth lists in multiple ProgramAreas (each program area in which the in-lining occurred as well as the program area corresponding to the out-of-line copy, if present). Each time one of these statement or line numbers appears, it will be suffixed by the letter "I" (such as 121I).
- If an out-of-line copy of the routine exists in C/C++, the statistics that are generated for that ProgramArea are an accumulation of the statistics for all copies (both in-lined and out-of-line) of the routine.

Optional sections

Certain sections of data (such as, <Frequency>...) are output only if the coverage run was set up using the corresponding options and the corresponding data was requested when the XML file was exported.

XML DTD

This section shows the XML DTD that is shipped as member EQACUDTD of the SEQASAMP library. This DTD defines the syntax of all exported XML files. This DTD is referenced in the exported XML file by the name EQACUDTD.dtd. Hence, if you use an XML parser to parse the exported XML file, you must make this SEQASAMP member available under that name on the system where the XML is to be parsed.

```

<?xml version="1.0" encoding="UTF-8"?>

<!--          DTD for CoverageFile Version 1.0          -->

<!-- A CoverageFile consists of RunDate, RunTime, and    -->
<!--          one or more LoadModules                    -->

<!ELEMENT CoverageFile (RunDate, RunTime, LoadModule+)>
  <!ATTLIST CoverageFile Version CDATA #FIXED "1.0">

<!ELEMENT RunDate (Year, Month, Day)>
<!ELEMENT RunTime (Hours, Minutes, Seconds)>

<!-- A LoadModule consists of MemberName and one or more Listing -->

<!ELEMENT LoadModule (MemberName, Listing+)>
<!ELEMENT MemberName (#PCDATA)> <!-- 1 to 8 characters -->

<!-- A Listing consists of DSName and one or more CompilationUnit -->

<!ELEMENT Listing (DSName, CompilationUnit)>
  <!ATTLIST Listing Language (COBOL|PLI|C|Assembler) #REQUIRED>
<!ELEMENT DSName (#PCDATA)> <!-- 1 to 54 characters-->

<!-- A CompilationUnit consists of CreateDate, CreateTime, -->
<!-- optional TestId, and a CSECT                          -->

<!ELEMENT CompilationUnit (CreateDate, CreateTime, TestId?, CSECT)>
  <!ATTLIST CompilationUnit Source (Listing) "Listing"
    Mode (Performance|Branch)
      "Performance"
    Counts (Executed|Frequency) "Executed"
    NumberPAs CDATA #REQUIRED
    NumberBPs CDATA #IMPLIED
    ExecutedBPs CDATA #IMPLIED>
<!ELEMENT CreateDate (Year, Month, Day)>
<!ELEMENT CreateTime (Hours, Minutes, Seconds)>
<!ELEMENT TestId (#PCDATA)> <!-- 1 to 16 characters -->

<!-- A CSECT consists of optional ExtName and one or more -->
<!-- ProgramArea                                          -->

<!ELEMENT CSECT (ExtName?, ProgramArea+)>
<!ELEMENT ExtName (#PCDATA)> <!-- 1 to 1024 characters -->
<!-- A ProgramArea consists of PAName, and zero or more of the -->
<!-- following (in any order): Executed, Unexecuted, BranchBoth, -->
<!-- BranchFallThru, BranchNeither, BranchUncond, Frequency, BP. -->

<!ELEMENT ProgramArea (PAName, (Executed*, Unexecuted*, BranchBoth*,
  BranchOnly*, BranchFallThru*, BranchNeither*,
  BranchUncond*, Frequency*, BP*)*)>
  <!ATTLIST ProgramArea NumberBPs CDATA #IMPLIED
    BPsExecuted CDATA #IMPLIED
    BPExecutions CDATA #IMPLIED>
<!ELEMENT PAName (#PCDATA)> <!-- any number of characters -->
<!ELEMENT Executed (#PCDATA)> <!-- 0 or more decimal numbers
  separated by blanks -->
<!ELEMENT Unexecuted (#PCDATA)> <!-- 0 or more decimal numbers
  separated by blanks -->
<!ELEMENT BranchBoth (#PCDATA)> <!-- 0 or more decimal numbers
  separated by blanks -->
<!ELEMENT BranchOnly (#PCDATA)> <!-- 0 or more decimal numbers
  separated by blanks -->

```

```

<!ELEMENT BranchFallThru (#PCDATA)> <!-- 0 or more decimal numbers
      separated by blanks -->
<!ELEMENT BranchNeither (#PCDATA)> <!-- 0 or more decimal numbers
      separated by blanks -->
<!ELEMENT BranchUncond (#PCDATA)> <!-- 0 or more decimal numbers
      separated by blanks -->
<!ELEMENT Frequency EMPTY>
  <!ATTLIST Frequency Stmt CDATA #REQUIRED
      Count CDATA #REQUIRED>
<!ELEMENT BP EMPTY>
  <!ATTLIST BP Branch (No|Always|CondTrue|CondFalse) "No"
      Offset CDATA #REQUIRED
      Optimization (NotInLine|InLine) "NotInLine"
      NumberStmts CDATA "1"
      Stmt CDATA #IMPLIED
      ExecutionCount CDATA #REQUIRED
      AsmTargetOffset CDATA #IMPLIED>

<!-- Common tags used by multiple elements -->

<!ELEMENT Year (#PCDATA)> <!-- 4 decimal digits -->
<!ELEMENT Month (#PCDATA)> <!-- 2 decimal digits (01 to 12) -->
<!ELEMENT Day (#PCDATA)> <!-- 2 decimal digits (01 to 31) -->

<!ELEMENT Hours (#PCDATA)> <!-- 2 decimal digits (00 to 23) -->
<!ELEMENT Minutes (#PCDATA)> <!-- 2 decimal digits (00 to 59) -->
<!ELEMENT Seconds (#PCDATA)> <!-- 2 decimal digits (00 to 59) -->

```

Example: Exported XML file

This section shows a sample exported XML file that corresponds to the summary report for the example summary report for COB01 in COBOL. This example is for reference purposes only and might not correspond exactly to the file that is exported from the corresponding program.

The coverage run that generated this file was run with the following options:

- Performance Mode Off
- Frequency Count Mode Off
- Debug Mode Off

The XML data was exported using the following options:

- Branch Analysis - YES
- Frequency - n/a (The setting of this option is not applicable, because the coverage run did not collect frequency data.)
- Breakpoint Details - NO

```

<?xml version="1.0" standalone="no"?>
<!DOCTYPE CoverageFile SYSTEM "EQACUDTD.dtd">
<CoverageFile Version="1.0">
  <RunDate>
    <Year>1999</Year>
    <Month>07</Month>
    <Day>22</Day>
  </RunDate>
  <RunTime>
    <Hours>12</Hours>
    <Minutes>57</Minutes>
    <Seconds>19</Seconds>
  </RunTime>
  <LoadModule>
    <MemberName>COB01</MemberName>
    <Listing Language="COBOL">
      <DSName>YOUNG.SAMPLE.COBOLST (COB01A)</DSName>
      <CompilationUnit Source="Listing" Mode="Branch" Counts="Executed" NumberPAs="5">
        <CreateDate>
          <Year>1999</Year>
          <Month>07</Month>
          <Day>10</Day>
        </CreateDate>
        <CreateTime>
          <Hours>07</Hours>
          <Minutes>50</Minutes>
          <Seconds>38</Seconds>
        </CreateTime>
        <CSECT>
          <ExtName>COB01A</ExtName>
          <ProgramArea>
            <PAName>PROG</PAName>
            <Executed>42 45 46 49 50 51</Executed>
            <Unexecuted></Unexecuted>
          </ProgramArea>
          <ProgramArea>
            <PAName>PROGA</PAName>
            <Executed>55 57 62</Executed>
            <Unexecuted>59</Unexecuted>
            <Executed>64</Executed>
            <BranchBoth>55 62</BranchBoth>
            <BranchOnly>57</BranchOnly>
          </ProgramArea>
          <ProgramArea>
            <PAName>PROCA</PAName>
            <Unexecuted>69</Unexecuted>
          </ProgramArea>
          <ProgramArea>
            <PAName>LOOP1</PAName>
            <Executed>72 73 74</Executed>
            <BranchFallThru>72</BranchFallThru>
          </ProgramArea>
          <ProgramArea>
            <PAName>LOOP2</PAName>
            <Executed>77 78</Executed>
            <BranchFallThru>77</BranchFallThru>
          </ProgramArea>
        </CSECT>
      </CompilationUnit>
    </Listing>
  </LoadModule>
</CoverageFile>

```

```

<Listing Language="COBOL">
  <DSName>YOUNG.SAMPLE.COBOLST(COB01B)</DSName>
  <CompilationUnit Source="Listing" Mode="Branch" Counts="Executed" NumberPAs="3">
    <CreateDate>
      <Year>2001</Year>
      <Month>08</Month>
      <Day>10</Day>
    </CreateDate>
    <CreateTime>
      <Hours>07</Hours>
      <Minutes>50</Minutes>
      <Seconds>39</Seconds>
    </CreateTime>
    <CSECT>
      <ExtName>COB01B</ExtName>
      <ProgramArea>
        <PAName>PROGB</PAName>
        <Executed>33 35 36 39 42</Executed>
        <Unexecuted>41</Unexecuted>
        <BranchBoth>33</BranchBoth>
        <BranchFallThru>35</BranchFallThru>
        <BranchOnly>39</BranchOnly>
      </ProgramArea>
      <ProgramArea>
        <PAName>PROCB</PAName>
        <Executed>46</Executed>
      </ProgramArea>
      <ProgramArea>
        <PAName>LOOP1</PAName>
        <Executed>49 50 51</Executed>
        <BranchFallThru>49</BranchFallThru>
      </ProgramArea>
    </CSECT>
  </CompilationUnit>
</Listing>
<Listing Language="COBOL">
  <DSName>YOUNG.SAMPLE.COBOLST(COB01C)</DSName>
  <CompilationUnit Source="Listing" Mode="Branch" Counts="Executed" NumberPAs="4">
    <CreateDate>
      <Year>2001</Year>
      <Month>08</Month>
      <Day>10</Day>
    </CreateDate>
    <CreateTime>
      <Hours>07</Hours>
      <Minutes>50</Minutes>
      <Seconds>40</Seconds>
    </CreateTime>
    <CSECT>
      <ExtName>COB01C</ExtName>
      <ProgramArea>
        <PAName>PROGC</PAName>
        <Executed>33 35 36 39 40</Executed>
        <BranchBoth>33 35 39</BranchBoth>
      </ProgramArea>
      <ProgramArea>
        <PAName>PROCC</PAName>
        <Executed>44 45</Executed>
        <Unexecuted>47</Unexecuted>
        <BranchOnly>45</BranchOnly>
      </ProgramArea>
    </CSECT>
  </CompilationUnit>
</Listing>

```

```

    <ProgramArea>
      <PAName>LOOP1</PAName>
      <Executed>52 53 54</Executed>
      <Unexecuted>56</Unexecuted>
      <BranchFallThru>52</BranchFallThru>
      <BranchOnly>54</BranchOnly>
    </ProgramArea>
    <ProgramArea>
      <PAName>LOOP2</PAName>
      <Executed>60 61</Executed>
      <BranchFallThru>60</BranchFallThru>
    </ProgramArea>
  </CSECT>
</CompilationUnit>
</Listing>
<Listing Language="COBOL">
  <DSName>YOUNG.SAMPLE.COBOLST(COB01D)</DSName>
  <CompilationUnit Source="Listing" Mode="Branch" Counts="Executed" NumberPAs="3">
    <CreateDate>
      <Year>2001</Year>
      <Month>08</Month>
      <Day>10</Day>
    </CreateDate>
    <CreateTime>
      <Hours>07</Hours>
      <Minutes>50</Minutes>
      <Seconds>42</Seconds>
    </CreateTime>
    <CSECT>
      <ExtName>COB01D</ExtName>
      <ProgramArea>
        <PAName>PROGD</PAName>
        <Unexecuted>33 35 36 38</Unexecuted>
        <BranchNeither>33 35</BranchNeither>
      </ProgramArea>
      <ProgramArea>
        <PAName>PROCD</PAName>
        <Unexecuted>42</Unexecuted>
      </ProgramArea>
      <ProgramArea>
        <PAName>LOOP1</PAName>
        <Unexecuted>46</Unexecuted>
      </ProgramArea>
    </CSECT>
  </CompilationUnit>
</Listing>
</LoadModule>
</CoverageFile>

```

Related references

“Example: COBOL summary report” on page 115

Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The accessibility features in z/OS provide accessibility for Debug Tool.

The major accessibility features in z/OS enable users to:

- Use assistive technology products such as screen readers and screen magnifier software
- Operate specific or equivalent features by using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products work with the user interfaces that are found in z/OS. For specific guidance information, consult the documentation for the assistive technology product that you use to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces by using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Volume 1* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Accessibility of this document

Information in the following formats of this document is accessible to visually impaired individuals who use a screen reader:

- PDF format when viewed with Adobe Acrobat Reader 5.0 or later
- BookManager[®] format when viewed with IBM BookManager BookServer (except for syntax diagrams)

Syntax diagrams start with the word **Format** or the word **Fragments**. Each diagram is preceded by two images. For the first image, the screen reader will say "Read syntax diagram". The associated link leads to an accessible text diagram. When you return to the document at the second image, the screen reader will say "Skip visual syntax diagram" and has a link to skip around the visible diagram.

For BookManager users only: A screen reader might say the lines, symbols, and words in a diagram, but not in a meaningful way. For example, you might hear "question question dash dash MOVE dash dash plus dash dash literal-1 dash dash plus" for part of the MOVE statement. You can enter **Say Next Paragraph** to move quickly through syntax diagrams if your screen reader has that capability.

Notices

This information was developed for products and services offered in the U.S.A. IBM might not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with the local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement might not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Copyright license

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or functions of these programs.

Programming interface information

This book is intended to help you debug application programs. This publication documents intended Programming Interfaces that allow you to write programs to obtain the services of Debug Tool.

Trademarks and service marks

The following terms, denoted by an asterisk (*) on the first occurrence in this publication, are trademarks or service marks of International Business Machines Corporation in the United States or other countries:

CICS
DB2
IBM
MVS
MVS/ESA
OS/390
VisualAge
z/OS

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Java™ and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

LINUX is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and Windows NT® are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a trademark of UNIX System Laboratories, licensed exclusively by X/Open Company, Ltd.

Bibliography

Debug Tool publications

Using CODE/370 with VS COBOL II and OS PL/I,
SC09-1862

Debug Tool for z/OS

Debug Tool Summary of Commands and Built-in Functions, SC19-1074

Debug Tool Customization Guide, SC19-1075

Program Directory for Debug Tool for z/OS,
GI10-8726

Debug Tool Reference and Messages, GC19-1073

Debug Tool User's Guide, SC19-1071

Debug Tool Utilities and Advanced Functions for z/OS

Debug Tool Coverage Utility Users Guide,
SC19-1072

Program Directory for Debug Tool Utilities and Advanced Functions for z/OS, GI10-8727

High level language publications

OS/390 C/C++

Compiler and Run-Time Migration Guide,
SC09-2359

Curses, SC28-1907

Language Reference, SC09-2360

Programming Guide, SC09-2362

Reference Summary, SX09-1313

Run-Time Library Reference, SC28-1663

User's Guide, SC09-2361

Enterprise COBOL for z/OS and OS/390

Migration Guide, GC27-1409

Customization, GC27-1410

Licensed Program Specifications, GC27-1411

Language Reference, SC27-1408

Programming Guide, SC27-1412

COBOL for OS/390 & VM

Compiler and Run-Time Migration Guide,
GC26-4764

Customization under OS/390, GC26-9045

Language Reference, SC26-9046

Programming Guide, SC26-9049

Enterprise PL/I for z/OS and OS/390

Diagnosis Guide, SC27-1459

Language Reference, SC27-1460

Licensed Program Specifications, GC27-1456

Messages and Codes, SC27-1461

Migration Guide, GC27-1458

Programming Guide, SC27-1457

VisualAge PL/I for OS/390

Compiler and Run-Time Migration Guide,
SC26-9474

Diagnosis Guide, SC26-9475

Language Reference, SC26-9476

Licensed Program Specifications, GC26-9471

Messages and Codes, SC26-9478

Programming Guide, SC26-9473

PL/I for MVS & VM

Compile-Time Messages and Codes, SC26-3229

Compiler and Run-Time Migration Guide,
SC26-3118

Diagnosis Guide, SC26-3149

Installation and Customization under MVS,
SC26-3119

Language Reference, SC26-3114

Licensed Program Specifications, GC26-3116

Programming Guide, SC26-3113

Reference Summary, SX26-3821

Related publications

CICS

Application Programming Guide, SC34-6231

Application Programming Primer, SC34-0674

Application Programming Reference, SC34-6232

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™ Application Programming: Database Manager, SC27-1286
IMS Application Programming: EXEC DLI Commands for CICS & IMS, SC27-1288
IMS Application Programming: Transaction Manager, SC27-1289

TSO/E

Command Reference, SA22-7782
Programming Guide, SA22-7788
System Programming Command Reference, SA22-7793
User's Guide, SA22-7794

z/OS

MVS JCL Reference, SA22-7597
MVS JCL User's Guide, SA22-7598
MVS System Commands, SA22-7627

z/OS Language Environment®

Concepts Guide, SA22-7567
Customization, SA22-7564
Debugging Guide, GA22-7560
Programming Guide, SA22-7561
Programming Reference, SA22-7562
Run-Time Migration Guide, GA22-7565
Vendor Interfaces, SA22-7568
Writing Interlanguage Communication Applications, SA22-7563

Softcopy publications

Online publications are distributed on CD-ROMs and can be ordered through your IBM representative. *Debug Tool User's Guide*, *Debug Tool Customization Guide*, and *Debug Tool Reference and Messages* are distributed on the following collection kit:

SK3T-4269

Online publications can also be downloaded from the IBM Web site. Visit the IBM Web site for each product to find online publications for that product.

Glossary

This glossary defines terminology and acronyms that are unique to this document or not commonly known.

A

annotated listing report. An annotated listing that shows which code statements have been executed.

APF-authorized. Authorized program facility.

B

background execution. The execution of lower-priority computer programs when higher-priority programs are not using the system resources. Contrast with *foreground execution*.

BP. See *breakpoint*.

breakpoint (BP). A point in a computer program where an instruction op code is replaced by a user SVC instruction so that the Coverage Utility monitor gets control from the operating system.

breakpoint library. A library (partitioned data set) of breakpoint data for your listings. Each member includes the data for one listing.

BRKOUT. The ddname of the file of test case coverage results (breakpoint output) that is created when Coverage Utility runs and used when Coverage Utility creates reports.

BRKTAB. The ddname of the file of breakpoint data (breakpoint table) that is created when Coverage Utility is set up and used when Coverage Utility runs.

C

code coverage. A measurement of the number of code statements that have been executed.

compile unit (CU). The programs that are contained within one compiler listing.

control file. A file that contains information describing the compile units to be analyzed.

CU. See *compile unit*.

D

DBCS. See *double-byte character set*.

ddname. The symbolic representation for a name placed in the name field of a DD statement.

double-byte character set. A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Because each character requires 2 bytes, the typing, display, and printing of DBCS characters requires hardware and programs that support DBCS. Contrast with *single-byte character set*.

dsname. Data set name.

E

EBCDIC. Extended binary-coded decimal interchange code. A coded character set of 256 8-bit characters.

ECSA. Extended Common System Area. A major element of MVS/ESA™ virtual storage above the 16MB line. This area contains pageable system data areas that are addressable by all active virtual storage address spaces. It duplicates the common system area (CSA) which exists below the 16MB line.

EQACUOBP. An execution monitor command that displays the status of breakpoints.

EQACUOCM. The Coverage Utility program that monitors your program while it is being run to collect test case coverage statistics.

EQACUOID. An execution monitor command that enables you to add a unique test case ID.

EQACUOPF. An execution monitor command that turns the monitor performance mode off.

EQACUOPN. An execution monitor command that turns the monitor performance mode on.

EQACUOQT. An execution monitor command that functions like the EQACUOSP command. However, unlike the EQACUOSP command, EQACUOQT does not write output to disk. Contrast with *EQACUOSP*.

EQACUORE. An execution monitor command that resets all statistics in the current monitor session to zero.

EQACUOSA. An execution monitor command that allows you to select the session ID, listing number, and program areas (PAs) for which you want statistics to be displayed.

EQACUOSE. An execution monitor command that displays a list of the current active sessions.

EQACUOSL. An execution monitor command that enables you to select listings for which to display statistics.

EQACUOSN. An execution monitor command that writes the coverage statistics (BRKOUT) to the data set name that is specified on the panel.

EQACUOSP. An execution monitor command that writes current statistics to disk and terminates the monitor session. Contrast with *EQACUOQT*.

EQQCUSET. A program that analyzes the compiler listings to determine breakpoint placement.

EQACUZPL. A program that builds input for EQACUZPP to insert breakpoints into load modules.

EQACUZPP. A program that uses breakpoint data to modify load modules by inserting breakpoints.

EQACUZPT. A program that uses breakpoint data to modify object modules by inserting breakpoints.

export. The process of saving (coverage) data in a format that can be processed by other programs.

F

foreground execution. The execution of a computer program that preempts the use of computer facilities.

I

instrument. To overlay instructions in a user program with special instructions, known as breakpoints, that gather test coverage statistics about the program.

J

jcldsn. JCL data set name.

L

LISTINA. The ddname of the Assembler H or High Level Assembler listing file that is used in setup or reports.

LISTINB. The ddname of the COBOL assembler listing file that is used in setup or reports.

LISTINP. The ddname of the PL/I or C assembler listing file that is used in setup or reports.

M

monitor. The program that measures test case coverage during execution of your programs.

monitor session. A distinct invocation of the monitor program.

O

op code. Operation code. A code for representing the operation parts of the machine instructions of a computer.

P

PA. See *program area*.

program area (PA). Each specific PA contains all of the breakpoints for one COBOL paragraph; PL/I procedure, ON-unit, or Begin-block; C function; or assembler CSECT.

R

reentrant program. A computer program that can be entered at any time before any prior execution of the program has been completed.

report. The Coverage Utility job that produces the summary and annotated listing reports after a test case run.

S

SBCS. See *single-byte character set*.

session. A distinct invocation of the monitor program.

session ID. The identification of your session to the monitor program. This ID defaults to your TSO user ID.

setup. The Coverage Utility job that analyzes your assembler listings in order to produce a table of breakpoint data and insert breakpoints into disk resident programs.

single-byte character set (SBCS). A character set in which each character is represented by a one-byte code. Contrast with *double-byte character set*.

SQA. See *system queue area*.

summary report. A Coverage Utility report that gives statistics about the coverage of all program areas (PAs) during the test run.

supervisor call (SVC). A request that serves as the interface into operating system functions, such as

allocating storage. The SVC protects the operating system from inappropriate user entry. All operating system requests must be handled by SVCs.

SVC. See *supervisor call*.

system queue area (SQA). An area of MVS storage that is used for running authorized programs or for storage allocation.

U

user defaults. Default settings that affect only your personal Coverage Utility sessions. You can change these defaults by using the Coverage Utility panels. Contrast with *site defaults*.

X

XML (Extended Markup Language). A data format that is designed to enable data to be self-defining and portable across many applications.

Index

A

ABEND codes 159
annotated listing report
 C/C++ 29
 creating JCL for 30
 creating specific 123
 displaying execution counts 126
 frequency count mode 126
 PL/I 29
 reducing size 124
annotation symbols
 annotated listing report 29, 123
 changes with performance mode 125
assembler samples 44
authorized data sets 159

B

branch, conditional
 breakpoints 89, 96
 coverage 5, 113
 coverage statistics 115
 instruction 89, 123
breakpoints
 definition 4
 displaying status 86
 errors executing 159
 purpose 5
BRKOUT
 definition 5
 specifying data set name 99
BRKTAB
 creating 61
 definition 5

C

C/C++
 annotated listing report 29
 listing DCB attributes 69
 program identification 113
 statements 56
 summary report 21, 117
CL/C++ samples 44
COBOL samples 43
code motion optimization 135
coder, definition 153
combine
 edit screen 157
 JCL, creating 156
 panel 156
command
 syntax diagrams ix
commands
 issuing 85
 monitor 86
 parameters 85, 101
compiler options 65
compilers supported 6

conditional branch
 breakpoints 89, 96
 coverage 5
 coverage suppression 119
 instruction
 annotated listing report 123
 fields 89
 symbols 29
conditional branch coverage
 overhead 82
 poor performance when
 measuring 161
 suppressing with performance
 mode 119
control file
 ASM sample 21
 C/C++ sample 21, 28
 COBOL sample 21
 combined 49
 contents 50
 creating 49
 function of 49
 load module example 60
 PL/I sample 21, 27
 statement syntax 51
coverage
 conditional branch, measuring 161
 overhead 82
 suppressing conditional branch 119
Coverage Utility
 See control file
Coverage Utility information
 commands
 EQACUOBP 86
 EQACUOID 89
 EQACUOPF 90
 EQACUOPN 91
 EQACUOQT 92
 EQACUORE 93
 EQACUOSA 94
 EQACUOSE 96
 EQACUOSL 97
 EQACUOSN 99
 EQACUOSP 100
 issuing commands 85
 monitor
 commands 85, 101
 execution 77, 83
 problems 159
 setup 61, 73
 CPP statement 56
 customizing Coverage Utility 9

D

data set attributes 189
date format, specifying 206
DBCS support 191
dead code elimination optimization 135
defaults file, creating and modifying 9
defaults, modifying 9

defaults, user 9
diagnosing problems
 See also error messages
 047 system abend 159
 7C1 system abend in user
 program 159
 conditional branch coverage, poor
 performance 161
 ECSA storage depleted 161
 SQA storage depleted 161
documents, licensed vii
DTSU
 See Debug Tool Setup Utility

E

ECSA storage depleted 161
ECSA usage 188
English, specifying 208
English, specifying uppercase 208
ENU 208
EQACUOBP 86
EQACUOID 89
EQACUOPF 90
EQACUOPN 91
EQACUOQT 92
EQACUORE 93
EQACUOSA 94
EQACUOSE 96
EQACUOSL 97
EQACUOSN 99
EQACUOSP 100
EQACUSET 70
EQACUZPL parameters 72
EQACUZPP parameters 72
EQACUZPT parameters 72
EQASTART 9
 parameters 205
executing
 See running
execution
 Coverage Utility overview 5
 monitor 77
 multiple user sessions 80
 summarized 5
exported data 209
exporting data
 in XML format 109
 parameters 140

F

FastPath
 overview 193
 quick start
 JCL 193
 parameters 195
 quick stop
 JCL 200
 parameters 201

FastPath (*continued*)
 snapshot summary JCL 198
 frequency count mode 126

I

inlined code
 in annotated report 125
 in summary report 67
 setting default 12
inlining 137
introducing
 Debug Tool Coverage Utility
 (Coverage Utility) 3
invoking
 See starting

J

Japanese, specifying 208
JCL
 Coverage Utility
 creating combine 156
 creating report 106
 for assembler examples 21
 for C/C++ examples 21
 for COBOL examples 31
 for PL/I examples 21
 setup
 creating monitor 77
 creating setup 63
 creating setup for compile job
 stream 70
JPN 208

K

KOR 208
Korean, specifying 208

L

language, specifying national 208
languages, supported 3
licensed documents vii
LINECOUNT parameter 206
listing report, annotated
 example
 assembler 37, 123
 C/C++ 35
 COBOL 32, 123
 PL/I 34, 123
 producing 29
load modules, instrumenting 62
LOCALE parameter 206
LookAt message retrieval tool viii

M

message retrieval tool, LookAt viii
module, definition 153
monitor program
 commands 85
 Coverage Utility 5
 execution 77

monitor program (*continued*)
 problems, diagnosing 159
 SQA, CSA, and ECSA usage 188
 multiple user sessions 77

N

national language, specifying 208
NATLANG parameter 208
numeric format, specifying 206

O

object modules, instrumenting 62
operating systems supported 6
optimization techniques
 code motion 135
 dead code elimination 135
 procedure inlining 136
 statement decomposition 136
optimized code 67, 135
options, compiler 65
overhead 82
overview, Coverage Utility 3

P

page size 206
panel interface 3
panels
 Debug Tool Coverage Utility panel 9
 Defaults 10
 Manipulate Defaults 10
 Reset Defaults to Site Defaults 13
parameters, common
 definition 205
 EQACUOBP 88
 EQACUOPF 91
 EQACUOPN 92
 EQACUOQT 93
 EQACUORE 94
 EQACUOSA 95
 EQACUOSE 97
 EQACUOSL 98
 EQACUOSN 99
 EQACUOSP 101
 EQACUSET 71
 EQACUZPL 72
 EQACUZPP 73
 EQACUZPT 72
 export program 141
 monitor 79
 overview 9
 report program 140
 summary program 139
performance mode
 changes in annotation symbols 125
 reducing monitor overhead 82
 suppressing conditional branch
 coverage 119
PL/I samples 43
problem diagnosis
 See also error messages
 047 system abend 159
 7C1 system abend in user
 program 159

problem diagnosis (*continued*)
 conditional branch coverage, poor
 performance 161
 ECSA storage depleted 161
 SQA storage depleted 161
 procedure inlining optimization 136
 program area
 data 114
 definition 17
 displaying statistics for 94
 project environments 153

R

report program parameters 140
reports
 annotated listing report 123
 from specific listings 123
 printing 124
 process overview 5
 processing optimized code 135
 requirements for running reports 5
 summary for assembler 113
 summary report 113
requirements, Coverage Utility 187, 191
resources required 187
results
 combining 155, 157
 rules used 157

S

sample
 Coverage Utility 17
 summary of test case coverage 22
samples
 assembler 44
 assembler summary 29
 C/C++ 44
 C/C++ summary 28
 COBOL 43
 COBOL summary 26
 compiling 21
 control file 21
 data sets 19
 link & run JCL 24
 monitor JCL 23
 PL/I 43
 PL/I summary 27
 producing summary 22
 report 19
 running 17
 running annotated JCL 30
 running summary JCL 24
 setup JCL 22
 summary report JCL 23
 using 17
sessions
 definition 80
 determining active sessions 96
 multiple 80
setup
 creating JCL using the panels 63
 for summary and listings 61
 JCL for the compile job stream 70
 overview 4

- setup (*continued*)
 - parameters
 - ATOGZAPL 72
 - EQACUSET 70
 - EQACUZPP 72
 - EQACUZPT 72
 - overview 70
 - requirements for setup 4
 - summarized 4
 - when to create or submit JCL 65
- setup program
 - parameters 70
 - resources required 187
- SQA storage depleted 161
- SQA usage 188
- starting Coverage Utility 9
- statement decomposition 136
- statistics, resetting 93
- stopping the monitor 100
- summary program
 - parameters 139
 - resources required 188
- summary test case coverage
 - assembler
 - COBOL 22
 - sample 44
 - setup 4
 - summary 21
 - C/C++ 21
 - COBOL
 - create JCL to start a monitor
 - session 23
 - create setup JCL 22
 - create summary JCL 23
 - edit JCL to link the modified object
 - modules 24
 - edit JCL to run the GO step 24
 - editing the control file 21
 - run the JCL 24
 - PL/I 21
- supervisor call (SVC)
 - definition 228
 - problems, diagnosing 159
 - used as breakpoints 77
- SVC
 - definition 228
 - problems, diagnosing 159
 - used as breakpoints 77
- symbols, annotation
 - annotated listing report 29, 123
 - changes with performance mode 125
- syntax
 - ASM 58
 - C 56
 - COBOL 53
 - DEFAULTS 52
 - examples 60
 - INCLUDE 52
 - PL/I 55
- syntax diagrams
 - how to read ix
- system ABEND codes 159
- system considerations 6

T

- terminating Coverage Utility
 - EQACUOQT command 92
 - EQACUOSP command 100
- test cases
 - combining results 155
 - individual, measuring coverage 158
 - rules used 155
 - specifying ID 89
- tester, definition 153
- time format, specifying 206
- timeout, monitor 93

U

- UEN 208
- unexecuted 114
- user data sets for samples 41
- user defaults
 - editing 10
 - panel 10
 - resetting 13
- user SVCs 5

V

- VisualAge PL/I load module
 - breakpoints 61

W

- workstation debugging
 - See* remote debug mode

X

- XML
 - exporting data in 109
 - file format 209
 - format data 209

Readers' Comments — We'd Like to Hear from You

Debug Tool Utilities and Advanced Functions for z/OS
Coverage Utility User's Guide and Messages
Version 7.1

Publication No. SC19-1072-00

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



Fold and Tape

Please do not staple

Fold and Tape



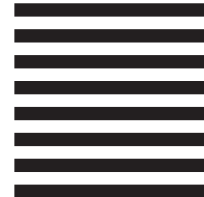
NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Reader Comments
DTX/E269
555 Bailey Ave.
San Jose, CA
95141-9989



Fold and Tape

Please do not staple

Fold and Tape



Program Number: 5655-R45

Printed in USA

SC19-1072-00

